IBM® Application Testing Collection for MVS/ESA™
Version 1 Release 5 Modification 0
User's Guide
Program Number: 5799-GBN
PRPQ: P85579

April 1999

Note!

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

This book is also available as an online book that can be viewed with:

- The IBM® BookManager® READ and IBM Library Reader™ licensed programs. The IBM Library Reader is available for downloading from the following Web site:
  http://booksrv2.raleigh.ibm.com/homepage/ilrserv.html
- Adobe Acrobat Reader 2.1 and later, which is available for downloading from the Adobe Web site.

Eighth Edition (April 1999)

This edition applies to Version 1 Release 5 Modification 0 of the IBM Application Testing Collection for MVS/ESA, program number 5799-GBN. Changes are made periodically to the information herein.

This publication is available in downloadable form from the IBM Year 2000 Technical Support Center Web site:
http://www.software.ibm.com/year2000/
Select the **Testing** link on the main home page and then look for the Application Testing Collection on the page displayed.

This publication is provided on an *as is* basis. Although it has been thoroughly edited, it may nevertheless contain inaccuracies.

© **Copyright International Business Machines Corporation 1997, 1999. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Table of Contents

Notices . . . xv
Trademarks . . . xv

Preface . . . xvii
Who Should Read This Book . . . xvii
How This Book Is Organized . . . xvii
Conventions for Words and Type . . . xviii
How to Read Syntax Diagrams . . . xviii
Related Information . . . xx
Getting Help . . . xx

Summary of Changes (V1R5M0) . . . xxi
Coverage Assistant (CA) . . . xxi
Coverage Assistant (CA), Distillation Assistant (DA), and Unit Test Assistant (UTA) . . . xxi
Summary of Changes (V1R4M0) . . . xxi
Coverage Assistant (CA) . . . xxi
Unit Test Assistant (UTA) . . . xxi
Source Audit Assistant (SAA) . . . xxi
Automated Regression Testing Tool (ARTT) . . . xxii
Summary of Changes (V1R3M4) . . . xxii
Coverage Assistant (CA), Unit Test Assistant (UTA) and Distillation Assistant (DA) . . . xxii
Source Audit Assistant (SAA) . . . xxii
Summary of Changes (V1R3M0) . . . xxii
Coverage Assistant (CA) . . . xxii
Unit Test Assistant (UTA) . . . xxii
ATC User Documentation . . . xxiii
Summary of Changes (V1R2M4) . . . xxiii
Coverage Assistant (CA) and Distillation Assistant (DA) . . . xxiii
General . . . xxiii

Introducing the IBM Application Testing Collection . . . 1

Overview . . . 3

Installing and Customizing ATC . . . 5

Prerequisites and Supported Software . . . 7

Converting Existing ATC Systems and Users to the Current Release . . . 9
V1R4M0 Systems and Users to V1R5M0 . . . 9
V1R3M4 Systems and Users to V1R5M0 . . . 9
V1R3M0 Systems and Users to V1R5M0 . . . 10
V1R2M4 Systems and Users to V1R5M0 . . . 10
V1R2M0 Systems and Users to V1R5M0 . . . 11

System Installation . . . 13
Installing Data Sets . . . 13

© Copyright IBM Corp. 1997, 1999

Modifying the FORMS, REXX, and BKSHELF Data Sets . . . 15
Setting up the Authorized Data Sets . . . 16
Installing and Enabling the Monitor SVCs . . . 18
Ensuring Users Have Access to AMASPZAP . . . 20
Editing the Site Defaults Data Set . . . 20
Verifying the Installation . . . 25

Basic User Setup . . . 27
Modifying Your ATC Defaults . . . 27
Editing Your User Defaults . . . 28
Resetting Your User Defaults to the System Defaults . . . 32
Allocating Data Sets and Testing Installation . . . 33
Accessing the Language Environment Runtime Library . . . 36
Running the Sample Test Cases . . . 36

Using Coverage Assistant . . . 37

Introduction . . . 39
What Is Coverage Assistant? . . . 39
What Does CA Require? . . . 40
How Does CA Work? . . . 40
Setup . . . 42
Execution . . . 42
Report . . . 43
Where Can You Get Further Details? . . . 43

Coverage Assistant Samples . . . 45
COBOL Summary of Test Case Coverage . . . 47
Edit the CA Control File . . . 49
Create Setup JCL . . . 50
Create JCL to Start a Monitor Session . . . 51
Create JCL for a Summary Report . . . 52
Create JCL to Link the Modified Object Modules . . . 52
Create JCL to Run the GO Step . . . 52
Execute the JCL . . . 53
Annotated COBOL Listings . . . 54
Edit the CA Control File . . . 57
Create Setup JCL . . . 57
Create JCL To Start a Monitor Session . . . 57
Create JCL for an Annotated Listing . . . 57
Create JCL to Link the Modified Object Modules . . . 58
Create JCL to Run the GO Step . . . 58
Execute the JCL . . . 58
PL/I Summary of Test Case Coverage . . . 59
Edit the CA Control File . . . 61
Create Setup JCL . . . 62
Create JCL to Start a Monitor Session . . . 62
Create JCL for a Summary Report . . . 63
Create JCL to Link the Modified Object Modules . . . 63
Create JCL to Run the GO Step . . . 63
Execute the JCL . . . 64
Annotated PL/I Listings . . . 64
Edit the CA Control File . . . 66

Create Setup JCL . . . 66
Create JCL to Start a Monitor Session . . . 66
Create JCL for an Annotated Listing . . . 66
Create JCL to Link the Modified Object Modules . . . 67
Create JCL to Run the GO Step . . . 67
Execute the JCL . . . 67
Assembler Summary of Test Case Coverage . . . 68
Edit the CA Control File . . . 71
Create Setup JCL . . . 72
Create JCL to Start a Monitor Session . . . 73
Create Summary Report JCL . . . 73
Create JCL to Link the Modified Object Modules . . . 74
Create JCL to Run the GO Step . . . 74
Execute the JCL . . . 74
Annotated Assembler Listings . . . 75
Edit the CA Control File . . . 79
Create Setup JCL . . . 79
Create JCL to Start a Monitor Session . . . 79
Create JCL for an Annotated Listing . . . 79
Create JCL to Link the Modified Object Modules . . . 79
Create JCL to Run the GO Step . . . 79
Execute the JCL . . . 80

Editing the Coverage Assistant Control File . . . 81
Contents of the Control File . . . 81

Coverage Assistant Reports . . . 83
Summary Coverage Report . . . 83
Areas of the Report . . . 86
Suppression of Conditional Branch Coverage with Performance Mode . . . 87
Examples of Reports Using Listings . . . 88
Summary Report for Assembler . . . 88
Areas of the Report . . . 88
Creating Coverage Reports . . . 90
Creating Summary and Annotated Listing Report JCL Using the Panels . . . 91
Creating JCL for a Summary without Annotated Listings . . . 94
Annotated Listing Coverage Report . . . 95
Selecting Specific Listings to Annotate . . . 101
Reducing the Size of Annotated Listings . . . 101
Displaying Execution Counts in an Annotated Listing . . . 102
Differences in CA Reports When DA and UTA Are Used . . . 103
Changes in Annotation Symbols with Performance Mode . . . 104
Parameters for the SUMMARY and REPORT Programs . . . 104
SUMMARY . . . 104
REPORT . . . 105
Printing Reports . . . 105
Targeted Summary Reports . . . 105
Target Control File . . . 106
Creating Targeted Summary Reports . . . 112

Using Coverage Assistant in a Large Project Environment . . . 115
Creating CA Files during Code Development . . . 115
Breakpoint Data . . . 116
Test Case Coverage Results . . . 116

Combining Test Case Coverage Results . . . 116
Creating the Combine JCL Using the Panels . . . 117
Rules for Combining Results . . . 120
Sample Combine Test Case . . . 120
Measuring Coverage for Individual Test Cases . . . 121

Using Distillation Assistant . . . 123

Introduction . . . 125
What Is Distillation Assistant? . . . 125
What Does DA Require? . . . 126
Input Master Data Set Restrictions . . . 126
Logical Distillation Requirements . . . 127
How Does DA Work? . . . 127
Setup . . . 130
Execution . . . 130
Physical Distillation . . . 130
Where Can You Get Further Details? . . . 131

Distillation Assistant Samples . . . 133
COB11x and PLI11x Test Cases . . . 135

Logical Distillation . . . 141
Description of Reading Input Data Sets . . . 141
Coverage of the Distilled Data Set . . . 141
PL/I ON-Units . . . 141
How Much Data Can Be Read . . . 142
Recording Which Keys Execute a Statement . . . 142

Editing the Distillation Assistant Control File . . . 145
Contents of the Control File . . . 145
Examples . . . 147

Physical Distillation . . . 149
Physical Distillation Summary . . . 149
Parameters Used by Physical Distillation . . . 150
Running Physical Distillation . . . 151
Generating JCL for Physical Distillation . . . 152
Editing Distillation JCL . . . 156
Submitting JCL for Physical Distillation . . . 157
Physical Distillation Return Codes . . . 158

Using Unit Test Assistant . . . 159

Introduction . . . 161
What Is Unit Test Assistant? . . . 161
What Does UTA Require? . . . 162
How Does UTA Work? . . . 162
Setup . . . 165
Execution . . . 165
Report . . . 166
Where Can You Get Further Details? . . . 166

Unit Test Assistant Samples . . . 167
COB02x Test Cases . . . 169
Multiple Compile Unit Test Case (COB01x) . . . 176

Unit Test Assistant Read and Warp Descriptions . . . 183
Description of the Variable Read Operation . . . 183
Where a Variable Is Read . . . 183
Which COBOL Storage Areas Can Be Read . . . 183
How Much Data Can Be Read . . . 184
Description of the Variable Warp Operation . . . 184
What COBOL Variables Can Be Warped . . . 184
When a COBOL Variable Is Warped . . . 185
What PL/I Variables Can Be Warped . . . 185
When a PL/I Variable Is Warped . . . 185
Reading and Warping on the Same Statement . . . 185

Editing the Unit Test Assistant Control File . . . 187
Contents of the Control File . . . 187
Examples . . . 188

Unit Test Assistant Reports . . . 189
Monitored Variables Report (MVR) . . . 189
Variable Data Report (VDR) . . . 191
Errors During Data Warping . . . 192
Combined Variable Data Report (CVDR) . . . 193
Creating Unit Test Report JCL Using the Panels . . . 194
Examples of Reports . . . 196
Parameters for the PRINTVAR Program . . . 196

Unit Test Assistant File Warping . . . 197
File Warp Operation . . . 197
File Warp Samples . . . 199
File Warp Control File Syntax . . . 200
Field Definition . . . 201
Group Level Definition . . . 201
Record Type Definition . . . 202
Positional Parameter Definitions . . . 203
Control File Example . . . 205
File Warp Return Codes . . . 206

Common CA, DA, and UTA Information . . . 207

CA, DA, and UTA Control File . . . 209
Contents of the Control File . . . 211
Control File Statement Syntax . . . 212
Control File Examples . . . 228

CA, DA, and UTA Setup . . . 231
Setup . . . 231
Compiler Options . . . 231
Instrumentation of Load Modules instead of Object Modules . . . 238
Creating the Setup JCL Using the Panels . . . 239
When to Create or Submit Setup JCL . . . 241

Setup JCL for the Compile Job Stream . . . 241
Parameters for SETUP and ZAPTXT Programs . . . 241
SETUP . . . 241
ZAPTXT . . . 243

Monitor Execution . . . 245
Creating the Monitor JCL Using the Panels . . . 246
Parameters for Start Monitor (CMDUSVC) . . . 249
Parameters for Variable Monitor (VARMON3) . . . 249
Multiple User Sessions . . . 249
Coverage of Common Modules with Multiple User Sessions . . . 250
Using the Performance Mode to Reduce Monitor Overhead . . . 253
Buffer Monitor . . . 254

Monitor Commands . . . 255
Issuing Commands . . . 255
CABPDSP . . . 256
CADATA . . . 259
CAIDADD . . . 261
CALIST . . . 262
CAPRFOFF . . . 263
CAPRFON . . . 264
CAQUIT . . . 265
CARESET . . . 266
CASESSN . . . 267
CASTATS . . . 268
CASTOP . . . 270
CAVADSP . . . 272

Diagnosing Monitor Problems . . . 275
047 Abend . . . 275
Operation Exception (0C1) on User Program . . . 275
System Interruption Code of Fnn on User Program . . . 275
Lack of SQA Space . . . 276
Poor Performance When Measuring Conditional Branch Coverage . . . 276

Using Source Audit Assistant . . . 277

Introduction . . . 279
What Is Source Audit Assistant? . . . 279
What Does SAA Require? . . . 279
How Does SAA Work? . . . 280

Source Audit Assistant Samples . . . 281
Sample Data Sets . . . 281
Execution and Verification . . . 282
Standard Group - Background Execution . . . 283
Standard Group - Foreground Execution . . . 285
Target Group . . . 287

Starting Source Audit Assistant . . . 289
Executing SAA in the Background . . . 289
Executing SAA in the Foreground . . . 293

Executing the SAA Postprocessor 296

Understanding the Comparison Report 299
Areas of the Comparison Report 299

Source Audit Assistant Postprocessor 301
SAA Postprocessor Inputs 301
SAA Postprocessor Outputs 302

Reserved Data Set Names in Source Audit Assistant 303

Appendixes 305

Appendix A. Problem Determination 307
Installation Abends 307
Error Messages 308
COMBINE: Creating the JCL (ARCOxxxx) 309
SUMMARY: Creating the JCL (ARSUxxxx) 310
SETUP: Creating the JCL (ASETxxxx) 311
COMBINE: Executing the JCL (CMBxxxx) 312
COMMANDS: Executing User Commands (CMD5xxxx) 314
COMMON: Common Messages for CA/DA/UTA User Interface (COMNxxxx) 319
DEFAULTS: Defaults Processing (DFLTxxxx) 326
FILE WARP & DISTILLATION: File Conversion (FCVxxxx) 328
FILE WARP: Control File Reader (FWPxxxx) 332
COMMON: Additional Common Messages (KPDSxxxx) 334
CTLFILE: Control File Processing (N2Oxxxx) 335
TARGETED SUMMARY: Targeted Summary (OCUSxxxx) 340
VARIABLE REPORTS: Executing the JCL (PVRxxxx) 343
DISTILLATION: Read the Keylist (RKEYxxxx) 346
REPORT: Executing the JCL (RPT03xx) 347
SAA: Source Audit Assistant (SAACxxxx) 350
SAA POSTPROCESSOR: Source Audit Assistant Change Validation Report (SAAFxxxx) 353
SAA EXECUTION: Source Audit Assistant Execution (SAARxxxx) 355
SETUP: Executing the JCL (SP601xx) 359
SUMMARY: Executing the JCL (SUM0xxxx) 362
VAREAD: Extracting Variable Information from Listing (VAR0xxx) 364
EXECUTE: Buffer Monitor (WVAxxxx) 373
ZAPTXT: Executing the JCL (ZAP00yy or ZAP88xx) 375
Understanding the SAA Log File 377
- Description of the Message Format 377
- Restrictions Regarding Embedded Statements 378
- Message List 379

Appendix B. ATC Requirements and Resources 383
Prerequisites and Supported Compilers 383
CA Resources 385
- Setup 385
- Monitor ECSA, SQA, and ESQA Usage 386
- Reports 386
- Data Set Attributes 387
- DDNAMEs 387

DA and UTA Resources 388
Setup 388
Monitor ECSA, SQA, and ESQA Usage 389
Data Set Attributes 389
DDNAMEs 389
SAA Resources 389

Appendix C. DBCS Support 391
DBCS Requirements for ATC Compilers and Assemblers 391
CA/DA/UTA DBCS Support 391
SAA DBCS Support 392
SAA Postprocessor DBCS Support 392

Glossary and Index 393

Glossary 395

Index 399

Figures

1. Site Defaults Data Set . . . 22
2. ATC Primary Option Menu . . . 27
3. Manipulate ATC Defaults Panel . . . 28
4. ATC Defaults Panel . . . 29
5. Reset Defaults to System Defaults Panel . . . 32
6. CA—Flow Diagram . . . 41
7. Sample Run—Flow Diagram . . . 46
8. Summary Reports for COB01M . . . 48
9. Control File for COB01M . . . 50
10. Annotated COBOL Listing . . . 55
11. Summary Report for PLI01M . . . 60
12. Control File for PLI01M . . . 62
13. Annotated PL/I Listing . . . 66
14. Summary Report for ASM01L . . . 70
15. Control File for ASM01L . . . 72
16. Annotated Assembler Listing . . . 76
17. Summary Coverage Report for COB01M in COBOL . . . 84
18. Summary Coverage Reports for PLI01M in PL/I . . . 85
19. Summary Report for ASM01L . . . 89
20. Coverage Reports Panel . . . 90
21. Create JCL for Summary Report Panel . . . 91
22. Create JCL for Summary and Annotated Listing Report Panel . . . 93
23. Annotated COBOL Listing . . . 96
24. Annotated PL/I Listing . . . 98
25. Annotated ASM Listing . . . 99
26. Annotated Listing with Execution Counts . . . 102
27. Sample Conditional Branch Coverage with UTA . . . 103
28. Sample Conditional Branch Coverage without UTA . . . 103
29. Sample Annotated Listing with UTA . . . 103
30. Sample Annotated Listing without UTA . . . 103
31. Target Control File . . . 111
32. Create JCL for Targeted Summary Report . . . 112
33. Using CA in a Large Project Environment—Flow Diagram . . . 116
34. Combining Results of Multiple Testers—Flow Diagram . . . 117
35. Create JCL for Combining Multiple Runs Panel . . . 118
36. ISPF Edit Screen for Combine . . . 119
37. DA—Flow Diagram . . . 129
38. Sample Run—Flow Diagram . . . 134
39. Input Master Data Set for COB11M and PLI11M . . . 136
40. New (Distilled) Master Data Set for COB11M and PLI11M . . . 136
41. Control File for COB11x . . . 138
42. Control File for PLI11x . . . 138
43. Partial Annotated Listing for COB011 Showing Key Numbers . . . 142
44. CACTL Statements for Distillation (COBOL) . . . 147
45. COBOL File Definition . . . 147
46. CACTL Statements for Distillation (PL/I) . . . 147
47. Generate JCL to Generate Key List and Distill Data Panel . . . 151
48. Generate JCL to Generate Key List and Distill DASD Data Panel . . . 152
49. Generate JCL to Generate Key List and Distill Tape Data Panel . . . 154
50. Edit Distillation JCL Panel . . . 156

© Copyright IBM Corp. 1997, 1999

51. Submit Distillation JCL Panel . . . 157
52. UTA—Flow Diagram . . . 164
53. Sample Run—Flow Diagram . . . 168
54. MVR for COB02M . . . 171
55. VDR for COB02M . . . 171
56. Control File for COB02x . . . 173
57. MVR for COB01M . . . 178
58. VDR for COB01M . . . 178
59. Control File for COB01M . . . 180
60. Sample UTA Control File . . . 188
61. COBOL File Definitions . . . 188
62. Control File for Warping a PL/I Input Buffer . . . 188
63. MVR for COB01M . . . 191
64. VDR for COB01M . . . 192
65. CVDR for COB01M . . . 193
66. Unit Test Report Panel . . . 194
67. UTA—Flow Diagram . . . 197
68. Copy Book Defining a File Record . . . 198
69. Example of a Warp Control File . . . 198
70. Generate JCL for File Warping Panel . . . 199
71. Example Control File with File Warping Definitions . . . 205
72. Coverage, Distillation and Unit Test Assistant Panel . . . 209
73. Work with the CA/DA/UTA Control File Panel . . . 209
74. CA Setup—Flow Diagram . . . 237
75. DA and UTA Setup—Flow Diagram . . . 237
76. Create JCL for Setup Panel . . . 239
77. CA Test Case Execution—Flow Diagram . . . 245
78. DA Test Case Execution—Flow Diagram . . . 246
79. UTA Test Case Execution—Flow Diagram . . . 246
80. Create JCL to Start the Monitor Panel . . . 247
81. Control the CA/DA/UTA Monitor Panel . . . 256
82. Monitor: Display Breakpoint Status Panel . . . 256
83. Breakpoint Status Panel . . . 257
84. Monitor: Take Snapshot of Data—Panel 1 . . . 259
85. Monitor: Take Snapshot of Data—Panel 2 . . . 259
86. Snapshot Status Panel . . . 260
87. Monitor: Add ID Panel . . . 261
88. Add ID Status Panel . . . 261
89. Monitor: Display Listings . . . 262
90. Listings Statistics Panel . . . 263
91. Monitor: Quit Monitor—Panel 1 . . . 265
92. Monitor: Quit Monitor—Panel 2 . . . 265
93. Quit Monitor Status Panel . . . 265
94. Monitor: Reset All Data in Monitor Panel . . . 266
95. Active Session Display Panel . . . 267
96. Release Level and Table Address Data . . . 267
97. Monitor: Display Statistics Panel . . . 268
98. PA Statistics Panel . . . 269
99. Monitor: Stop Monitor—Panel 1 . . . 270
100. Monitor: Stop Monitor—Panel 2 . . . 270
101. Stop Completed Panel . . . 271
102. Monitor: Display Variable Status Panel . . . 272
103. Variable Information Panel . . . 272
104. ATC Primary Option Menu . . . 282

105. Execute Source Audit Assistant Panel 283
106. SAA Background Execution Parameters Panel 283
107. SAA Foreground Execution Parameters Panel 285
108. Status Messages When All Filter Options Are Yes 286
109. SAA Foreground Execution Parameters Panel for Target Group Samples 287
110. Status Messages When All Filter Options Are No 287
111. SAA Postprocessor Panel 288
112. Execute Source Audit Assistant Panel 289
113. SAA Background Execution Parameters Panel 290
114. SAA Foreground Execution Parameters Panel 293
115. SAA Postprocessor Panel 296
116. Output Report Created by Comparing COBOL Source Files 299
117. Sample SAA Change Validation Report 302

Notices

This information was developed for products and services offered in the U.S.A. IBM may 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 described 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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785, U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling: (1) the exchange of information between independently created programs and other programs (including this one) and (2) the mutual use of the information which has been exchanged, should contact the IBM Corporation, Department J01, 555 Bailey Avenue, San Jose, CA 95161-9023. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

IBM
BookManager
CICS
DB2
DFSMS/MVS
Language Environment
Library Reader
MVS/ESA
OS/390

Other company, product, and service names may be trademarks or service marks of others.

© Copyright IBM Corp. 1997, 1999

Preface

The IBM® Application Testing Collection is a suite of tools that provide software to help you resolve potential problems related to the Year 2000.

Who Should Read This Book

Read this book if you are an application programmer who must ensure that programs comply with Year 2000 requirements.

How This Book Is Organized

This book is organized into the following parts:

- **"Introducing the IBM Application Testing Collection"** offers a general description of the Application Testing Collection suite of tools.
- **"Installing and Customizing ATC"** describes how to install ATC and customize functions for your location.
- **"Using Coverage Assistant"** describes information that is specific to the Coverage Assistant tool, including steps that explain how to edit the control file and run reports.
- **"Using Distillation Assistant"** describes information that is specific to the Distillation Assistant tool, including steps that explain how to edit the control file and run distillation.
- **"Using Unit Test Assistant"** describes information that is specific to the Unit Test Assistant tool, including steps that explain how to edit the control file and run reports.
- **"Common CA, DA, and UTA Information"** describes information that is common between Coverage Assistant, Distillation Assistant, and Unit Test Assistant. This information includes setting up, executing the monitor, and troubleshooting.
- **"Using Source Audit Assistant"** explains how to use the Source Audit Assistant tool, including the use of filters, which allow you to specify types of information that you want to see in your comparison report.
- **Appendix A, "Problem Determination"** explains any error messages you may encounter while using ATC.
- **Appendix B, "ATC Requirements and Resources"** contains ATC resource requirements.
- **Appendix C, "DBCS Support"** describes variances among the ATC tools for *DBCS (double-byte character set)* support.

© Copyright IBM Corp. 1997, 1999

Conventions for Words and Type

The following list shows special ways in which some characters and words are displayed in this manual and describes the meaning associated with each one:

Display Method	Meaning
`Monospaced type`	Shows something that you type (such as a command), an example, or something that is displayed on your monitor (for example, an error message, or the name of a panel or field).
Italic type	Indicates information that you supply (such as a parameter or a variable), or italic type can indicate a new term. See the glossary for definitions of new terms.
Bold type	Indicates information that you should pay particular attention to.

How to Read Syntax Diagrams

Throughout this book, syntax is presented in diagrams. The following list describes how to read the diagrams to enter commands correctly.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ►►— symbol indicates the beginning of a statement.

  The —► symbol indicates that the statement syntax is continued on the next line.

  The ►— symbol indicates that a statement is continued from the previous line.

  The —►◄ symbol indicates the end of a statement.

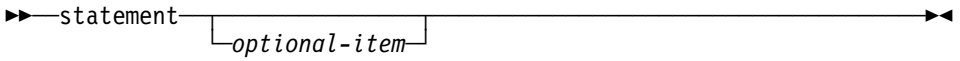
  Diagrams of syntactical units other than complete statements start with the ►— symbol and end with the —► symbol.

  Fragments of a syntax diagram are enclosed by the symbol ├─┤.

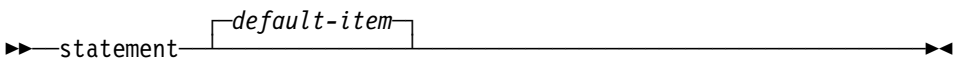
- Required items appear on the horizontal line (the main path).

  ►►—`statement`—*`required-item`*—►◄

- Optional items appear below the main path.

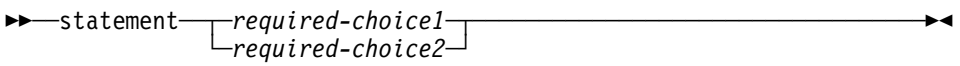


- Defaults appear above the main path.

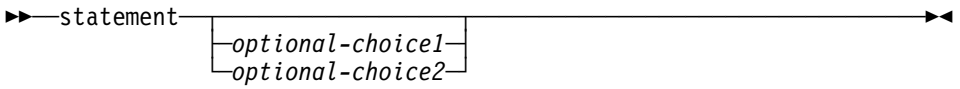


- If you can choose from two or more items, they appear vertically, in a stack.

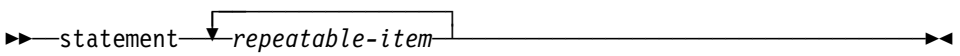
  If you *must* choose one of the items, one item of the stack appears on the main path.



  If choosing one of the items is optional, the entire stack appears below the main path.

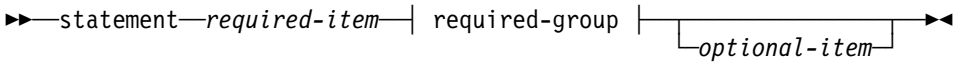


- An arrow returning to the left above the main path indicates an item that can be repeated.

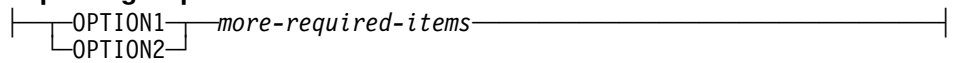


  A repeat arrow above a stack indicates that you can make more than one choice from the stacked items, or repeat a single choice.

- An out-of-line fragment of a syntax diagram, which is shown later in the syntax diagram (or in another syntax diagram), is enclosed in vertical bars.



  required-group:



- Keywords appear as bold uppercase letters and should be spelled exactly as shown (for example, **DEFAULTS**). However, keywords may be coded in any combination of uppercase and lowercase characters.

  Variables appear in all italic letters (for example, *label*). They represent user-supplied names or values.

- Only one keyword from a particular group can be specified, and the same keyword cannot be specified more than once unless explicitly indicated.
- If punctuation marks, parentheses, arithmetic operators, or any such symbols are shown, you must enter them as part of the syntax.

Related Information

The following publications are available in downloadable form from the IBM Year 2000 Technical Support Center Web site:

- *IBM Application Testing Collection for MVS/ESA Version 1 Release 5 Modification 0 General Information,* Program Number 5799-GBN, PRPQ P85579
- *IBM Automated Regression Testing Tool Version 2 Release 2 Modification 0 User's Guide,* Program Number 5799-GBN, PRPQ P85579

To locate these documents on the Internet:

1. Go to http://www.software.ibm.com/year2000/
2. Select the **Testing** link on the main home page, and then look for the Application Testing Collection on the page displayed.

IBM has developed the following Redbook that outlines the Y2K test process and augments it with real-world application examples and tool usage information:

VisualAge 2000 Test Solution: Testing Your Year 2000 Conversion, SG24-2230

To order this Redbook from the Internet:

1. Go to http://www.redbooks.ibm.com/index.html
2. Select the **Redbooks Online!** button at the bottom of the page.
3. In the search field displayed, enter the Redbook's title or document number.
4. Look for ordering instructions on the page displayed.

Getting Help

To open a problem with ATC, please enter an electronic PMR (problem management record) in RETAIN (remote technical assistance information network [IBM]) to the queue and center ATC,136 using the Compid 5799GBN00.

If you **do not** have access to RETAIN, in the United States you can call 1-800-237-5511 to have support personnel open a PMR for you in RETAIN.

Summary of Changes (V1R5M0)

Changes have been made to the following components of the Application Testing Collection (ATC) for Version 1 Release 5 Modification 0:

Coverage Assistant (CA)

The performance of CA targeted summary report has been improved and a batch interface has been added.

Coverage Assistant (CA), Distillation Assistant (DA), and Unit Test Assistant (UTA)

The SETUP process has been updated to allow load modules to be instrumented directly as an alternative to instrumenting object modules, which are then linked into load modules.

Summary of Changes (V1R4M0)

Changes have been made to the following components of the Application Testing Collection (ATC) for Version 1 Release 4 Modification 0:

Coverage Assistant (CA)

CA now supports IBM High Level Assembler (HLASM) Release 1 Version 3.

CA targeted coverage sample JCL for batch invocation has been added to the samples.

Unit Test Assistant (UTA)

File warping is the modification of variables (typically date variables) in program input files to simulate input conditions for testing. For example, file warping could be used to modify dates in input files to post-Year 2000 values for Year 2000 testing of remediated programs.

The new UTA file warp feature can copy any QSAM or VSAM file and warp fields in the copied file for testing. Any zoned or packed numeric field can be incremented, decremented, or set. Any zoned, packed, or character field can be set to a common value. For example, file warping could be used to clear fields in test copies of production input files for privacy or security reasons.

UTA logging and warping of COBOL variables now takes place at the beginning of a statement rather than at the end.

Source Audit Assistant (SAA)

SAA now supports IBM High Level Assembler (HLASM) Release 1 Version 3.

SAA postprocessor sample JCL for batch invocation has been added to the samples.

© Copyright IBM Corp. 1997, 1999

Automated Regression Testing Tool (ARTT)

ARTT, an automated code capture and verification tool, has been added to the collection of tools.

Summary of Changes (V1R3M4)

Changes have been made to the following components of the Application Testing Collection (ATC) for Version 1 Release 3 Modification 4:

Coverage Assistant (CA), Unit Test Assistant (UTA) and Distillation Assistant (DA)

- Support has been added for DBCS characters in the input compiler and assembler listings, and in identifiers and comments in the control files.
- Support has been added for the COBOL LANGUAGE(JAPANESE) compiler option.
- The start monitor job now detects when another session has already been started with a matching BRKTAB/listing/obj combination (error message CMD5023W).

Source Audit Assistant (SAA)

- Support for DBCS characters in the input source/listings has been added for the Comments and Declares filters for COBOL and assembler.
- Restrictions on the length of input data set names have been removed.
- Support for DBCS characters has been added to the SAA postprocessor.
- The seed list data set is no longer required in the SAA postprocessor; however, if it is not specified, a change validation report is not created.

Summary of Changes (V1R3M0)

Changes have been made to the following components of the Application Testing Collection for release V1R3M0:

Coverage Assistant (CA)

The requirement for the LANGUAGE(UE) option has been removed for the High Level Assembler.

Unit Test Assistant (UTA)

ATC now includes the UTA tool, which allows you to capture and log the values assigned to selected variables in your application programs at selected points during their execution. This is called *unit testing*. Unit testing allows you to confirm the effectiveness of Year 2000 changes that have been made to an application program.

In addition, Unit Test Assistant offers the ability to perform data *warping*. This means that variables can be modified automatically as they are encountered during program execution. UTA will intercept data entering or leaving a program at I/O time (or at other times where application logic dictates) and change the value of

that data in a manner that you specify. This feature is especially useful when doing future date testing of Year 2000 remediated code.

ATC User Documentation

The following changes pertain to the ATC user documentation. Information that has changed since the last publication of the *User's Guide* and the *General Information* document is marked with revision bars in the left margins of affected pages.

- New information about DBCS support has been added to the *User's Guide* and the *General Information* document.
- Changes to the *User's Guide* and the *General Information* document have been made to reflect functional changes to the product. Also, minor technical and editorial changes have been made throughout both documents.
- A BookManager® READ bookshelf, containing the *User's Guide* and the *General Information* document, has been added to the group of documentation offerings shipped with the ATC package.

Summary of Changes (V1R2M4)

The following changes have been made to the Application Testing Collection for release V1R2M4. This *User's Guide* reflects these changes.

Coverage Assistant (CA) and Distillation Assistant (DA)

The monitor and breakpoints are used by Coverage Assistant (CA) and Distillation Assistant (DA).

The breakpoint method now uses a user SVC rather than an invalid opcode. The monitor now consists of an SVC rather than a First Level Interrupt Handler. This allows the monitor to be run on systems that may be running production applications and removes the need to start the monitor before any other monitor at IPL time.

General

The following changes affect two or more ATC tools:

- Support for new levels of compilers and the *Millennium Language Extensions (MLE)* for the COBOL and PL/I compilers has been added.
- The installation procedures, system defaults, and personal defaults have been changed to handle the new user SVCs.
- Program abends because of breakpoint mismatch or monitor not installed have been updated to reflect the new SVC breakpoints. Appendix A, “Problem Determination” on page 307 contains these updated abends.
- The formulas for calculating monitor storage usage and binary file disk usage have changed. Appendix B, “ATC Requirements and Resources” on page 383 contains the new formulas.
- The SETUP (S*) and start monitor (X*) JCL are significantly different for V1R2M4 (as compared to V1R2M0). You must recreate any manually generated or custom JCL (using the samples as templates).
- The start monitor JCL generation panel now allows you to edit the Session ID option, but it continues to default to your TSO user ID.

- The DCB characteristics of the VARCTL file has been changed to LRECL=64, any valid block size.
- The format of all of the binary files (BRKOUT, BRKTAB, DBGTAB, VARTAB, and VARCTL) has changed. You must regenerate all binary files when moving to this release.
- Since the breakpointing technique has changed, any existing breakpointed object or load modules must be regenerated.
- Many of the parameters of the CA commands have changed, and some new CA commands have been created. The new CALIST and existing CASTATS and CABPDSP commands provide a hierarchy of commands that allow you to look at a running session at different levels of detail.

Introducing the IBM Application Testing Collection

© Copyright IBM Corp. 1997, 1999

Overview

The Application Testing Collection (ATC) is a set of tools that enable you to examine application programs for compliance with Year 2000 requirements. The tools that make up ATC are as follows:

Coverage Assistant

Coverage Assistant (CA) measures code coverage in application programs written in the COBOL, PL/I, and S/390 assembler languages and compiled by specific IBM* COBOL and PL/I compilers or assembled by the High Level Assembler or Assembler H.

Distillation Assistant

Distillation Assistant (DA) monitors file reads for a specified file that contains logical keys and determines all of the keys that caused increased code coverage. It then creates a "distilled" copy of the input file, containing only the records containing these keys. This smaller, distilled file can then be used during program testing to obtain equivalent code coverage, but at the same time, decrease the time and resources required for the testing. DA does this for applications written in the COBOL or PL/I language and compiled by specific IBM COBOL or PL/I compilers.

Unit Test Assistant

Unit Test Assistant (UTA) reads and records the values assigned to variables at selected statement numbers in an application program as it is executing. It creates a report, which allows subsequent analysis of the variable data. UTA does this for applications written in the COBOL language and compiled by specific IBM COBOL compilers.

In addition, UTA provides two types of data warping:

1. Dynamic data warping
2. File warping

UTA *dynamic data warping* automatically modifies variables as they are encountered during program execution. When data enters or exits a program at I/O time (or at other times where the application logic dictates), UTA changes the value of that data in a manner that you specify. UTA does this for COBOL variables and PL/I input buffers.

A standard *file warping* process is that of aging, or warping, occurrences of dates in input data files. Using UTA file warping, copies of input files are made with data fields of records warped under user control.

Source Audit Assistant

Source Audit Assistant (SAA) compares two levels of source code and places the results in a comparison report. SAA helps locate differences, verify whether changes are valid, and identify items that need closer examination.

Automated Regression Testing Tool

Automated Regression Testing Tool (ARTT) is a code capture and verification tool that records a baseline execution of your program and automatically compares it with a proof execution using real or previously captured input. ARTT permits all levels of testing (unit, function, integration, and system) with or without a production system, and its data conversion capability allows testing to continue when I/O and programs

© Copyright IBM Corp. 1997, 1999

are in incompatible formats (for example, when one has been modified to accept future dates and the other has not). Data conversion can be particularly useful if you are unsure of the status of I/O received from outside sources. For more information, see the user documentation that was shipped with ARTT.

Installing and Customizing ATC

© Copyright IBM Corp. 1997, 1999

Prerequisites and Supported Software

ATC was developed and tested in the following environments. These environments and newer releases support ATC.

- MVS/ESA™ 5.1.0, DFSMS/MVS® 1.2, TSO/E 2.5 and ISPF 4.2.1
- MVS/ESA 5.2.2, DFSMS/MVS 1.3, TSO/E 2.5 and ISPF 4.2.1
- OS/390® 2.4.0, DFSMS/MVS 1.4, TSO/E 2.6, and ISPF for OS/390 1.3
- Language Environment® for MVS & VM 1.5

  Note: The ATC programs use Language Environment for MVS & VM 1.5, however it is not a requirement for the user's programs.

All IBM supported releases of OS/390 and the OS/390 versions of DFSMS/MVS, TSO/E, ISPF, and Language Environment support ATC.

Coverage Assistant supports the following compilers and assemblers:[1]

- IBM COBOL for OS/390 & VM 2.1 (plus *Millennium Language Extensions [MLE]*)
- IBM COBOL for MVS & VM 1.2 (plus Millennium Language Extensions [MLE])
- IBM VS COBOL II Release 4.0
- IBM OS/VS COBOL Release 2.4
- IBM PL/I for MVS & VM 1.1.1 (plus Millennium Language Extensions [MLE])
- IBM OS PL/I Optimizing Compiler 2.3.0
- IBM PL/I Optimizing Compiler 1.5.1
- IBM High Level Assembler Version 1 Release 2 and Release 3
- IBM Assembler H Version 2

Distillation Assistant supports the following compilers:[1]

- IBM COBOL for OS/390 & VM 2.1 (plus Millennium Language Extensions [MLE])
- IBM COBOL for MVS & VM 1.2 (plus Millennium Language Extensions [MLE])
- IBM VS COBOL II Release 4.0
- IBM OS/VS COBOL Release 2.4
- IBM PL/I for MVS & VM 1.1.1 (plus Millennium Language Extensions [MLE])
- IBM OS PL/I Optimizing Compiler 2.3.0
- IBM PL/I Optimizing Compiler 1.5.1

Unit Test Assistant supports the following compilers:[1]

- IBM COBOL for OS/390 & VM 2.1 (plus Millennium Language Extensions [MLE])
- IBM COBOL for MVS & VM 1.2 (plus Millennium Language Extensions [MLE])
- IBM VS COBOL II Release 4.0
- IBM OS/VS COBOL Release 2.4

[1] For a list of required compiler options and restrictions, see "Compiler Options" on page 231.

© Copyright IBM Corp. 1997, 1999

- The following PL/I compilers are supported for data warping of file input buffers **only**:
  - IBM PL/I for MVS & VM 1.1.1 (plus Millennium Language Extensions [MLE])
  - IBM OS PL/I Optimizing Compiler 2.3.0
  - IBM PL/I Optimizing Compiler 1.5.1

Source Audit Assistant supports the following languages in source code form (the reports option only supports seed analysis and template target control file generation for COBOL and PL/I):[2]

- Assembler
- C
- C++
- COBOL
- PL/I

Source Audit Assistant supports the following languages in listing form (the reports option only supports seed analysis and template target control file generation for COBOL and PL/I):

- IBM COBOL for OS/390 & VM 2.1 (plus Millennium Language Extensions [MLE])
- IBM COBOL for MVS & VM 1.2 (plus Millennium Language Extensions [MLE])
- IBM VS COBOL II Release 4.0
- IBM OS/VS COBOL Release 2.4
- IBM PL/I for MVS & VM 1.1.1 (plus Millennium Language Extensions [MLE])
- IBM OS PL/I Optimizing Compiler 2.3.0
- IBM PL/I Optimizing Compiler 1.5.1
- IBM High Level Assembler Version 1 Releases 1, 2, and 3
- IBM Assembler H Version 2

[2] For restrictions, see “What Does SAA Require?” on page 279.

Converting Existing ATC Systems and Users to the Current Release

You may have to make modifications when converting your existing ATC systems and users to the current ATC release. The following information provides detailed conversion instructions based upon the release you are presently using.

V1R4M0 Systems and Users to V1R5M0

When converting from ATC release V1R4M0 to release V1R5M0, note the following:

- **Running more than one release at the same time.** V1R5M0 is intended to be installed in a separate code base so that you can have both releases of the tool available at the same time.
- **V1R5M0 and V1R4M0 monitors.** The V1R5M0 monitor replaces the V1R4M0 monitor. If both releases of code must be run on the same machine, the V1R5M0 monitor should be used for both releases.
- **Defaults changes.** The changes to defaults are minor. However, it is still recommended that each user reset the defaults (ATC option 0.2) when switching from one release to another. (Users will have to reenter any personal changes.)
- **Binary data sets.** The binary data sets (BRKOUT, BRKTAB, DBGTAB, VARTAB, and VARCTL) are upwards compatible between V1R4M0 and V1R5M0.
- **JCL.** Any existing JCL (or JCL generators) should be updated to point to the V1R5M0 data sets. Any existing CA Targeted Summary report JCL should be regenerated with the new JCL generator or recreated using the V1R5M0 sample as a template.

V1R3M4 Systems and Users to V1R5M0

When converting from ATC release V1R3M4 to release V1R5M0, note the following:

- **Running more than one release at the same time.** V1R5M0 is intended to be installed in a separate code base so that you can have both releases of the tool available at the same time.
- **V1R5M0 and V1R3M4 monitors.** The V1R5M0 monitor replaces the V1R3M4 monitor. If both releases of code must be run on the same machine, the V1R5M0 monitor should be used for both releases.
- **Defaults changes.** The changes to defaults are minor. However, it is still recommended that each user reset the defaults (ATC option 0.2) when switching from one release to another. (Users will have to reenter any personal changes.)
- **Binary data sets.** The binary data sets (BRKOUT, BRKTAB, DBGTAB, VARTAB, and VARCTL) are upwards compatible between V1R3M4 and V1R5M0.

© Copyright IBM Corp. 1997, 1999

- **JCL.** Any existing JCL (or JCL generators) should be updated to point to the V1R5M0 data sets. The *hi_lev_qual*.*.SETCTL data set (ddname SETCTL) is no longer needed for the SETUP (S*) JCL. Any existing CA Targeted Summary report JCL should be regenerated with the new JCL generator or recreated using the V1R5M0 sample as a template.
- **CACTL.** UTA logging and warping of COBOL variables now takes place at the beginning of a statement rather than at the end. Any existing control files containing a *coverage* statement must be modified accordingly.

V1R3M0 Systems and Users to V1R5M0

When converting from ATC release V1R3M0 to release V1R5M0, note the following:

- **Running more than one release at the same time.** V1R5M0 is intended to be installed in a separate code base so that you can have both releases of the tool available at the same time.
- **V1R5M0 and V1R3M0 monitors.** The V1R5M0 monitor replaces the V1R3M0 monitor. If both releases of code must be run on the same machine, the V1R5M0 monitor should be used for both releases.
- **Defaults changes.** The changes to defaults are minor. However, it is still recommended that each user reset the defaults (ATC option 0.2) when switching from one release to another. (Users will have to reenter any personal changes.)
- **Binary data sets.** The binary data sets (BRKOUT, BRKTAB, DBGTAB, VARTAB, and VARCTL) are upwards compatible between V1R3M0 and V1R5M0.
- **JCL.** Any existing JCL (or JCL generators) should be updated to point to the V1R5M0 data sets. Any existing batch SAA JCL must be regenerated. The *hi_lev_qual*.*.SETCTL data set (ddname SETCTL) is no longer needed for the SETUP (S*) JCL. Any existing CA Targeted Summary report JCL should be regenerated with the new JCL generator or recreated using the V1R5M0 sample as a template.
- **CACTL.** UTA logging and warping of COBOL variables now takes place at the beginning of a statement rather than at the end. Any existing control files containing a *coverage* statement must be modified accordingly.

V1R2M4 Systems and Users to V1R5M0

When converting from ATC release V1R2M4 to release V1R5M0, note the following:

- **Running more than one release at the same time.** V1R5M0 is intended to be installed in a separate code base so that you can have both releases of the tool available at the same time.
- **V1R5M0 and V1R2M4 monitors.** The V1R5M0 monitor replaces the V1R2M4 monitor. If both releases of code must be run on the same machine, the V1R5M0 monitor should be used for both releases.
- **Defaults changes.** The changes to defaults are minor. However, it is still recommended that each user do a RESET DEFAULTS (ATC option 0.2) when

switching from one release to another. (Users will have to reenter any personal changes.)

- **Binary data sets.** The binary data sets (BRKOUT, BRKTAB, DBGTAB, VARTAB, and VARCTL) are upwards compatible between V1R2M4 and V1R5M0.
- **JCL.** Any existing JCL (or JCL generators) should be updated to point to the V1R5M0 data sets. Any existing batch SAA JCL must be regenerated. The *hi_lev_qual*.*.SETCTL data set (ddname SETCTL) is no longer needed for the SETUP (S*) JCL. Any existing CA Targeted Summary report JCL should be regenerated with the new JCL generator or recreated using the V1R5M0 sample as a template.

V1R2M0 Systems and Users to V1R5M0

When converting from ATC release V1R2M0 to release V1R5M0, note the following:

- **Running more than one release at the same time.** The V1R5M0 level of code introduces a new monitor technology that uses *user SVCs (supervisor calls*) as breakpoints rather than invalid opcodes. This new monitor can be run on the same machine as the older monitor (V1R2M0) if you install the V1R2M1 (or newer) maintenance level on your V1R2M0 code base.

  V1R5M0 is intended to be installed in a separate code base so that you can have both releases of the tool available at the same time. Follow on releases will only use the new monitor technology, so you are encouraged to move all of your test scenarios to the V1R5M0 code base.
- **Defaults changes.** It is recommended that each user do a RESET DEFAULTS (ATC option 0.2) when switching from one release to another. (Users will have to reenter any personal changes.)
- **Data set changes.** The DCB characteristics of the VARCTL data set have been changed to LRECL=64, any valid block size. If you use sequential data sets for the VARCTL file, you do not have reallocate existing files since they are normally deleted and recreated by the JCL generated by the tool. If you use partitioned data sets, you must reallocate these data sets to use the new LRECL.

  The internal format of all of the binary data sets (BRKOUT, BRKTAB, DBGTAB, VARTAB, and VARCTL) has changed in V1R5M0. You must regenerate all binary files when moving to this release (no binary files generated by V1R2M0 can be used as input to the V1R5M0 monitor and other tools). In addition, you must regenerate any existing breakpointed object and must relink any existing load modules containing breakpointed code.

  Existing control files (CACTL and TARGCTL), compiler listings, and object modules may be reused.
- **JCL changes.** The SETUP (S*), start monitor (X*), and batch SAA JCL are significantly different for V1R5M0 (as compared to V1R2M0); therefore, you must:
  - Regenerate any S* and X* JCLs that you want to rerun
  - Regenerate any existing batch SAA JCL that you want to rerun
  - Update any custom JCL generators (using the samples as templates)

  - Any other existing JCL or JCL generators should be updated to point to the V1R5M0 data sets.

  Any existing CA targeted summary report JCL should be regenerated with the new JCL generator or recreated using the V1R5M0 sample as a template.

- **CA command (exec) changes.** Many of the parameters of the CA commands have changed, and some new CA commands have been created. The new CALIST and existing CASTATS and CABPDSP commands provide a hierarchy of commands that allow you to look at a running session at different levels of detail. If you have existing CLIST, REXX, or JCL procedures that invoke these commands, you should review each invocation to ensure that it meets the current syntax definition.
- **Monitor common storage usage.** The storage usage formula for the monitor has changed. For details, see "Monitor ECSA, SQA, and ESQA Usage" on page 386.
- **CACTL.** UTA logging and warping of COBOL variables now takes place at the beginning of a statement rather than at the end. Any existing control files containing a *coverage* statement must be modified accordingly.

System Installation

This chapter provides information about installing the Application Testing Collection. To use ATC, you must perform a system installation for each MVS system.

System installation consists of the following activities:

1. Installing the data sets
2. Modifying the FORMS, REXX, and BKSHELF data sets
3. Setting up the authorized data sets
4. Installing and enabling the monitor *SVCs (supervisor calls)*
5. Ensuring users have access to AMASPZAP
6. Editing the site defaults data set
7. Verifying the installation.

Each activity is described more fully in topics that follow in this chapter. For information about installing Automated Regression Testing Tool (ARTT), see the user documentation that was shipped with ARTT.

Installing Data Sets

To use this tool on MVS, install the following data sets from the installation media. During installation, *hi_lev_qual* will be replaced by the MVS data set high-level qualifier that you specify.

- Install Notes and JCL:

hi_lev_qual.V1R5M0.README	Installation notes and JCL

- Execution Libraries:

hi_lev_qual.V1R5M0.LOADLIB	Executable modules
hi_lev_qual.V1R5M0.REXX	Command procedures (REXX execs)
hi_lev_qual.V1R5M0.PANELS	ISPF panels
hi_lev_qual.V1R5M0.SKELS	ISPF JCL skeletons
hi_lev_qual.V1R5M0.MSGS	ISPF messages
hi_lev_qual.V1R5M0.TABLES	ISPF tables

- Defaults Data Sets:

hi_lev_qual.V1R5M0.MASTER.DEFAULTS	Site defaults
hi_lev_qual.V1R5M0.FORMS	Control file templates

- General Information Guide:

hi_lev_qual.V1R5M0.ATCGI.LIST3820	ATC General Information Guide
hi_lev_qual.V1R5M0.ATCGI.PSBIN	ATC General Information Guide (for a PostScript** printer)
hi_lev_qual.V1R5M0.ATCGI.BOOK	ATC General Information Guide (for online viewing)
hi_lev_qual.V1R5M0.ATCGI.PDFBIN	ATC General Information Guide (for online viewing with Adobe** Acrobat** Reader)

© Copyright IBM Corp. 1997, 1999

- User's Guide:

hi_lev_qual.V1R5M0.ATCUG.LIST3820	ATC User's Guide
hi_lev_qual.V1R5M0.ATCUG.PSBIN	ATC User's Guide (for a PostScript** printer)
hi_lev_qual.V1R5M0.ATCUG.BOOK	ATC User's Guide (for online viewing)
hi_lev_qual.V1R5M0.ATCUG.PDFBIN	ATC User's Guide (for online viewing with Adobe** Acrobat** Reader)

- Bookshelf:

hi_lev_qual.V1R5M0.BKSHELF	ATC bookshelf (for online viewing)
hi_lev_qual.V1R5M0.BKINDEX	ATC bookshelf index (for online viewing)

- CA/DA/UTA Sample Libraries:

hi_lev_qual.V1R5M0.SAMPLE.COBOL	Sample COBOL test cases
hi_lev_qual.V1R5M0.SAMPLE.COBOLST	Sample COBOL listings
hi_lev_qual.V1R5M0.SAMPLE.COSVSLST	Sample OS/VS COBOL listings
hi_lev_qual.V1R5M0.SAMPLE.PLI	Sample PL/I test cases
hi_lev_qual.V1R5M0.SAMPLE.PLILST	Sample PL/I listings
hi_lev_qual.V1R5M0.SAMPLE.ASM	Sample ASM test cases
hi_lev_qual.V1R5M0.SAMPLE.ASMLLST	Sample High Level Assembler listings
hi_lev_qual.V1R5M0.SAMPLE.ASMHLST	Sample Assembler H listings
hi_lev_qual.V1R5M0.SAMPLE.JCL	Sample JCL
hi_lev_qual.V1R5M0.SAMPLE.OBJ	Sample object code
hi_lev_qual.V1R5M0.SAMPLE.ZAPOBJ	Modified object code
hi_lev_qual.V1R5M0.SAMPLE.RUNLIB	Sample executable programs

- DA Sample Master Data Set:

  hi_lev_qual.V1R5M0.SAMPLE.COB11.MASTER

- UTA Sample File Warp Input Data Set:

  hi_lev_qual.V1R5M0.SAMPLE.FWARPIN

- CA/DA/UTA Sample Control Data Sets:

hi_lev_qual.V1R5M0.SAMPLE.CACTL	CA/DA/UTA control statements
hi_lev_qual.V1R5M0.SAMPLE.TARGCTL	CA target control statements
hi_lev_qual.V1R5M0.SAMPLE.CBCTL	Combine control statements
hi_lev_qual.V1R5M0.SAMPLE.FWCTL	UTA file warp control statements

- CA/DA/UTA Sample Output:

  hi_lev_qual.V1R5M0.SAMPLE.BRKOUT
  hi_lev_qual.V1R5M0.SAMPLE.BRKTAB
  hi_lev_qual.V1R5M0.SAMPLE.DBGTAB
  hi_lev_qual.V1R5M0.SAMPLE.REPORT
  hi_lev_qual.V1R5M0.SAMPLE.SUMMARY
  hi_lev_qual.V1R5M0.SAMPLE.TARGREP
  hi_lev_qual.V1R5M0.SAMPLE.VARCTL
  hi_lev_qual.V1R5M0.SAMPLE.VARDATA
  hi_lev_qual.V1R5M0.SAMPLE.VARID
  hi_lev_qual.V1R5M0.SAMPLE.VARTAB
  hi_lev_qual.V1R5M0.SAMPLE.COB11M.DISTILL

hi_lev_qual.V1R5M0.SAMPLE.COB112.DISTILL
hi_lev_qual.V1R5M0.SAMPLE.COB11O.DISTILL
hi_lev_qual.V1R5M0.SAMPLE.PLI11M.DISTILL
hi_lev_qual.V1R5M0.SAMPLE.PLI112.DISTILL
hi_lev_qual.V1R5M0.SAMPLE.PLI111.DISTILL
hi_lev_qual.V1R5M0.SAMPLE.FWARPOUT

- SAA Sample Data Sets:

  hi_lev_qual.V1R5M0.SAMPLE.SAA.NEW.COPY
  hi_lev_qual.V1R5M0.SAMPLE.SAA.OLD.COPY
  hi_lev_qual.V1R5M0.SAMPLE.SAA.COPY.B.NNN.CMP
  hi_lev_qual.V1R5M0.SAMPLE.SAA.COPY.B.NNY.CMP
  hi_lev_qual.V1R5M0.SAMPLE.SAA.COPY.B.NYY.CMP
  hi_lev_qual.V1R5M0.SAMPLE.SAA.COPY.B.YYY.CMP
  hi_lev_qual.V1R5M0.SAMPLE.SAA.COPY.NNN.CMP
  hi_lev_qual.V1R5M0.SAMPLE.SAA.COPY.NNY.CMP
  hi_lev_qual.V1R5M0.SAMPLE.SAA.COPY.NYY.CMP
  hi_lev_qual.V1R5M0.SAMPLE.SAA.COPY.YYY.CMP
  hi_lev_qual.V1R5M0.SAMPLE.SAA.NEW.COBOL
  hi_lev_qual.V1R5M0.SAMPLE.SAA.OLD.COBOL
  hi_lev_qual.V1R5M0.SAMPLE.SAA.SEEDLIST
  hi_lev_qual.V1R5M0.SAMPLE.SAA.COBOLM.CMP
  hi_lev_qual.V1R5M0.SAMPLE.SAA.COBOLM.REP
  hi_lev_qual.V1R5M0.SAMPLE.SAA.COBOLM.TARGCTL
  hi_lev_qual.V1R5M0.SAMPLE.SAA.NEW.COBLST
  hi_lev_qual.V1R5M0.SAMPLE.SAA.OLD.COBLST
  hi_lev_qual.V1R5M0.SAMPLE.SAA.COBLST.CMP

Modifying the FORMS, REXX, and BKSHELF Data Sets

Once the data sets are installed on your system, you must modify the following data sets to specify your data set names.

- Edit the CBCTL, EXMCTLA, EXMCTLB, and EXMCTLP members of the FORMS data set. Change all entries that start with ATC to the *high-level qualifier* used for installation at your site, which is referred to as *hi_lev_qual* in the rest of this manual.
- Edit the ATSTART member of the REXX data set, following the directions in the member's prolog for site customization.
- Edit the BKSHELF data set. Change all entries that start with *ATC* to *hi_lev_qual*. If you want this bookshelf to be listed when BookManager READ is started on your system, refer to the “List of Bookshelves Available to All Users” topic in *BookManager READ/MVS Installation, Planning, and Customization,* SC38-2035.

Setting up the Authorized Data Sets

The following authorized data set or sets are needed for ATC:

1. Monitor interface and variable monitor

   a. Copy the following modules from the LOADLIB into an authorized data set that your users are permitted to access via STEPLIB/PGM= in a batch job and via TSO CALL in a REXX EXEC:

      - CMDUSVC (monitor interface)
      - VARMON3 (variable monitor)

   b. Add the CMDUSVC program to the AUTHPGM entry in the SYS1.PARMLIB member (IKJTSO*xx*) so that it can be called as an authorized TSO program; otherwise, an 047 abend will occur. Have your systems programmer tell TSO to use this new authority (dynamically, by issuing the PARMLIB UPDATE(*xx*) command from TSO, or by doing an IPL). You must be able to invoke this program using the TSO CALL command.

      You may need to contact your site information systems personnel to complete this change.

2. Monitor installer/enabler and monitor SVCs

   The MONINSTS and MONSVC load modules must be placed in an authorized library from which only a system programmer (or whoever is allowed to install SVCs) can execute from. This can be done in either of the following ways:

   - Use RACF to control who can execute these programs.

     a. Move the following modules from the LOADLIB into the same authorized data set that you used for CMSUSVC and VARMON3.

        – MONINSTS (monitor installer/enabler)
        – MONSVC (monitor SVCs)

     b. Set up RACF PROGRAM profiles to restrict who can execute these programs. Here is an example:

```
RDEFINE PROGRAM(MONINSTS) NOTIFY(notify) UACC(NONE) +
  DATA('RACF profile for ATC monitor') +
  ADDMEM('authlib'/'volser'/PADCHK) OWNER(owner))
RDEFINE PROGRAM(MONSVC) NOTIFY(notify) UACC(NONE) +
  DATA('RACF profile for ATC monitor') +
  ADDMEM('authlib'/'volser'/PADCHK) OWNER(owner)

SETROPTS WHEN(PROGRAM) REFRESH

PERMIT MONINSTS CLASS(PROGRAM) ID(id) ACCESS(READ)
PERMIT MONSVC   CLASS(PROGRAM) ID(id) ACCESS(READ)

SETROPTS WHEN(PROGRAM) REFRESH
```

     where:

     notify TSO user Id of the person who should be notified of a RACF access failure

     authlib Data set name of authorized library containing MONINSTS and MONSVC

volser Volume serial of authlib data set (or ****** to specify the current SYSRES volume)

owner TSO user ID or RACF group name that will own this profile

id TSO user ID or RACF group name of the person or persons who should have the ability to install the SVCs

c. Allow the MONINSTS-started task (described in “Installing and Enabling the Monitor SVCs” on page 18) to execute these modules.

d. Ensure that your users do not have read/execute access to these modules in an authorized library.

- Use RACF to control who has access to the authorized library that contains these load modules.

  a. Move the following modules from the LOADLIB into an authorized data set that ONLY a systems programmer (or whoever is allowed to install SVCs) has read/write access.

     – MONINSTS (monitor installer/enabler)
     – MONSVC (monitor SVCs)

  b. Allow the MONINSTS-started task (described in “Installing and Enabling the Monitor SVCs” on page 18) to execute programs out of this data set.

  c. Ensure that your users do not have read/execute access to these modules in an authorized library.

  You may need to contact your site information systems personnel to complete these changes.

Installing and Enabling the Monitor SVCs

Before a user starts a monitor session, the monitor SVCs must be installed and enabled. The MONINSTS module does this for you. It must be run to install/enable the SVCs, reinstall/enable the SVCs, or at any IPL time after the initial installation (to reinstall and re-enable the monitor SVCs).

To do the initial installation:

1. Acquire two free *user* SVC numbers from your systems programmer.

   Note: SYS1.PARMLIB(IEASVC*xx*) does not need to be updated since these user SVCs will be installed dynamically. However, you do need to ensure that these SVC numbers **are not** being used on your system.

2. Edit hi_lev_qual.V1R5M0.SAMPLE.JCL(INSTATC) and change the following:

   a. Change the STEPLIB data set name to the authorized data set that contains the MONINSTS and MONSVC modules.

   b. Change the JOB card to run on your system.

   c. Change the PARM operands to contain the two SVC numbers you acquired for ATC. **Make sure you enter these numbers (in hexadecimal notation) correctly!**

3. Submit this JCL on the system on which you intend to run the monitor (this JCL must be run by a system programmer who has access to these modules). The job should get a return code of 0 (RC=0). For a list of possible abends while running MONINSTS, see “Installation Abends” on page 307.

4. To verify that the monitor has been installed/enabled properly, run the following command from ISPF 6:

   ```
   ex 'hi_lev_qual.V1R5M0.REXX(CASESSN)' 'LEVEL'
   ```

   You should then see an ISPF Browse panel that looks similar to this:

   ```
   BROWSE    YOUNG.MSGS.FILE                        Line 00000000 Col 00
   Command ===>                                              Scroll ===>
   ****************************** Top of Data ************************
   Monitor Release: V1R5M0 Date: 1998.346
   MAST: 00F9B8E8 PSA: 00F87000 CPU: 00F87000 SEST: 00F9BCE8 UNID: 00000018
   ```

   Verify that the monitor release is V1R5M0 and the date is 1998.346 or later.

5. To reinstall and re-enable the monitor SVCs after a system IPL, make the following changes:

   a. Add line 000600 in the following example to the SYS1.PARMLIB(COMMND*xx*) data set. (Usually it is COMMND00, however your systems programmer should know if it has been renamed.) The SYS1.PARMLIB(COMMND*xx*) data set contains the names of programs to start at IPL time, and line 000600 installs and enables the ATC monitor SVCs.

```
 000100 COM='MN JOBNAMES,T'
 000110 COM='SET DIAG=01'
 000200 COM='K M,AMRF=N'
 000300 COM='K S,DEL=RD,SEG=20,CON=N,RNUM=20,RTME=1/2,MFORM=M,L=01'
 000400 COM='S JES2,PARM='WARM,NOREQ''
 000500 COM='S IRRDPTAB'
>000600 COM='S MONINSTS'
 000700 COM='S VTAM43P1,,,(LIST=15)'
 000800 COM='S EZAZSSI,P=SPIMVSP1'
 000900 COM='S STARTUP,M=STARTP1'
```

   b. Add the following PROC to your SYS1.PROCLIB data set:

```
//MONINSTS  PROC
//*
//* This is a cataloged proc that can be placed in SYS1.PROCLIB and
//* invoked at IPL time to install/enable the ATC monitor SVCs.
//*
//* The following line must go in the COMMNDxx member of SYS1.PARMLIB
//* that is used during IPL:
//*
//* COM='S MONINSTS'
//*
//* Change the PARM operands to contain the 2 ATC SVC numbers
//* (in HEX).
//*
//ATCMTR    EXEC PGM=MONINSTS,PARM=(FE,FF)
//*
//* The following AUTHLIB must contain MONINSTS and MONSVC.
//*
//STEPLIB   DD DSN=LSTTOOL.TA.APF.LOAD,DISP=SHR
//SYSOUT    DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
```

   c. Make the same changes to this PROCLIB member as you made to 'hi_lev_qual.V1R5M0.SAMPLE.JCL(INSTATC)'.

6. Give the MONINSTS-started task RACF access to the MONINSTS and MONSVC load modules.

Ensuring Users Have Access to AMASPZAP

User access to the SPZAP service aid program (AMASPZAP) is required if the user chooses to place breakpoints in load modules directly (rather than in object modules) via the TOLOADDSN CA/DA/UTA control card keyword. If your users need to do this, ensure that they can execute the AMASPZAP program.

Editing the Site Defaults Data Set

Complete the following steps to edit *hi_lev_qual*.V1R5M0.MASTER.DEFAULTS for your installation:

1. Change all occurrences of ATC.V1R5M0 to *hi_lev_qual*.V1R5M0. For example, if you want to use the high-level qualifier SMITH, change all occurrences of ATC.V1R5M0 to SMITH.V1R5M0. In addition, change all occurrences of ATC.EX to *hi_lev_qual*.EX. Figure 1 on page 22 shows the contents of the site defaults data set for MVS as shipped.
2. In the execute step data entry for EXEJOBLB, change the name LSTTOOL.TA.APF.LOAD to the name of the authorized library you used for the CMDUSVC and VARMON3 programs identified in “Setting up the Authorized Data Sets” on page 16.
3. Enter the ATC Monitor SVC numbers (in hexadecimal notation) in the ATGSVC2B and ATGSVC4B entries.
4. For JCL generation, if your system requires it, change the *JOBL*n* lines. When JCL is created, these will be the first three lines of the JCL (for each respective job). You may also need to change all of the *JOBJL lines to specify any JES control information required by your site. If your installation will primarily run PL/I test cases (instead of COBOL), change the EXMCTLTY line to P.
5. The Language Environment for MVS & VM 1.5 (or later) runtime library is required for some ATC batch jobs. If this library is not available through your system's normal search path for load modules, you should enter its name in the ATCJBLB2 line. (An example would be CEE.V1R5M0.SCEERUN.)
6. If your site **does not** use UNIT=SYSDA for batch data set allocations, then change all occurrences of UNIT=SYSDA to the proper specification for your site. You will need to make the same change in the ATGUTFW, ATSAABG, ATSAABG2, QCOMBINE, QEXECUTE, and QSETUP members of the SKELS data set.
7. If your site **does not** use UNIT=VIO for batch temporary data set allocations, then edit the ATSAABG and ATSAABG2 members of the SKELS data set and change all occurrences of UNIT=VIO to the proper specification for your site.
8. By setting the USEPRGNM variable setting to N, you can tell ATC **not** to generate or build any data set names automatically.

9. If the USEPRGNM variable setting is Y, then you can tell ATC to generate or build each data set as either a sequential or a partitioned data set.
10. To generate a data set as a:
    - Sequential data set, set the DSORG variable after the data set variable to SEQ.
    - Partitioned data set, set the DSORG variable after the data set variable to PDS.

    ATC will use the following forms to generate data set names:

    - For sequential data sets:

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

      For example: 'YOUNG.TEST.COB01M.BRKTAB'
    - For partitioned data sets:

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

      For example: 'YOUNG.TEST.BRKTAB(COB01M)'

```
/*********************************************************************/
| /* ATC DEFAULTS FILE V1R5M0                                                    @DRC */
  /* GENERAL DEFAULTS                                                              */
  /*                                    ISPF                                       */
  /* DEFAULTS DATA                      VARIABLE ANY COMMENTS                      */
| ATC.V1R5M0.REXX                    /* EXCMDDSN  DATASET CONTAINING ATC REXX EXECS       @DRC */
  CAHLQ                              /* PROJQUAL  PROJECT LEVEL QUALIFER FOR MVS DATASET NAMES  */
  CAMEMBER                           /* PROGNAME  PROGRAM NAME-FOR 2ND LEVEL DS QUALIFIER  @D3C */
  Y                                  /* USEPRGNM  USE PROGRAM NAME FOR FILE NAME?               */
  I                                  /* SHOWMSGS  DISPLAY MESSAGES? (N/E/W/R/I)                 */
  I                                  /* LOGMSGS   LOG MESSAGES?     (N/E/W/R/I)                 */
  Y                                  /* CAENABLE  PROCESS FOR CA (Y/N)                          */
  N                                  /* DAENABLE  PROCESS FOR DA (Y/N)                          */
  N                                  /* UTENABLE  PROCESS FOR UTA (Y/N)                         */
  CAHLQ.JCL.CNTL                     /* JCLLIB    NAME OF DATASET TO PUT GENERATED JCL IN       */
  JCL.CNTL                           /* JCLLIBNM  JCLLIB DESIRED TYPE QUALIFIER                 */
  PDS                                /* JCLLIBDT  JCLLIB DESIRED DSORG                          */
  LRECL(80) RECFM(F B) BLKSIZE(0)    /* JCLLIBA0  JCLLIB DESIRED ALLOCATION PARMS               */
  TRACKS SPACE(10 10)                /* JCLLIBA1  JCLLIB DESIRED ALLOCATION PARMS               */
| ATC.V1R5M0.LOADLIB                /* ATCJBLB   NAME OF DATASET WITH ALL ATC LOAD MODULES@DRC */
  NONE                               /* ATCJBLB2  2ND DATASET IN JOBLIB CONCATENATION           */
  NONE                               /* ATCJBLB3  3RD DATASET IN JOBLIB CONCATENATION           */
  NONE                               /* ATCJBLB4  4TH DATASET IN JOBLIB CONCATENATION           */
  NONE                               /* ATCJBLB5  5TH DATASET IN JOBLIB CONCATENATION           */
  NONE                               /* ATCJBLB6  6TH DATASET IN JOBLIB CONCATENATION           */
  CAHLQ.CAMEMBER.CACTL               /* EXMCTLFI  NAME OF CA CONTROL FILE DATASET         @D3C */
  CACTL                              /* EXMCTLNM  EXMCTLFI DESIRED TYPE QUALIFIER               */
  PDS                                /* EXMCTLDT  EXMCTLFI DESIRED DSORG                        */
  LRECL(255) RECFM(V B) BLKSIZE(0)   /* EXMCTLA0  EXMCTLFI DESIRED ALLOCATION PARMS             */
  TRACKS SPACE(10 10)                /* EXMCTLA1  EXMCTLFI DESIRED ALLOCATION PARMS             */
  SPACE=(TRK,(2,2)),UNIT=SYSDA,      /* EXMCTDD1  EXMCTLFI DESIRED DD PARMS               @D4A */
  DCB=(RECFM=VB,LRECL=255,BLKSIZE=0) /* EXMCTDD2  EXMCTLFI DESIRED DD PARMS               @D4A */
  B                                  /* EXMCTLTY  TYPE OF CA CONTROL FILE (B/P/A)         @DLB */
  /*****************************************************************************************************/
  /* SETUP STEP DEFAULTS                                                           */
  /*****************************************************************************************************/
  USERID                             /* STPJOBNM  NAME ON JOB CARD FOR SETUP STEP               */
  (12345678),                        /* STPJOBL1  FIRST LINE OF JOB CARD FOR SETUP STEP         */
  USERID,NOTIFY=USERID,USER=USERID,  /* STPJOBL2  SECOND LINE OF JOB CARD FOR SETUP STEP        */
  MSGCLASS=H,CLASS=A                 /* STPJOBL3  THIRD LINE OF JOB CARD FOR SETUP STEP         */
  NONE                               /* STPJOBL4  FORTH LINE OF JOB CARD FOR SETUP STEP         */
  NONE                               /* STPJOBJL  JES CONTROL CARD FOR SETUP STEP               */
  NONE                               /* STPJOBJ2  JES CONTROL CARD FOR SETUP STEP               */
  NONE                               /* STPJOBJ3  JES CONTROL CARD FOR SETUP STEP               */
  CAHLQ.CAMEMBER.BRKTAB              /* BRKTBFIL  NAME OF BREAKPOINT TABLE DATASET        @D3C */
  BRKTAB                             /* BRKTBNM   BREAKPOINT TABLE NAME                         */
  SEQ                                /* BRKTBDT   BREAKPOINT TABLE DSORG                  @D4A */
  LRECL(256) RECFM(F B) BLKSIZE(4096)/* BRKTBA0   BREAKPOINT TABLE ALLOCATION PARMS       @D4A */
  TRACKS SPACE(2 2)                  /* BRKTBA1   BREAKPOINT TABLE ALLOCATION PARMS       @D4A */
  SPACE=(TRK,(2,2)),UNIT=SYSDA,      /* BRKTBDD1  BREAKPOINT TABLE DD PARMS               @D4C */
  DCB=(DSORG=PS,RECFM=FB,LRECL=256,BLKSIZE=4096)  /* BRKTBDD2  BREAKPOINT TABLE DD PARMS  @D4C */
  CAHLQ.CAMEMBER.DBGTAB              /* DBGTBFIL  NAME OF DEBUG TABLE DATASET             @D3C */
  DBGTAB                             /* DBGTBNM   DEBUG TABLE NAME                              */
  SEQ                                /* DBGTBDT   DEBUG TABLE DSORG                       @D4A */
  LRECL(128) RECFM(F B) BLKSIZE(4096)/* DBGTBA0   DEBUG TABLE ALLOCATION PARMS            @D4A */
  TRACKS SPACE(2 2)                  /* DBGTBA1   DEBUG TABLE ALLOCATION PARMS            @D4A */
  SPACE=(TRK,(2,2)),UNIT=SYSDA,      /* DBGTBDD1  DEBUG TABLE DD PARMS                    @D4C */
  DCB=(DSORG=PS,RECFM=FB,LRECL=128,BLKSIZE=4096)  /* DBGTBDD2  DEBUG TABLE DD PARMS       @D4C */
  CAHLQ.CAMEMBER.VARCTL              /* VARCTFIL  NAME OF VARIABLE CONTROL DATASET        @D3C */
  VARCTL                             /* VARCTNM   VARIABLE CONTROL NAME                         */
  SEQ                                /* VARCTDT   VARIABLE CONTROL DSORG                  @D4A */
  LRECL(64) RECFM(F B) BLKSIZE(0)    /* VARCTA0   VARIABLE CONTROL ALLOCATION PARMS       @DEC */
  TRACKS SPACE(2 2)                  /* VARCTA1   VARIABLE CONTROL ALLOCATION PARMS       @D4A */
  SPACE=(TRK,(2,2)),UNIT=SYSDA,      /* VARCTDD1  VARIABLE CONTROL DD PARMS               @D4C */
  DCB=(DSORG=PS,RECFM=FB,LRECL=64,BLKSIZE=0)      /* VARCTDD2  VARIABLE CONTROL DD PARMS  @DEC */
  FE                                 /* ATGSVC2B  SVC NUMBER (HEX) FOR 2 BYTE BREAKPOINT  @DEA */
  FF                                 /* ATGSVC4B  SVC NUMBER (HEX) FOR 4 BYTE BREAKPOINT  @DEA */
  N                                  /* ATGPRFMD  ENABLE THE MONITOR PERFORMANCE MODE     @DGC */
  NN                                 /* DEBGMODE  USE DEBUG MODE (Y/N) AND FREQUENCY COUNT (Y/N)*/
```

Figure 1 (Part 1 of 3). Site Defaults Data Set

```
/*********************************************************************************************************************/
/* EXECUTE STEP DATA                                                                                                 */
/*********************************************************************************************************************/
LSTTOOL.TA.APF.LOAD                          /* EXEJOBLB  DATASET W/ CA MODULES FOR EXECUTE STEP        */
USERID                                       /* EXEJOBNM  NAME ON JOB CARD FOR EXECUTE STEP             */
(12345678),                                  /* EXEJOBL1  FIRST LINE OF JOB CARD FOR EXECUTE STEP       */
USERID,NOTIFY=USERID,USER=USERID,            /* EXEJOBL2  SECOND LINE OF JOB CARD FOR EXECUTE STEP      */
MSGCLASS=H,CLASS=A,REGION=4M,MSGLEVEL=(1,1)  /* EXEJOBL3  THIRD LINE OF JOB CARD FOR EXECUTE STEP       */
NONE                                         /* EXEJOBL4  FORTH LINE OF JOB CARD FOR EXECUTE STEP       */
NONE                                         /* EXEJOBJL  JES CONTROL CARD FOR EXECUTE STEP             */
NONE                                         /* EXEJOBJ2  JES CONTROL CARD FOR EXECUTE STEP             */
NONE                                         /* EXEJOBJ3  JES CONTROL CARD FOR EXECUTE STEP             */
20                                           /* MINWAIT   MIN TIME TO WRITE BUFFERS (1/100SEC)     @D7A */
800                                          /* MAXWAIT   MAX TIME TO WRITE BUFFERS (1/100SES)     @D7A */
CAHLQ.CAMEMBER.BRKOUT                        /* BRKOTFIL  NAME OF BREAKOUT TABLE DATASET           @D3C */
CAHLQ.CAMEMBER.CMBOUT                        /* COMBROUT  NAME OF COMBINED BREAKOUT TABLE DATASET  @D4C */
BRKOUT                                       /* BRKOTNM   BREAKOUT TABLE NAME                           */
CMBOUT                                       /* COMBRNM   COMBINED BREAKOUT TABLE NAME             @D4A */
SEQ                                          /* BRKOTDT   BREAKOUT TABLE DSORG                     @D4A */
SEQ                                          /* COMBRDT   COMBINED BREAKOUT TABLE DSORG            @D4A */
LRECL(256) RECFM(F B) BLKSIZE(4096)          /* BRKOTA0   BREAKOUT TABLE ALLOCATION PARMS          @D4A */
TRACKS SPACE(3 3)                            /* BRKOTA1   BREAKOUT TABLE ALLOCATION PARMS          @D4A */
SPACE=(TRK,(3,3)),UNIT=SYSDA,                /* BRKOTDD1  BREAKOUT TABLE DD PARMS                  @D4C */
DCB=(DSORG=PS,RECFM=FB,LRECL=256,BLKSIZE=4096)  /* BRKOTDD2  BREAKOUT TABLE DD PARMS                  @D4C */
CAHLQ.CAMEMBER.VARTAB                        /* VARTBFIL  NAME OF VARIABLE TABLE DATASET           @D3C */
VARTAB                                       /* VARTBNM   VARIABLE TABLE NAME                           */
SEQ                                          /* VARTBDT   VARIABLE TABLE DSORG                     @D4A */
LRECL(128) RECFM(F B) BLKSIZE(4096)          /* VARTBA0   VARIABLE TABLE ALLOCATION PARMS          @D4A */
TRACKS SPACE(3 3)                            /* VARTBA1   VARIABLE TABLE ALLOCATION PARMS          @D4A */
SPACE=(TRK,(3,3)),UNIT=SYSDA,                /* VARTBDD1  VARIABLE TABLE DD PARMS                  @D4C */
DCB=(DSORG=PS,RECFM=FB,LRECL=128,BLKSIZE=4096)  /* VARTBDD2  VARIABLE TABLE DD PARMS                  @D4C */
/*********************************************************************************************************************/
/* REPORT STEP DATA                                                                                                  */
/*********************************************************************************************************************/
USERID                                       /* REPJOBNM  NAME ON JOB CARD FOR REPORT STEP              */
(12345678),                                  /* REPJOBL1  FIRST LINE OF JOB CARD FOR REPORT STEP        */
USERID,NOTIFY=USERID,USER=USERID,            /* REPJOBL2  SECOND LINE OF JOB CARD FOR REPORT STEP       */
MSGCLASS=H,CLASS=A                           /* REPJOBL3  THIRD LINE OF JOB CARD FOR REPORT STEP        */
NONE                                         /* REPJOBL4  FORTH LINE OF JOB CARD FOR REPORT STEP        */
NONE                                         /* REPJOBJL  JES CONTROL CARD FOR REPORT STEP              */
NONE                                         /* REPJOBJ2  JES CONTROL CARD FOR REPORT STEP              */
NONE                                         /* REPJOBJ3  JES CONTROL CARD FOR REPORT STEP              */
CAHLQ.CAMEMBER.CBCTL                         /* CBCTLFIL  NAME OF COMBINE INPUT CONTROL FILE DATASET@D3C*/
CBCTL                                        /* CBCTLNAM  CBCTLNAM DESIRED TYPE QUALIFIER               */
PDS                                          /* CBCTLDT   CBCTLNAM DESIRED DSORG                        */
LRECL(255) RECFM(V B) BLKSIZE(0)             /* CBCTLA0   CBCTLNAM DESIRED ALLOCATION PARMS             */
TRACKS SPACE(10 10)                          /* CBCTLA1   CBCTLNAM DESIRED ALLOCATION PARMS             */
CAHLQ.CAMEMBER.REPORT                        /* REPRTFIL  NAME OF REPORT FILE DATASET              @D3C */
CAHLQ.CAMEMBER.SUMMARY                       /* SUMMFIL   NAME OF SUMMARY FILE DATASET             @D3C */
REPORT                                       /* REPRTTYP  REPORT FILE DATASET NAME                      */
SUMMARY                                      /* SUMMTYP   REPORT FILE DATASET NAME                      */
SEQ                                          /* REPRTDT   REPORT FILE DSORG                        @D4A */
LRECL(133) RECFM(F B A) BLKSIZE(27930)       /* REPRTA0   REPORT FILE ALLOCATION PARMS             @D4A */
TRACKS SPACE(10 10)                          /* REPRTA1   REPORT FILE ALLOCATION PARMS             @D4A */
SPACE=(TRK,(10,10)),UNIT=SYSDA,              /* REPRTDD1  REPORT FILE DD PARMS                     @D4C */
DCB=(DSORG=PS,RECFM=FBA,LRECL=133,BLKSIZE=27930) /* REPRTDD2  REPORT FILE DD PARMS                 @D4C */
I                                            /* SUMINEXT  SUMMARY INTERNAL(I) OR EXTERNAL(E)            */
N                                            /* PLXINLIN  SUMMARY INCLUDE PL/X INLINED CODE (I/N)  @DPA */
A                                            /* USEROPT   REPORT ALL(A) OR UNEXECUTED(U) CODE           */
Y                                            /* PRNTREPT  PRINT REPORT FILE DATASET? (Y/N)              */
/*********************************************************************************************************************/
/* UNIT TEST REPORTS DATA                                                                                            */
/*********************************************************************************************************************/
F                                            /* VARREPT   FULL OR COMBINED VAR REPORT              @D2A */
CAHLQ.CAMEMBER.VARID                         /* VARIDFIL  NAME OF VARIABLE ID DATASET              @D3C */
VARID                                        /* VARIDNM   VARIABLE ID NAME                              */
SEQ                                          /* VARIDDT   VARIABLE ID DSORG                        @D4A */
LRECL(255) RECFM(V B) BLKSIZE(27998)         /* VARIDA0   VARIABLE ID ALLOCATION PARMS             @D4A */
```

Figure 1 (Part 2 of 3). Site Defaults Data Set

```
TRACKS SPACE(2 2)                              /* VARIDA1   VARIABLE ID ALLOCATION PARMS            @D4A */
SPACE=(TRK,(2,2)),UNIT=SYSDA,                  /* VARIDDD1  VARIABLE ID DD PARMS                    @D4C */
DCB=(DSORG=PS,RECFM=VB,LRECL=255,BLKSIZE=27998)  /* VARIDDD2  VARIABLE ID DD PARMS                  @D4C */
CAHLQ.CAMEMBER.VARDATA                         /* VARDAFIL  NAME OF VARIABLE DATA DATASET           @D3C */
VARDATA                                        /* VARDANM   VARIABLE DATA NAME                        */
SEQ                                            /* VARDADT   VARIABLE DATA DSORG                     @D4A */
LRECL(255) RECFM(V B) BLKSIZE(27998)           /* VARDAA0   VARIABLE DATA ALLOCATION PARMS          @D4A */
TRACKS SPACE(2 2)                              /* VARDAA1   VARIABLE DATA ALLOCATION PARMS          @D4A */
SPACE=(TRK,(2,2)),UNIT=SYSDA,                  /* VARDADD1  VARIABLE DATA DD PARMS                  @D4C */
DCB=(DSORG=PS,RECFM=VB,LRECL=255,BLKSIZE=27998)  /* VARDADD2  VARIABLE DATA DD PARMS                @D4C */
CAHLQ.CAMEMBER.FWCTL                           /* FWCTLFI   DSNAME OF FILE WARP CONTROL FILE        @DOA */
FWCTL                                          /* FWCTLNM   FILE WARP CONTROL DATASET TYPE NAME     @DOA */
PDS                                            /* FWCTLDT   FILE WARP CONTROL FILE DSORG            @DOA */
/*************************************************************************************************************/
| /* TARGETED SUMMARY DATA                                                                                    */
| /*************************************************************************************************************/
| CAHLQ.CAMEMBER.TARGCTL                         /* FOCCTLFI  DSNAME OF TARGETED SUMMARY CTL  FILE    @D5C */
| TARGCTL                                       /* FOCCTLTP  TARGETED SUMMARY CTL FILE DATA SET TYPE @D5C */
| PDS                                           /* FOCCTLDT  TARGETED SUMMARY CONTROL FILE  DSORG    @D4A */
| /*                                                                                              3@DSD */
| CAHLQ.CAMEMBER.TARGREP                        /* FOCREPFI  DSNAME OF TARGETED SUMMARY REP ORT DATAST@D5C */
| TARGREP                                       /* FOCREPTP  TARGETED SUMMARY REPORT DATASE T TYPE   @D5C */
| SEQ                                           /* FOCREPDT  TARGETED SUMMARY REPORT DATASE T DSORG  @D4A */
/*************************************************************************************************************/
/* SOURCE AUDIT ASSISTANT DEFAULT VALUES                                                                     */
/*************************************************************************************************************/
USERID                                         /* SAAJBNM  NAME ON JOB CARD FOR SAA STEP            @DA1  */
(12345678),                                    /* SAAJBL1  FIRST LINE OF JOB CARD FOR SAA STEP      @DA1  */
USERID,NOTIFY=USERID,USER=USERID,              /* SAAJBL2  SECOND LINE OF JOB CARD FOR SAA STEP     @DA1  */
MSGCLASS=H,CLASS=A                             /* SAAJBL3  THIRD LINE OF JOB CARD FOR SAA STEP      @DA1  */
NONE                                           /* SAAJBL4  FORTH LINE OF JOB CARD FOR SAA STEP      @DA1  */
CAHLQ.SAA.NEW.COBOL(*)                         /* SAANDSN   NEW DSNAME                                    */
CAHLQ.SAA.OLD.COBOL(*)                         /* SAAODSN   OLD DSNAME                                    */
CAHLQ.SAA.CMP                                  /* SAACDSN   CMP DSNAME                                    */
CAHLQ.SAA.LOG                                  /* SAALDSN   LOG DSNAME                                    */
COBOL                                          /* SAALANG   LANGUAGE  (COBOL,PLI,C,C++,LCOB)              */
N                                              /* SAAFCOM   FILTER COMMENTS                               */
N                                              /* SAAFDCL   FILTER DATA DEFINITIONS                       */
N                                              /* SAAFLIN   FILTER REFORMATTED LINES                      */
1                                              /* SAASTART  BEGINNING COLUMN OF COMPARISON          @D1A  */
176                                            /* SAAEND    ENDING COLUMN OF COMPARISON       @D1A @D6C  */
N                                              /* SAAED     JCL EDIT FLAG                           @DA1  */
N                                              /* SAADBCS   SCAN FOR DBCS (Y/N)                     @DLA  */
/*************************************************************************************************************/
/* SAA POST-PROCESSOR DEFAULT VALUES                                                                         */
/*************************************************************************************************************/
CAHLQ.SAA.SEEDLIST                             /* SAASEEDS  SEED LISTDSNAME                         @DMA  */
CAHLQ.SAA.TARGCTL                              /* SAAVARS   TARGET CONTROL TEMPLATE DSN             @DMA  */
CAHLQ.SAA.REPORT                               /* SAAREP    CHANGE VALIDATION REPORT DSN            @DMA  */
```

Figure 1 (Part 3 of 3). Site Defaults Data Set

Verifying the Installation

To ensure that the ATC package has been installed correctly and to give you some familiarity with ATC if you are a new user, we recommend that you run the sample test cases shipped with the package. Run the following test cases as appropriate, depending on the version of the compiler you use:

Tool	Test Cases
CA	PLI01M, PLI012, PLI011, ASM01L, and ASM01H
CA and UTA	COB01M, COB012, and COB01O
UTA	COB02M, COB022, and COB02O
DA	COB11M, COB112, COB11O, PLI11M, PLI112, and PLI111

For details on setting up your user ID to use ATC and then running the samples, see “Basic User Setup” on page 27, “Allocating Data Sets and Testing Installation” on page 33, “Coverage Assistant Samples” on page 45, “Distillation Assistant Samples” on page 133, and “Unit Test Assistant Samples” on page 167. For SAA, see “Source Audit Assistant Samples” on page 281.

Basic User Setup

This chapter contains topics describing ATC customization procedures and setup options. These topics will be of most interest to individuals who are new to ATC.

Modifying Your ATC Defaults

Typically, you do not have to perform any installation or customization procedures; however, if you want to change ATC user defaults, the ATC panels let you change all of the user defaults provided in the site defaults data set (listed on page 13). Using the panels, you can modify your defaults in the following ways:

- Edit your defaults
- Reset your defaults to the system defaults
- Import defaults from a sequential data set
- Export defaults to a sequential data set

Note: If you are changing from one release of ATC to another, it is recommended that you do a Defaults RESET (ATC option 0.2) and reenter any personal changes.

To start changing your ATC user defaults, complete the following steps:

1. Start ATC by selecting ISPF option 6 and entering:

   `EX '`*`hi_lev_qual`*`.V1R5M0.REXX(ATSTART)'`

   where:

   hi_lev_qual is the MVS data set high-level qualifier under which ATC was installed.

   The first panel that you will see is the `ATC Primary Option Menu`, shown in Figure 2.

```
------------------------ ATC Primary Option Menu V1R5M0 ----------------------
Option ===>_

0  Defaults      Manipulate ATC defaults
1  CA/DA/UTA     Coverage, Distillation and Unit Test Assistant
2  SAA           Source Audit Assistant
3  SINFO         SInfo Assistant

Enter X to Terminate
```

Figure 2. ATC Primary Option Menu

2. To specify your ATC user default values, select option `0` from the `ATC Primary Option Menu`. This displays the `Manipulate ATC Defaults` panel shown in Figure 3 on page 28.

© Copyright IBM Corp. 1997, 1999

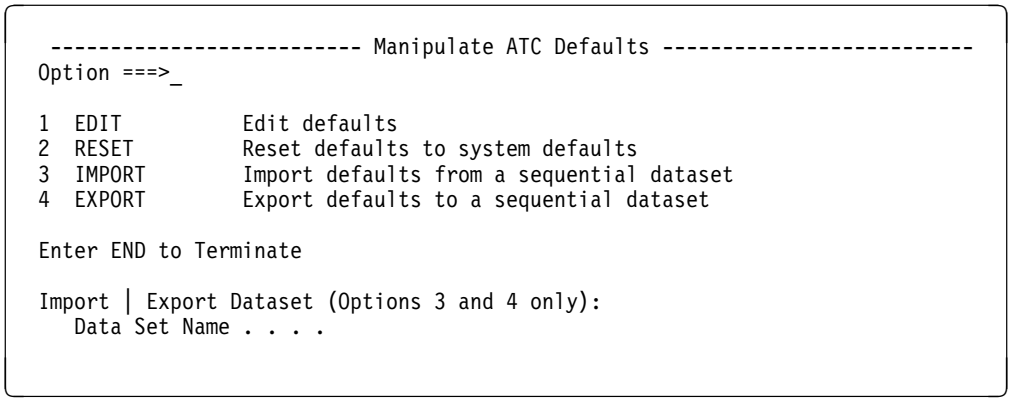


Figure 3. Manipulate ATC Defaults Panel

You can change your ATC user defaults using the options and `Data Set Name` field on this panel. The options and field are as follows:

EDIT Edit your user defaults.

RESET Reset the user defaults to the system default values.

IMPORT Import previously exported default values from the data set specified in `Data Set Name`.

EXPORT Export the current default values to the data set specified in `Data Set Name`.

Data Set Name Name of the data set that is the source or target of the import or export operation, respectively.

The following topics describe editing and resetting your defaults to the system defaults in more detail.

Editing Your User Defaults

To edit your user defaults:

1. Select option `1` from the `Manipulate ATC Defaults` panel. The scrollable panel shown in Figure 4 on page 29 is displayed.
2. If you choose, you can change the `Project Qualifier` value to the high-level qualifier you want ATC to use to construct names for user and project data sets.
3. If you want ATC to generate or build any data set names automatically, make sure Use Pgm Name For File Name is set to Yes. ATC uses the Project Qualifier, the Program Name, and each data set's specified values for Type and DSORG to build names of the following forms:
   - For sequential data sets:

     `'proj_qual.program_name.file_type'`

     For example: 'YOUNG.TEST.COB01M.BRKTAB'
   - For partitioned data sets:

     `'proj_qual.file_type(program_name)'`

     For example: 'YOUNG.TEST.BRKTAB(COB01M)'

If No is specified for Use Pgm Name For File Name, ATC will not build or change any data set names automatically.

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

Enter END (to Exit and Save changes) or CANCEL (to Exit without saving)
---------------------------- General Defaults ----------------------------
Project Qualifier. . . . . YOUNG.TEST
Use Pgm Name for File Name YES        (Yes|No)
Program Name . . . . . . . COB01M
JCL Output Dsn . . . . . . 'YOUNG.TEST.JCL.CNTL'
  Type . . . . . . . . . . JCL.CNTL
  DSORG. . . . . . . . . . PDS        (SEQ|PDS)
  Alloc Parms. . . . . . . LRECL(80) RECFM(F B) BLKSIZE(0)
                           TRACKS SPACE(10 10)
1st JOBLIB Dsn . . . . . . 'ATC.V1R5M0.LOADLIB'
2nd Alternate JOBLIB Dsn .
3rd Alternate JOBLIB Dsn .
4th Alternate JOBLIB Dsn .
5th Alternate JOBLIB Dsn .
6th Alternate JOBLIB Dsn .
REXX Dsn . . . . . . . . . 'ATC.V1R5M0.REXX'
Display Messages . . . . . I          (S|E|W|R|I)
Log Messages . . . . . . . I          (S|E|W|R|I)
Enable CA. . . . . . . . . YES        (Yes|No)
Enable DA. . . . . . . . . NO         (Yes|No)
Enable UTA . . . . . . . . NO         (Yes|No)
Control File Dsn . . . . . 'YOUNG.TEST.CACTL(COB01M)'
  Type . . . . . . . . . . CACTL
  DSORG. . . . . . . . . . PDS        (SEQ|PDS)
  Alloc Parms. . . . . . . LRECL(255) RECFM(V B) BLKSIZE(0)
                           TRACKS SPACE(10 10)
  DD Parms . . . . . . . . SPACE=(TRK,(2,2)),UNIT=SYSDA,
                           DCB=(RECFM=VB,LRECL=255,BLKSIZE=0)
Type of Control File . . . COBOL      (COBOL|PL/I|ASM)
------------------------------ Setup Defaults ------------------------------
Jobcard Name . . . . . . . YOUNG
Jobcard Operands . . . . . (12345678),
                           YOUNG,NOTIFY=YOUNG,USER=YOUNG,
                           MSGCLASS=H,CLASS=A

JES Control Card . . . . .


Breakpoint Table Dsn . . . 'YOUNG.TEST.COB01M.BRKTAB'
  Type . . . . . . . . . . BRKTAB
  DSORG. . . . . . . . . . SEQ        (SEQ|PDS)
  Alloc Parms. . . . . . . LRECL(256) RECFM(F B) BLKSIZE(4096)
                           TRACKS SPACE(2 2)
  DD Parms . . . . . . . . SPACE=(TRK,(2,2)),UNIT=SYSDA,
                           DCB=(DSORG=PS,RECFM=FB,LRECL=256,BLKSIZE=4096)
Debug Table Dsn. . . . . . 'YOUNG.TEST.COB01M.DBGTAB'
  Type . . . . . . . . . . DBGTAB
  DSORG. . . . . . . . . . SEQ        (SEQ|PDS)
  Alloc Parms. . . . . . . LRECL(128) RECFM(F B) BLKSIZE(4096)
                           TRACKS SPACE(2 2)
  DD Parms . . . . . . . . SPACE=(TRK,(2,2)),UNIT=SYSDA,
                           DCB=(DSORG=PS,RECFM=FB,LRECL=128,BLKSIZE=4096)
Variable Cntl Dsn. . . . . 'YOUNG.TEST.COB01M.VARCTL'
  Type . . . . . . . . . . VARCTL
  DSORG. . . . . . . . . . SEQ        (SEQ|PDS)
  Alloc Parms. . . . . . . LRECL(64) RECFM(F B) BLKSIZE(0)
                           TRACKS SPACE(2 2)
```

Figure 4 (Part 1 of 3). ATC Defaults Panel

```
  DD Parms . . . . . SPACE=(TRK,(2,2)),UNIT=SYSDA,
                     DCB=(DSORG=PS,RECFM=FB,LRECL=64,BLKSIZE=0)
SVC number for 2 byte BP . FE         (in HEX)
SVC number for 4 byte BP . FF         (in HEX)
Performance Mode . . . . . NO         (Yes|No)
Debug Mode . . . . . NO         (Yes|No)
Frequency Count Mode . . . NO         (Yes|No)
------------------------------ Monitor Defaults ------------------------------
Loadlib Dsn. . . . . 'LSTTOOL.TA.APF.LOAD'
Jobcard Name . . . . . YOUNG
Jobcard Operands . . . . . (12345678),
                     YOUNG,NOTIFY=YOUNG,USER=YOUNG,
                     MSGCLASS=H,CLASS=A,REGION=4M,MSGLEVEL=(1,1)

JES Control Card . . . . .


Min Wait time to Write Buf 20         (1/100 secs)
Max Wait time to Write Buf 800        (1/100 secs)
Breakout Table Dsn . . . . 'YOUNG.TEST.COB01M.BRKOUT'
  Type . . . . . BRKOUT
  DSORG. . . . . SEQ        (SEQ|PDS)
  Alloc Parms. . . . . LRECL(256) RECFM(F B) BLKSIZE(4096)
                     TRACKS SPACE(3 3)
  DD Parms . . . . . SPACE=(TRK,(3,3)),UNIT=SYSDA,
                     DCB=(DSORG=PS,RECFM=FB,LRECL=256,BLKSIZE=4096)
Variable Table Dsn . . . . 'YOUNG.TEST.COB01M.VARTAB'
  Type . . . . . VARTAB
  DSORG. . . . . SEQ        (SEQ|PDS)
  Alloc Parms. . . . . LRECL(128) RECFM(F B) BLKSIZE(4096)
                     TRACKS SPACE(3 3)
  DD Parms . . . . . SPACE=(TRK,(3,3)),UNIT=SYSDA,
                     DCB=(DSORG=PS,RECFM=FB,LRECL=128,BLKSIZE=4096)
------------------------- Coverage Report Defaults -------------------------
Combined Cntl Dsn. . . . . 'YOUNG.TEST.CBCTL(COB01M)'
  Type . . . . . CBCTL
  DSORG. . . . . PDS        (SEQ|PDS)
  Alloc Parms. . . . . LRECL(255) RECFM(V B) BLKSIZE(0)
                     TRACKS SPACE(10 10)
Combined Breakout Dsn. . . 'YOUNG.TEST.COB01M.CMBOUT'
  Type . . . . . CMBOUT
  DSORG. . . . . SEQ        (SEQ|PDS)
Jobcard Name . . . . . YOUNG
Jobcard Operands . . . . . (12345678),
                     YOUNG,NOTIFY=YOUNG,USER=YOUNG,
                     MSGCLASS=H,CLASS=A

JES Control Card . . . . .


Report File Dsn. . . . . 'YOUNG.TEST.COB01M.REPORT'
Summary File Dsn . . . . . 'YOUNG.TEST.COB01M.SUMMARY'
  Report File Type . . . . REPORT
  Summary File Type. . . . SUMMARY
  DSORG. . . . . SEQ        (SEQ|PDS)
```

Figure 4 (Part 2 of 3). ATC Defaults Panel

```
  Alloc Parms. . . . . . LRECL(133) RECFM(F B A) BLKSIZE(27930)
                         TRACKS SPACE(10 10)
  DD Parms . . . . . . . SPACE=(TRK,(10,10)),UNIT=SYSDA,
                         DCB=(DSORG=PS,RECFM=FBA,LRECL=133,BLKSIZE=27930)
Summary Type . . . . . . INTERNAL   (Internal|External)
Summary PL/X Inline. . . N          (I|N)
Report User Options. . . A          (A|U)
Print Report File Dataset. YES      (Yes|No)
------------------------ Unit Test Report Defaults ------------------------
Variable Report Type . . . FULL     (Full|Combine)
Variable ID Dsn. . . . . . 'YOUNG.TEST.COB01M.VARID'
  Type . . . . . . . . . . VARID
  DSORG. . . . . . . . . . SEQ      (SEQ|PDS)
  Alloc Parms. . . . . . . LRECL(255) RECFM(V B) BLKSIZE(27998)
                         TRACKS SPACE(2 2)
  DD Parms . . . . . . . SPACE=(TRK,(2,2)),UNIT=SYSDA,
                         DCB=(DSORG=PS,RECFM=VB,LRECL=255,BLKSIZE=27998)
Variable Data Dsn. . . . . 'YOUNG.TEST.COB01M.VARDATA'
  Type . . . . . . . . . . VARDATA
  DSORG. . . . . . . . . . SEQ      (SEQ|PDS)
  Alloc Parms. . . . . . . LRECL(255) RECFM(V B) BLKSIZE(27998)
                         TRACKS SPACE(2 2)
  DD Parms . . . . . . . SPACE=(TRK,(2,2)),UNIT=SYSDA,
                         DCB=(DSORG=PS,RECFM=VB,LRECL=255,BLKSIZE=27998)
File Warp Control Dsn. . . 'YOUNG.TEST.FWCTL(COB01M)'
  Type . . . . . . . . . . FWCTL
  DSORG. . . . . . . . . . PDS      (SEQ|PDS)
------------------------ Targeted Summary Defaults ------------------------
Targeted Summary Ctl Dsn   'YOUNG.TEST.TARGCTL(COB01M)'
  Type . . . . . . . . . . TARGCTL
  DSORG. . . . . . . . . . PDS      (SEQ|PDS)
  Alloc Parms. . . . . . .

Targeted Sum. Report Dsn . 'YOUNG.TEST.COB01M.TARGREP'
  Type . . . . . . . . . . TARGREP
  DSORG. . . . . . . . . . SEQ      (SEQ|PDS)
  Alloc Parms. . . . . . .
---------------------- Source Audit Assistant Defaults ----------------------
Jobcard Name . . . . . . . YOUNG
Jobcard Operands . . . . . (12345678),
                         YOUNG,NOTIFY=YOUNG,USER=YOUNG,
                         MSGCLASS=H,CLASS=A

Unmodified Source Dsn. . . 'YOUNG.TEST.SAA.OLD.COBOL(*)'
Modified Source Dsn. . . . 'YOUNG.TEST.SAA.NEW.COBOL(*)'
Output Compare Dsn . . . . 'YOUNG.TEST.SAA.CMP'
Output Log Dsn . . . . . . 'YOUNG.TEST.SAA.LOG'
Programming Language . . . COBOL    (ASM|LASM|C|C++|COBOL|LCOB|PL/I|LPLI)
Filter Comments. . . . . . N        (Y|N)
Filter Data Definitions. . N        (Y|N)
Filter Reformatted Lines . N        (Y|N)
Start Column . . . . . . . 1        (1-176|blank)
End Column . . . . . . . . 176      (1-176|blank)
Edit JCL . . . . . . . . . N        (Y|N)
Enable DBCS. . . . . . . . N        (Y|N)
----------------------- SAA Post-Processor Defaults -----------------------
Seed List Dsn. . . . . . . 'YOUNG.TEST.SAA.SEEDLIST'
Target Control Dsn . . . . 'YOUNG.TEST.SAA.TARGCTL'
Change Validation Rpt Dsn. 'YOUNG.TEST.SAA.REPORT'
```

Figure 4 (Part 3 of 3). ATC Defaults Panel

Resetting Your User Defaults to the System Defaults

To reset your user defaults to the site defaults:

1. Select option 2 from the `Manipulate ATC Defaults` panel. This displays the panel shown in Figure 5.

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

Project Qualifier. . . . . YOUNG.TEST
Program Name . . . . . . . COB01M

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

Figure 5. Reset Defaults to System Defaults Panel

The panel's fields are as follows:

Project Qualifier Specifies the qualifier to be used to construct file names for user and project files.

Program Name Specifies the file_type qualifier to be used when file names for user and project files. `Use Program Name for File Name` is set to Yes.

2. If you want to reset all of your user defaults to the system 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.

Allocating Data Sets and Testing Installation

ATC provides samples for CA, DA, and UTA for each of the supported languages. This section describes how to set up your proj_qual data sets before running the samples. For SAA samples, see "Source Audit Assistant Samples" on page 281.

1. Allocate the following data sets:
   a. proj_qual.JCL.CNTL with DSORG=PO, RECFM=FB, and LRECL=80.
   b. proj_qual.ZAPOBJ with DSORG=PO, RECFM=FB, LRECL=80, BLKSIZE=3200.
   c. proj_qual.RUNLIB with DSORG=PO, RECFM=U, BLKSIZE=23200.
2. Test the installation for each supported compiler:
   a. Test the CA and UTA installation for the COBOL for MVS & VM compiler[3] by copying the following from *hi_lev_qual*.V1R5M0.SAMPLE.JCL to the JCL data set you allocated in step 1:

      COMPCOBM (Compiles test cases.)
      LCOB01M (Links COB01M test case.)
      GCOB01M (Executes COB01M.)

      For information on running this sample, see "Coverage Assistant Samples" on page 45 and "Unit Test Assistant Samples" on page 167.

   b. Test the CA and UTA installation for the VS COBOL II compiler by copying the following from *hi_lev_qual*.V1R5M0.SAMPLE.JCL to the JCL data set you allocated in 1:

      COMPCOB2 (Compiles test cases.)
      LCOB012 (Links COB012 test case.)
      GCOB012 (Executes COB012.)

      For information on running this sample, see "Coverage Assistant Samples" on page 45 and "Unit Test Assistant Samples" on page 167.

   c. Test the CA and UTA installation for the OS/VS COBOL compiler by copying the following from *hi_lev_qual*.V1R5M0.SAMPLE.JCL to the JCL data set you allocated in 1:

      COMPCOBO (Compiles test cases.)
      LCOB01O (Links COB01O test case.)
      GCOB01O (Executes COB01O.)

      For information on running this sample, see "Coverage Assistant Samples" on page 45 and "Unit Test Assistant Samples" on page 167.

[3] You can also test the CA and UTA installation for the COBOL for OS/390 & VM compiler by copying the same JCL members and making minor edits to the compiler, link-edit, and runtime library names.

d. Test the CA installation for the PL/I for MVS & VM 1.1.1 compiler by copying the following from *hi_lev_qual*.V1R5M0.SAMPLE.JCL to the JCL data set you allocated in 1 on page 33:

   COMPPLIM (Compiles test cases.)
   LPLI01M (Links PLI01M test case.)
   GPLI01M (Executes PLI01M.)

   For information on running this sample, see "Coverage Assistant Samples" on page 45.

e. Test the CA installation for the PL/I 2.3.0 compiler by copying the following from *hi_lev_qual*.V1R5M0.SAMPLE.JCL to the JCL data set you allocated in 1 on page 33:

   COMPPLI2 (Compiles test cases.)
   LPLI012 (Links PLI012 test case.)
   GPLI012 (Executes PLI012.)

   For information on running this sample, see "Coverage Assistant Samples" on page 45.

f. Test the CA installation for the PL/I 1.5.1 compiler by copying the following from *hi_lev_qual*.V1R5M0.SAMPLE.JCL to the JCL data set you allocated in 1 on page 33:

   COMPPLI1 (Compiles test cases.)
   LPLI011 (Links PLI011 test case.)
   GPLI011 (Executes PLI011.)

   For information on running this sample, see "Coverage Assistant Samples" on page 45.

g. Test the DA installation for the COBOL for MVS & VM compiler[4] by copying the following from *hi_lev_qual*.V1R5M0.SAMPLE.JCL to the JCL data set you allocated in 1 on page 33:

   COMPCOBM (Compiles test cases.)
   LCOB11M (Links COB11M test case.)
   GCOB11M (Executes COB11M.)

   For information on running this sample, see "Distillation Assistant Samples" on page 133.

h. Test the DA installation for the VS COBOL II compiler by copying the following from *hi_lev_qual*.V1R5M0.SAMPLE.JCL to the JCL data set you allocated in 1 on page 33:

   COMPCOB2 (Compiles test cases.)
   LCOB112 (Links COB112 test case.)
   GCOB112 (Executes COB112.)

   For information on running this sample, see "Distillation Assistant Samples" on page 133.

---

4 You can also test the DA installation for the COBOL for OS/390 & VM compiler by copying the same JCL members and making minor edits to the compiler, link-edit, and runtime library names.

i. Test the DA installation for the OS/VS COBOL compiler by copying the following from *hi_lev_qual*.V1R5M0.SAMPLE.JCL to the JCL data set you allocated in 1 on page 33:

   COMPCOBO (Compiles test cases.)
   LCOB11O (Links COB11O test case.)
   GCOB11O (Executes COB11O.)

   For information on running this sample, see "Distillation Assistant Samples" on page 133.

j. Test the DA installation for the PL/I for MVS & VM 1.1.1 compiler by copying the following from *hi_lev_qual*.V1R5M0.SAMPLE.JCL to the JCL data set you allocated in 1 on page 33:

   COMPPLIM (Compiles test cases.)
   LPLI11M (Links PLI11M test case.)
   GPLI11M (Executes PLI11M.)

   For information on running this sample, see "Distillation Assistant Samples" on page 133.

k. Test the DA installation for the PL/I 2.3.0 compiler by copying the following from *hi_lev_qual*.V1R5M0.SAMPLE.JCL to the JCL data set you allocated in 1 on page 33:

   COMPPLI2 (Compiles test cases.)
   LPLI112 (Links PLI112 test case.)
   GPLI112 (Executes PLI112.)

   For information on running this sample, see "Distillation Assistant Samples" on page 133.

l. Test the DA installation for the PL/I 1.5.1 compiler by copying the following from *hi_lev_qual*.V1R5M0.SAMPLE.JCL to the JCL data set you allocated in 1 on page 33:

   COMPPLI1 (Compiles test cases.)
   LPLI111 (Links PLI111 test case.)
   GPLI111 (Executes PLI111.)

   For information on running this sample, see "Distillation Assistant Samples" on page 133.

m. Test the CA installation for Assembler H and the High Level Assembler by copying the following from *hi_lev_qual*.V1R5M0.SAMPLE.JCL to the JCL data set you allocated in 1 on page 33:

   COMPASMH (Compiles Assembler H test cases.)
   COMPASML (Compiles High Level Assembler test cases.)
   LASM01H (Links ASM01H test case.)
   LASM01L (Links ASM01L test case.)
   GASM01H (Executes ASM01H.)
   GASM01L (Executes ASM01L.)

   For information on running this sample, see "Coverage Assistant Samples" on page 45.

The recommended approach for running the samples is to use the supplied compiler listings and object and edited copies of the link-edit and GO JCL as input and to generate all outputs under proj_qual. In order to do this, edit the L* and G* members of your proj_qual.JCL.CNTL data set and change the names of the ZAPOBJ and RUNLIB data sets to match your proj_qual data set names. Change the *hi_lev_qual* of the OBJ data sets to match the *hi_lev_qual* under which ATC was installed. Change any compiler link-edit and runtime libraries to use your system data set names for these libraries.

If you want to recompile the samples, allocate listing and OBJ data sets under proj_qual (see the shipped sample data sets for appropriate allocation parameters), and then edit the COMP* members of your proj_qual.JCL.CNTL data set to point the listing and OBJ data sets to your proj_qual data sets.

Accessing the Language Environment Runtime Library

The Language Environment for MVS & VM 1.5 (or later) runtime library is required for some ATC functions. For batch jobs, see "Editing the Site Defaults Data Set" on page 20. For the foreground functions (only the Targeted Summary function, at this time), this library must be available either through your system's normal search path for load modules or through a STEPLIB, TSOLIB, or ISPLLIB mechanism for your TSO session. This runtime library is typically named CEE.V1R5M0.SCEERUN.

Running the Sample Test Cases

If you are a new user, you can become familiar with ATC by running the sample test cases shipped with the package. Run the following test cases as appropriate, depending on the version of the compiler you use:

Tool	Test Cases
CA	PLI01M, PLI012, PLI011, ASM01H, and ASM01L
CA and UTA	COB01M, COB012, and COB01O
UTA	COB02M, COB022, and COB02O
DA	COB11M, COB112, COB11O, PLI11M, PLI112, and PLI111

For details on running the samples, see "Coverage Assistant Samples" on page 45, "Distillation Assistant Samples" on page 133, and "Unit Test Assistant Samples" on page 167. For SAA, see "Source Audit Assistant Samples" on page 281.

Using Coverage Assistant

© Copyright IBM Corp. 1997, 1999

Introduction

This chapter contains the following topics:

- What Is Coverage Assistant?
- What Does CA Require?
- How Does CA Work?
- Where Can You Get Further Details?

What Is Coverage Assistant?

Coverage Assistant (CA) contains programs that allow you to execute application programs in a test environment and retrieve information that helps you determine which code statements have been executed. This process is called measuring test case coverage.

CA supports applications generated by the following compilers and assemblers:

- IBM COBOL for OS/390 & VM 2.1 (plus Millennium Language Extensions [MLE])
- IBM COBOL for MVS & VM 1.2 (plus Millennium Language Extensions [MLE])
- IBM VS COBOL II Release 4.0
- IBM OS/VS COBOL Release 2.4
- IBM PL/I for MVS & VM 1.1.1 (plus Millennium Language Extensions [MLE])
- IBM OS PL/I Optimizing Compiler 2.3.0
- IBM PL/I Optimizing Compiler 1.5.1
- IBM High Level Assembler Version 1 Release 2 and Release 3
- IBM Assembler H Version 2

For each COBOL paragraph, PL/I procedure, ON-unit, Begin-block, or assembler listing, CA can provide you with:

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

In addition, targeted summary reports allow you to obtain similar information, which is “targeted” on specific statements of interest. You can specify a statement by its number, or you can specify all statements that reference specific COBOL or PL/I variables.

© Copyright IBM Corp. 1997, 1999

CA has the following characteristics:

- Low overhead

  For a test case coverage run, CA typically adds very little to the execution time of the program. CA inserts SVC instructions into the 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. By using this technique, the increase in test program execution time is minimal.

- Panel-driven user interface

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

What Does CA Require?

CA has the following requirements:

- CA runs under MVS. Detailed MVS system resource requirements for CA are described in Appendix B, "ATC Requirements and Resources" on page 383. CA uses ISPF services to display dialogs and to produce the JCL to run the CA steps.
- As part of its input, CA requires listings created by the application program compilers and assemblers that it supports. These compilers and assemblers offer options that allow you to include assembler statements in the listings. CA uses these statements to determine where to insert breakpoints.
- CA also requires the application program object modules as input. CA creates copies of these object modules with breakpoints inserted into them.

See "Execution" on page 42 for a description of how the CA authorized programs intercept breakpoints. For CA program runtime library requirements, see "Accessing the Language Environment Runtime Library" on page 36. No change to the runtime environment of the programs you are testing is required.

How Does CA Work?

Running CA consists of the following steps. This list is an overview of the process. **Each activity is described in more detail in topics that follow in this chapter.**

Step 1 Setup

a. Compile the source code that you want to analyze, using the required compiler options.

b. Generate CA JCL using the CA ISPF dialog:

   1) Edit the CA control file.
   2) Create the setup JCL.
   3) Create the monitor JCL.
   4) Create the report or summary JCL.

c. Edit the link-edit JCL to include the modified object modules, which are created by the setup step. Note that you can also instrument load modules after your build process.

d. Edit the GO JCL (or program invocation) to point to the instrumented load module from step 1c.

Step 2 Execution

a. Run the setup JCL (created at step 1b2).
b. Run the link-edit JCL (created at step 1c).
c. Run the JCL to start a monitor session (created at step 1b3).
d. Run your application using the load module(s) from step 2b.
e. Stop the monitor session (with the CASTOP command).

Step 3 Report

a. Run the report or summary JCL (created at step 1b4).
b. Run as many targeted summary reports as you want.

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.

Figure 6 shows a diagram of the entire process.

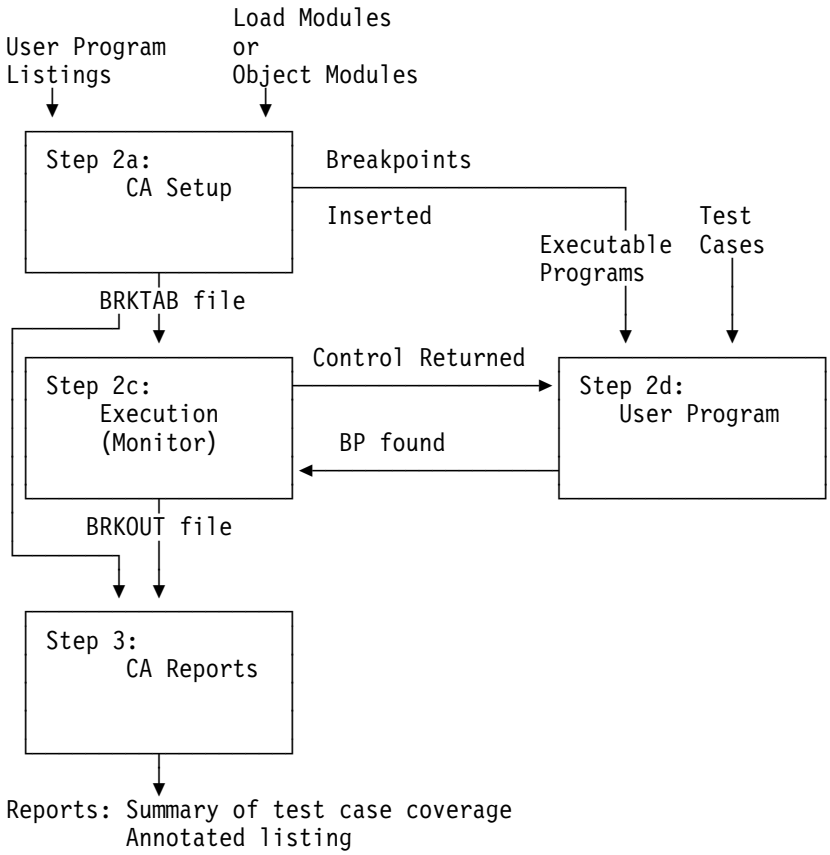


Figure 6. CA—Flow Diagram

Setup

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

Setup runs in MVS. To run it, you need:

1. Source/assembler listings of the object modules
2. The object modules or load modules you want to test.

The Setup step produces:

1. Modified test programs containing breakpoints
2. A file of breakpoint-related information (called a BRKTAB in this User's Guide) required for the monitor program in the Execution step.

Execution

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

Start a monitor session and 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 in this User's Guide.

The monitor inserts reserved supervisor call (SVC) instructions as breakpoints and is given control by MVS when these SVC instructions are executed in a program. Using SVCs as breakpoints is the architected way to receive control from MVS, and requires no modification to MVS. This technique is called user SVCs.

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 CA report program uses the results from the monitor to produce summary reports and annotated listings. You can print a summary report of overall test case coverage and annotated listings for each module tested. These reports are described at "Coverage Assistant Reports" on page 83.

To run reports you need:

1. The BRKTAB data set from the Setup step
2. The BRKOUT data set from the Execution step
3. Any listings you want to annotate

In addition to the basic CA reports, you can also produce *targeted summary* reports, which allow you to target certain statements and/or COBOL or PL/I variables. The format of a targeted summary report is identical to the format of a summary report, except that the content is restricted to statements that you specify. (You can specify a statement number, or you can specify all statements that reference specific COBOL or PL/I variables.)

To run targeted summary reports you need:

1. A control file that specifies the statements and/or COBOL or PL/I variables of interest
2. The BRKTAB data set from the Setup step
3. The BRKOUT data set from the Execution step

Where Can You Get Further Details?

Refer to the following sections for additional information.

For information about...	See...
Installing ATC on your system	"System Installation" on page 13
Samples of CA test case coverage, including sample reports	"Coverage Assistant Samples" on page 45
Editing the CACTL file, which contains the names of the listing data sets	"Editing the Coverage Assistant Control File" on page 81
Setting up the table of breakpoints from the listings	"CA, DA, and UTA Setup" on page 231
Starting the monitor session and running test cases on your programs	"Monitor Execution" on page 245
Reports on the test run	"Coverage Assistant Reports" on page 83
Using CA in a large project environment	"Using Coverage Assistant in a Large Project Environment" on page 115
Commands that control the monitor program	"Monitor Commands" on page 255
System resources needed by CA	Appendix B, "ATC Requirements and Resources" on page 383

Coverage Assistant Samples

This section describes samples of CA test case coverage support using examples provided with the ATC package.

CA provides statistics on 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
- An assembler listing

CA 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.

The samples include a summary of test case coverage and annotated listings for several COBOL, PL/I, and assembler programs:

- COB01M (COBOL for IBM MVS & VM 1.2 sample)[5]
- COB012 (IBM VS COBOL II Release 4.0 sample)
- COB01O (IBM OS/VS COBOL Release 2.4 sample)
- PLI01M (IBM PL/I for MVS & VM 1.1.1 sample)
- PLI012 (IBM OS PL/I Optimizing Compiler 2.3.0 sample)
- PLI011 (IBM PL/I Optimizing Compiler 1.5.1 sample)
- ASM01L (IBM High Level Assembler Version 1 Release 2 sample)
- ASM01H (IBM Assembler H Version 2 sample)

A flow diagram of the steps required to run these samples is shown in Figure 7 on page 46. The names in the steps are the member names of the JCL executed for the step. (For example: SCOB*nnx*, where *nnx* is 01M, 012, or 01O; SPLI*nnx*, where *nnx* is 01M, 012, or 011; or SASM*nnx*, where *nnx* is 01L or 01H.)

The following CA samples use the ATC ISPF dialog to create the JCL to run the CA steps. The ATC ISPF dialog is provided as an aid in creating the JCL. Once the JCL is created for a test environment, it does not have to be recreated from the dialog. In a typical user test environment, the creation of the JCL can be incorporated into the user's procedures. You do not have to use the ISPF dialog to use CA. For information on integrating the creation and running of the CA JCL, see “Using Coverage Assistant in a Large Project Environment” on page 115.

[5] You can also test the CA and UTA installation for the COBOL for OS/390 & VM compiler by copying the JCL members and making minor edits to the compiler, link-edit, and runtime library names.

© Copyright IBM Corp. 1997, 1999

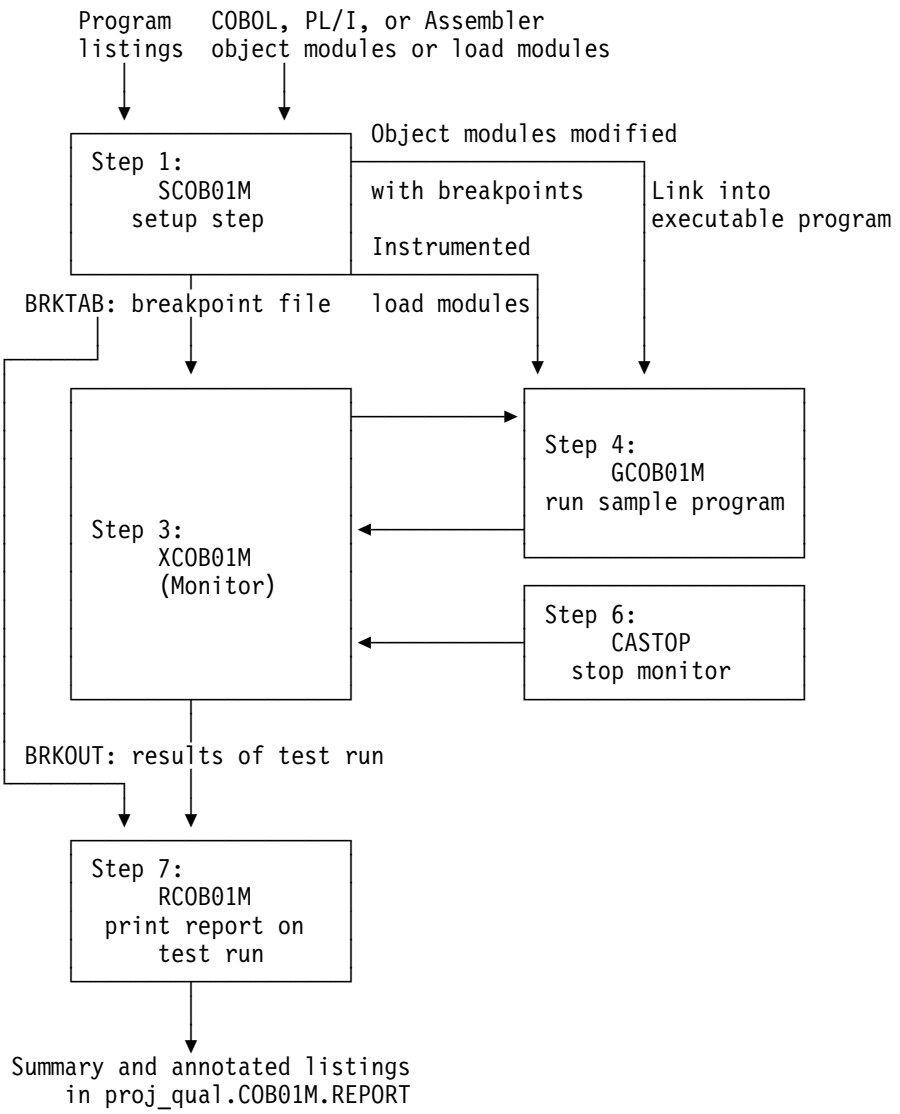


Figure 7. Sample Run—Flow Diagram

Note: The JCL member names for the steps, such as SCOB01M, depend on the test case you are running. For example, the setup JCL for the PLI01M test case would be SPLI01M.

COBOL Summary of Test Case Coverage

The content of the summary report is the same regardless of the compiler for which it is created. For COBOL programs, breakpoints are inserted into the object modules during Setup. When you are ready to test your program, you link the object modules that have been modified with breakpoints.

Figure 8 on page 48 is a summary of a COBOL program called COB01M. To produce a summary for COB01M, perform the following steps. **Steps 2a through 6 are described in more detail in topics that follow in this chapter.**

1. Compile the COBOL source you want to test. This produces listings that include the assembler statements needed by CA. (This has already been done for the COB01M example. The listings are in *hi_lev_qual*.V1R5M0.SAMPLE.COBOLST for the COBOL for MVS & VM and VS COBOL II compilers, and in *hi_lev_qual*.V1R5M0.SAMPLE.COSVSLST for the OS/VS compiler.)

   Make sure to use the compiler options specified in “Setup” on page 231.

2. Start the ATC ISPF dialog by entering the following from ISPF option `6`:

   ```
   EX 'hi_lev_qual.V1R5M0.REXX(ATSTART)'
   ```

   The `ATC Primary Option Menu` is displayed (shown in Figure 2 on page 27).

   a. Edit the CA control file.

      Verify that the control file includes the listings of the object modules you want to test.

   b. Create the setup JCL.

      Create the JCL that enables the Setup job to produce a file containing breakpoint data and to instrument programs. You can instrument either object modules or load modules.

   c. Create the monitor JCL.

      Create the JCL to start a monitor session.

   d. Create the summary report JCL.

      Create the JCL to produce the test case coverage summary after COB01M has executed.

3. End the ATC ISPF dialog by pressing the End key (PF3) on the `ATC Primary Option Menu`.

4. If you instrumented object modules, create the JCL to link the modified object modules.

   After the Setup step, and before starting the monitor session, you must link the modified object modules into an executable program 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. Create the JCL to run the GO step.

   Create the JCL to run your program. Specify the same modified load module as in step 4 on page 47.

6. Execute the JCL.

   Execute the created JCL files for COB01M in the correct order. (This order is shown in “Execute the JCL” on page 53.)

Figure 8 shows a sample summary report. For details about the information included in the report, see “Summary Coverage Report” on page 83.

```
1 *********  CA SUMMARY:                    PROGRAM AREA DATA                    ********
0                DATE: 02/15/1999
                 TIME: 11:47.41
          TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                           |                                        |  STATEMENTS:          | BRANCHES:          |
  | PA LOAD MOD PROCEDURE     | LISTING NAME                           |   TOTAL     EXEC   %  |  CPATH   TAKEN   %  |
  ------------------------------------------------------------------------------------------------------------------------
     1 COB01M                  ATC.V1R5M0.SAMPLE.COBOLST(COB01AM)            6        6 100.0       0       0 100.0
     2          PROGA                                                        8        7  87.5       6       5  83.3
     3          PROCA                                                        1        0   0.0       0       0 100.0
     4          PROGB                                                        7        5  71.4       6       3  50.0
     5          PROCB                                                        2        2 100.0       0       0 100.0
     6 COB01M   PROGC           ATC.V1R5M0.SAMPLE.COBOLST(COB01CM)            7        5  71.4       6       5  83.3
     7          PROCC                                                        3        2  66.7       2       1  50.0
     8 COB01M   PROGD           ATC.V1R5M0.SAMPLE.COBOLST(COB01DM)            6        0   0.0       6       0   0.0
     9          PROCD                                                        1        0   0.0       0       0 100.0
  ------------------------------------------------------------------------------------------------------------------------
  Summary for all PAs:                                                       41       27  65.9      26      14  53.8

1 *********  CA SUMMARY:                    UNEXECUTED CODE                      ********
0                DATE: 02/15/1999
                 TIME: 11:47.41
          TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                           |                                        |
  | PA LOAD MOD PROCEDURE     | LISTING NAME                           | start    end     start    end     start   end
  ------------------------------------------------------------------------------------------------------------------------
     2 COB01M   PROGA           ATC.V1R5M0.SAMPLE.COBOLST(COB01AM)          67     67
     3          PROCA                                                       79     79
     4          PROGB                                                      118    119
     6 COB01M   PROGC           ATC.V1R5M0.SAMPLE.COBOLST(COB01CM)          39     40
     7          PROCC                                                       58     58
     8 COB01M   PROGD           ATC.V1R5M0.SAMPLE.COBOLST(COB01DM)          37     46
     9          PROCD                                                       51     51
  ------------------------------------------------------------------------------------------------------------------------

1 *********  CA SUMMARY:                    BRANCHES THAT HAVE NOT GONE BOTH WAYS ********
0                DATE: 02/15/1999
                 TIME: 11:47.41
          TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                           |                                        |
  | PA LOAD MOD PROCEDURE     | LISTING NAME                           |  stmt      stmt      stmt      stmt      stmt
  ------------------------------------------------------------------------------------------------------------------------
     2 COB01M   PROGA           ATC.V1R5M0.SAMPLE.COBOLST(COB01AM)          66
     4          PROGB                                                      113       118
     6 COB01M   PROGC           ATC.V1R5M0.SAMPLE.COBOLST(COB01CM)          37
     7          PROCC                                                       57
     8 COB01M   PROGD           ATC.V1R5M0.SAMPLE.COBOLST(COB01DM)          37        41        45
  ------------------------------------------------------------------------------------------------------------------------
```

Figure 8. Summary Reports for COB01M

Edit the CA Control File

CA uses assembler statements from the compiler listings to determine where to insert breakpoints. You supply the names of the listing files in the CA control file (CACTL).

Make sure to use the compiler options specified in “Setup” on page 231.

The CACTL control file for the COBOL summary example is *hi_lev_qual*.V1R5M0.SAMPLE.CACTL(COB01M). The file is shown in Figure 9 on page 50. This example shows how to instrument object modules. To instrument load modules, see “Instrumentation of Load Modules instead of Object Modules” on page 238.

To edit the CACTL for the COB01M summary example:

1. Select option `1` from the `ATC Primary Option Menu`.

   The `Coverage, Distillation, and Unit Test Assistant` panel is displayed.

2. Select option `1`.

   The `Work with the CA/DA/UTA Control File` panel is displayed.

3. Specify the following:

Use Program Name for File Name	`YES`
Program Name	`COB01M`
Listing Type	`COBOL`

   An ISPF edit session for the CA control file you requested is displayed.

   The data in the control file consists of the following:

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

   If the control file you requested did not previously exist, it is created with comments in it to help you enter the appropriate information in the fields.

   If you want to use the shipped sample CACTL member as a template for your control file, delete the existing lines in the member and copy in *hi_lev_qual*.V1R5M0.SAMPLE.CACTL(COB01M). Change ATC. to *hi_lev_qual*. for all occurrences. Change the value of the ToObjDsn operand to proj_qual.ZAPOBJ. If you have recompiled the samples into proj_qual data sets, you will also have to edit the ListDsn and FromObjDsn keyword operands to point to your proj_qual listing and object data set names.

   The control file shown in Figure 9 on page 50 is the control file for COB01M. It contains both CA and UTA control information. Only the DEFAULTS and COBOL statements are processed by CA. (PL/I statements would also be processed by CA, if any were present.)

4. Verify that the listing file names and the copy to/from object module names are correct.

5. Press the End key (PF3) to terminate the edit session.

For more detailed information, see “Editing the Coverage Assistant Control File” on page 81.

```
*
* Cobol Example
*
* Statements required for coverage and unit test
*
               Defaults ListDsn=ATC.V1R5M0.SAMPLE.COBOLST(*),
                        LoadMod=COB01M,
                        FromObjDsn=ATC.V1R5M0.SAMPLE.OBJ,
                        ToObjDsn=ATC.V1R5M0.SAMPLE.ZAPOBJ

COB01AM:       COBOL ListMember=COB01AM
COB01CM:       COBOL ListMember=COB01CM
COB01DM:       COBOL ListMember=COB01DM

*
* Statements required for unit test
*

COB01AM_S:     Scope COBOL=COB01AM,ExtProgram-Id=COB01AM
COB01AM_S_B:   Scope COBOL=COB01AM,ExtProgram-Id=COB01AM,
                           NestedProgram-Id=COB01BM
COB01CM_S:     Scope COBOL=COB01CM,ExtProgram-Id=COB01CM

TAPARM1:       Variable Scope=COB01AM_S,Name=(TAPARM1)
CITY:          Variable Scope=COB01AM_S,Name=(CITY In LOC-ID In TASTRUCT)
STATE:         Variable Scope=COB01AM_S,Name=(STATE In LOC-ID In TASTRUCT)
TBPARM2:       Variable Scope=COB01AM_S_B,Name=(TBPARM2)
TCPARM1:       Variable Scope=COB01CM_S,Name=(TCPARM1)

               Coverage Variable=TAPARM1,Length=2,NAME // read TAPARM1
*                                      wherever it occurs (by NAME)

               Coverage Variable=CITY,Length=3,NAME   // read CITY of
*                                                        TASTRUCT

               Coverage Variable=STATE,Length=2,FULL  // read STATE of
*                               STRUCT whether directly referenced or
*                               via structure (FULL)

               Coverage Variable=TBPARM2,Length=2,MaxSave=1,Stmts=(113)
*                               in COB01BM read TBPARM2 on line 113 only,
*                               only once

               Coverage Variable=TCPARM1,Length=2,MaxSave=1,NAME // in
*                                      COB01CM read TCPARM1 only once
```

Figure 9. Control File for COB01M

Create Setup JCL

Before test cases can be executed on the COB01M test program, CA must insert breakpoints into the test program. CA does this using the Setup JCL.

When you execute the Setup JCL, the CA Setup program analyzes the assembler statements in the compiler listings and creates a table containing breakpoint data (address, op code, and so on). User SVC instructions are inserted for the instructions at the breakpoints in the instrumented object modules or load modules. If you instrumented object modules, you then link these modified object modules into a modified COB01M load module for CA to use.

To create the Setup JCL:

1. Select option 1 from the ATC Primary Option Menu.

   The Coverage, Distillation, and Unit Test Assistant panel is displayed.

2. Select option 2.

   The Create JCL for Setup panel is displayed. You create the JCL for the setup of COB01M from this panel.

   All of the default values on this panel are correct for the COB01M example. The defaults will usually be correct for your test coverage run. The only field that you may need to change is the Program Name field.

3. If necessary, change the program name to COB01M.

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 identified on the panel using member name SCOB01M.

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

For more detailed information, see “CA, DA, and UTA Setup” on page 231.

Create JCL to Start a Monitor Session

JCL is required to start a CA monitor session.

To create the JCL to start a monitor session:

1. Select option 1 from the ATC Primary Option Menu.

   The Coverage, Distillation and Unit Test Assistant panel is displayed.

2. Select option 3.

   The Create JCL to Start the Monitor panel is displayed. You create the JCL to start a CA monitor session for COB01M from this panel.

3. If necessary, change the program name to COB01M.

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 identified on the panel using member name XCOB01M.

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

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

For more detailed information, see “Monitor Execution” on page 245.

Create JCL for a Summary Report

JCL is required to generate a summary report.

To create the summary report JCL:

1. Select option `1` from the `ATC Primary Option Menu`.

   The `Coverage, Distillation, and Unit Test Assistant` panel is displayed.
2. Select option `4`.

   The `Coverage Reports` panel is displayed.
3. Select option `1`.

   The `Create JCL for Summary Report` panel is displayed. You create the JCL for generating the COB01M summary report from this panel.
4. If necessary, change `Program Name` to `COB01M`.
5. Select option `1`.

   Informational messages are written to your screen as the JCL is created. The created JCL is put into the JCL library identified on the panel using member name TCOB01M.
6. Press the End key (PF3) to exit the panel.

For more detailed information, see “Summary Coverage Report” on page 83.

Create JCL to Link the Modified Object Modules

If you instrumented object modules, you must link the modified object modules (modified by the Setup step) into an executable program for testing. You can use the normal JCL that links your program, but be sure to specify the object module library that contains the modified object modules. Sample JCL to link the COB01M example is provided in *hi_lev_qual*.V1R5M0.SAMPLE.JCL(LCOB01M).

Create JCL to Run the GO Step

You can use the normal JCL that executes your program, but be sure to specify the load module library that contains the link-edited modified object modules. Sample JCL to execute the GO step for the COB01M example is provided in *hi_lev_qual*.V1R5M0.SAMPLE.JCL(GCOB01M).

Execute the JCL

When you have created all of the COB01M JCL, you can run the COB01M summary example by executing the following functions in the order listed. To see a flow diagram of these steps, go to Figure 7 on page 46.

1. SCOB01M[6]

   Performs the Setup step. All JCL steps should complete with condition code 0.

2. LCOB01M[7]

   If you instrumented object modules, links the object modules that were modified with breakpoints in the Setup step into the COB01M load module.

3. XCOB01M[6]

   Starts the monitor. JCL completes with condition code 0.

4. GCOB01M[7]

   Runs sample program COB01M. COB01M runs to completion with condition code 0.

5. CASTATS[8]

   Displays statistics with the CASTATS command. You should see a nonzero EVNTS count in the TOTALS line. (This is an optional step for illustrative purposes.)

6. CASTOP[8]

   Stops the monitor session. CA writes the statistics to disk.

7. TCOB01M[6]

   Creates the summary of COB01M. The summary is in data set proj_qual.COB01M.SUMMARY.

---

6 JCL created from the panels and put into the JCL library.

7 JCL supplied with the installation materials in *hi_lev_qual*.V1R5M0.SAMPLE.JCL. (Sample JCL for all of the steps can be found in this partitioned data set [PDS].)

8 Monitor commands issued from either the `Control the CA/DA/UTA Monitor` panel or the TSO command processor (ISPF option 6) by entering:

```
EX 'hi_lev_qual.V1R5M0.REXX(cacmd)'
```

where `cacmd` is the command issued (such as, CASTATS, CASTOP, and so on).

Annotated COBOL Listings

To produce an annotated COBOL listing of COB01M, perform the following steps. **Steps 2a through 6 are described in more detail in topics that follow in this chapter.**

1. Compile the COBOL source you want to test. This produces listings that include the assembler statements needed by CA. (This has already been done for the COB01M example. The listings are in *hi_lev_qual*.V1R5M0.SAMPLE.COBOLST.)

   Make sure to use the compiler options specified in “Setup” on page 231.

2. Start the ATC ISPF dialog by entering the following from ISPF option 6:

   ```
   EX 'hi_lev_qual.V1R5M0.REXX(ATSTART)'
   ```

   The `ATC Primary Option Menu` is displayed (shown in Figure 2 on page 27).

   a. Edit the CA control file.

      Verify that the control file includes the listings of the object modules 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 an annotated listing.

      Create the JCL to produce the annotated listing after COB01M has executed.

3. End the ATC ISPF dialog by pressing the End key (PF3) on the `ATC Primary Option Menu`.

4. Create the JCL to link the modified object modules.

   After the Setup step, and before starting the monitor, you must link the modified object modules into an executable program 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. Create the JCL to run the GO step.

   Create the JCL to run your program. Specify the same modified load module as in step 4.

6. Execute the JCL.

   Execute the created JCL files for COB01M in the correct order. (This order is shown in “Execute the JCL” on page 58.)

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

& 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.

: Non-branch instruction has executed.

¬ Instruction has not executed.

Figure 10 shows a sample annotated listing.

```
000001               IDENTIFICATION DIVISION.
000002               PROGRAM-ID. COB01AM.
000003              ****************************************************************
000004              *                                                              *
000005              * LICENSED MATERIALS - PROPERTY OF IBM                         *
000006              *                                                              *
000007              * 5799-GBN                                                     *
000008              *                                                              *
000009              * (C) COPYRIGHT IBM CORP. 1997, 1998 ALL RIGHTS RESERVED       *
000010              *                                                              *
000011              * US GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION OR  *
000012              * DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE CONTRACT WITH IBM  *
000013              * CORP.                                                        *
000014              *                                                              *
000015              *                                                              *
000016              ****************************************************************
000017              ****************************************************************
000018              *                                                              *
000019              * COBOL FOR MVS & VM TEST.                                     *
000020              *                                                              *
000021              * MEMBER COB01AM HAS ENTRY POINT COB01AM.                      *
000022              * CALLS COB01BM, WHICH CALLS COB01CM, WHICH CALLS COB01DM.     *
000023              ****************************************************************
000024
000025               ENVIRONMENT DIVISION.
000026
000027               DATA DIVISION.
000028
000029               WORKING-STORAGE SECTION.
000030               01 TAPARM1      PIC 99 VALUE 5.
000031               01 TAPARM2      PIC 99 VALUE 2.
000032               01 COB01BM      PIC X(7) VALUE 'COB01BM'.
000033               01 P1PARM1      PIC 99 VALUE 0.
000034
000035               01 TASTRUCT.
000036                 05 LOC-ID.
000037                    10 STATE    PIC X(2).
000038                    10 CITY     PIC X(3).
000039                 05 OP-SYS     PIC X(3).
000040
```

Figure 10 (Part 1 of 3). Annotated COBOL Listing

```
000041                PROCEDURE DIVISION.
000042
000043               * THE FOLLOWING ALWAYS PERFORMED
000044
000045               * ACCESS BY TOP LEVEL QUALIFIER
000046 :                  MOVE 'ILCHIMVS' TO TASTRUCT.
000047
000048               * ACCESS BY MID LEVEL QUALIFIERS
000049 :                  MOVE 'ILSPR' TO LOC-ID.
000050 :                  MOVE 'AIX' TO OP-SYS.
000051
000052               * ACCESS BY LOW LEVEL QUALIFIERS
000053 :                  MOVE 'KY' TO STATE.
000054 :                  MOVE 'LEX' TO CITY.
000055 :                  MOVE 'VM ' TO OP-SYS.
000056
000057                PROGA.
000058
000059               * THIS PERFORM EXECUTED
000060 &                  PERFORM WITH TEST BEFORE UNTIL TAPARM1 = 0
000061 :    1               SUBTRACT 1 FROM TAPARM1
000062 :    1               CALL 'COB01BM'
000063                   END-PERFORM
000064
000065               * THIS IF ALWAYS FALSE
000066 >                  IF TAPARM2 = 0
000067 ¬    1               PERFORM PROCA
000068                   END-IF
000069
000070               * THIS PERFORM EXECUTED
000071 &                  PERFORM WITH TEST BEFORE UNTIL TAPARM2 = 0
000072 :    1               SUBTRACT 1 FROM TAPARM2
000073                   END-PERFORM
000074 :                  STOP RUN
000075                   .
000076
000077                PROCA.
000078               * PROCA NEVER CALLED
000079 ¬                  MOVE 10 TO P1PARM1
000080                   .
000081
000082               * START OF COB01BM NESTED IN COB01AM
000083
000084   1            IDENTIFICATION DIVISION.
000085   1            PROGRAM-ID. COB01BM.
000086   1           ****************************************************************
000087   1           *                                                              *
000088   1           * COBOL FOR MVS & VM TEST.                                     *
000089   1           *                                                              *
000090   1           * COB01BM, CALLED BY COB01AM.                                  *
000091   1           ****************************************************************
000092   1
000093   1            ENVIRONMENT DIVISION.
000094   1
000095   1            DATA DIVISION.
000096   1
000097   1            WORKING-STORAGE SECTION.
000098   1            01 TBPARM1      PIC 99 VALUE 5.
000099   1            01 TBPARM2      PIC 99 VALUE 0.
000100   1            01 COB01CM      PIC X(7) VALUE 'COB01CM'.
000101   1            01 P1PARM1      PIC 99 VALUE 0.
000102   1
```

Figure 10 (Part 2 of 3). Annotated COBOL Listing

```
000103   1            PROCEDURE DIVISION.
000104   1
000105   1            PROGB.
000106   1           * THIS PERFORM EXECUTED
000107 & 1                PERFORM WITH TEST BEFORE UNTIL TBPARM1 = 0
000108 : 1  1               SUBTRACT 1 FROM TBPARM1
000109 : 1  1               CALL 'COB01CM'
000110   1                END-PERFORM
000111   1
000112   1           * THIS IF EXECUTED
000113 V 1                IF TBPARM2 = 0
000114 : 1  1               PERFORM PROCB
000115   1                END-IF
000116   1
000117   1           * THIS PERFORM NOT EXECUTED
000118 ¬ 1                PERFORM WITH TEST BEFORE UNTIL TBPARM2 = 0
000119 ¬ 1  1               SUBTRACT 1 FROM TBPARM2
000120   1                END-PERFORM
000121   1                .
000122   1
000123   1            PROCB.
000124   1           * PROCB EXECUTED
000125 : 1                MOVE 10 TO P1PARM1
000126   1                .
000127   1
000128 : 1                EXIT PROGRAM.
000129   1
000130   1            END PROGRAM COB01BM.
000131                END PROGRAM COB01AM.
```

Figure 10 (Part 3 of 3). Annotated COBOL Listing

Edit the CA Control File

This step is identical to the corresponding procedure for summary reports at “Edit the CA Control File” on page 49.

Create Setup JCL

This step is identical to the corresponding procedure for summary reports at “Create Setup JCL” on page 50.

Create JCL To Start a Monitor Session

This step is identical to the corresponding procedure for summary reports at “Create JCL to Start a Monitor Session” on page 51.

Create JCL for an Annotated Listing

To create the annotated listing JCL:

1. Select option `1` from the `ATC Primary Option Menu`.

   The `Coverage, Distillation, and Unit Test Assistant` panel is displayed.

2. Select option `4`.

   The `Coverage Reports` panel is displayed.

3. Select option `2`.

   The `Create JCL for Summary and Annotation Report` panel is displayed. You create the JCL for printing the annotated listing (along with a summary) for COB01M from this panel.

4. If necessary, change the program name to `COB01M`.
5. Select option `1`.

   Informational messages are written to your screen as the JCL is created. The created JCL is put into the JCL library identified on the panel using member name RCOB01M.
6. Press the End key (PF3) to exit the panel.

For more detailed information, see “Annotated Listing Coverage Report” on page 95.

Create JCL to Link the Modified Object Modules

This step is identical to the corresponding procedure for summary reports at “Create JCL to Link the Modified Object Modules” on page 52.

Create JCL to Run the GO Step

This step is identical to the corresponding procedure for summary reports at “Create JCL to Run the GO Step” on page 52.

Execute the JCL

When you have created all of the COB01M JCL, you can run the COB01M example by executing the following JCL in the order listed. See Figure 7 on page 46 for a flow diagram of these steps.

1. SCOB01M[9]

   Performs the Setup step. All JCL steps should complete with condition code 0.
2. LCOB01M[10]

   If you instrumented object modules, links the object modules that were modified with breakpoints during the Setup step into the COB01M load module.
3. XCOB01M[9]

   Starts the monitor. JCL completes with condition code 0.
4. GCOB01M[10]

   Runs sample program COB01M. COB01M runs to completion with condition code 0.
5. CASTATS[11]

   Displays statistics with the CASTATS command. You should see a nonzero EVNTS count in the TOTALS line. (This is an optional step for illustrative purposes.)

---

9 JCL created from the panels and put into the JCL library.

10 JCL supplied with the installation materials in *hi_lev_qual*.V1R5M0.SAMPLE.JCL. (Sample JCL for all of the steps can be found in this partitioned data set [PDS].)

11 Monitor commands issued from either the `Control the CA/DA/UTA Monitor` panel or the TSO command processor (ISPF option 6) by entering:

```
EX 'hi_lev_qual.V1R5M0.REXX(cacmd)'
```

where `cacmd` is the command issued (such as, CASTATS, CASTOP, and so on).

6. CASTOP[11]

   Stops the monitor session. CA writes the statistics to disk.

7. RCOB01M[9]

   Creates the annotated listing (and summary) of COB01M. The report is in data set proj_qual.COB01M.REPORT.

summary of test case summary of test case

PL/I Summary of Test Case Coverage

The content of the summary report is the same regardless of the compiler for which it is created. For PL/I programs, the breakpoints are inserted into the object modules during Setup. When you are ready to test your program, you link the object modules that have been modified with breakpoints.

Figure 11 on page 60 is a summary of a PL/I program called PLI01M. To produce a summary for PLI01M, perform the following steps. **Steps 2a through 6 are described in more detail in topics that follow in this chapter.**

1. Compile the PL/I source you want to test. This produces listings that include the assembler statements needed by CA. (This has already been done for the PLI01M example. The listings are in *hi_lev_qual*.V1R5M0.SAMPLE.PLILST.)

   Make sure to use the compiler options specified in "Setup" on page 231.

2. Start the ATC ISPF dialog by entering the following from ISPF option 6:

   ```
   EX 'hi_lev_qual.V1R5M0.REXX(ATSTART)'
   ```

   The `ATC Primary Option Menu` is displayed (shown in Figure 2 on page 27).

   a. Edit the CA control file.

      Verify that the control file includes the listings of the object modules or load modules you want to test.

   b. Create the setup JCL.

      Create the JCL that enables the Setup job to produce a file containing breakpoint data and to modify copies of your object modules by inserting breakpoints.

   c. Create the monitor JCL.

      Create the JCL to start a monitor session.

   d. Create the summary report JCL.

      Create the JCL to produce the test case coverage summary after PLI01M has executed.

3. End the ATC ISPF dialog by pressing the End key (PF3) on the `ATC Primary Option Menu`.

4. Create the JCL to link the modified object modules.

   If you instrumented object modules, you must link the modified object modules into an executable program 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. Create the JCL to run the GO step.

   Create the JCL to run your program. Specify the same modified load module as in step 4 on page 59.

6. Execute the JCL.

   Execute the created JCL files for PLI01M in the correct order. (This order is shown in “Execute the JCL” on page 64.)

Figure 11 shows a sample summary report. For details about the information included in the report, see “Summary Coverage Report” on page 83.

```
1 *********  CA SUMMARY:                    PROGRAM AREA DATA                    ********
0               DATE: 02/15/1999
                TIME: 12:28.34
         TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                           |                                        |  STATEMENTS:          | BRANCHES:          |
  | PA LOAD MOD PROCEDURE     | LISTING NAME                           |   TOTAL     EXEC   %  |  CPATH   TAKEN   %  |
  ------------------------------------------------------------------------------------------------------------------------
     1 PLI01M   PLI01AM         ATC.V1R5M0.SAMPLE.PLILST(PLI01AM)              9        9 100.0       6       5  83.3
     2          PROC2A                                                         2        0   0.0       0       0 100.0
     3 PLI01M   PLI01BM         ATC.V1R5M0.SAMPLE.PLILST(PLI01BM)             11        8  72.7       6       4  66.7
     4          PROC1                                                          2        2 100.0       0       0 100.0
     5 PLI01M   PLI01CM         ATC.V1R5M0.SAMPLE.PLILST(PLI01CM)              7        4  57.1       6       3  50.0
     6          PROC1                                                          4        3  75.0       2       1  50.0
     7 PLI01M   PLI01DM         ATC.V1R5M0.SAMPLE.PLILST(PLI01DM)              2        0   0.0       2       0   0.0
     8          PROC1                                                          4        0   0.0       2       0   0.0
  ------------------------------------------------------------------------------------------------------------------------
  Summary for all PAs:                                                         41       26  63.4      24      13  54.2

1 *********  CA SUMMARY:                    UNEXECUTED CODE                      ********
0               DATE: 02/15/1999
                TIME: 12:28.34
         TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                           |                                        |
  | PA LOAD MOD PROCEDURE     | LISTING NAME                           | start    end     start    end     start   end
  ------------------------------------------------------------------------------------------------------------------------
     2 PLI01M   PROC2A          ATC.V1R5M0.SAMPLE.PLILST(PLI01AM)            17     18
     3 PLI01M   PLI01BM         ATC.V1R5M0.SAMPLE.PLILST(PLI01BM)             9      9        15     16
     5 PLI01M   PLI01CM         ATC.V1R5M0.SAMPLE.PLILST(PLI01CM)             6      6        10     11
     6          PROC1                                                        16     16
     7 PLI01M   PLI01DM         ATC.V1R5M0.SAMPLE.PLILST(PLI01DM)             3     11
     8          PROC1                                                         6     10
  ------------------------------------------------------------------------------------------------------------------------

1 *********  CA SUMMARY:                    BRANCHES THAT HAVE NOT GONE BOTH WAYS ********
0               DATE: 02/15/1999
                TIME: 12:28.34
         TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                           |                                        |
  | PA LOAD MOD PROCEDURE     | LISTING NAME                           |  stmt      stmt      stmt      stmt      stmt
  ------------------------------------------------------------------------------------------------------------------------
     1 PLI01M   PLI01AM         ATC.V1R5M0.SAMPLE.PLILST(PLI01AM)            10
     3 PLI01M   PLI01BM         ATC.V1R5M0.SAMPLE.PLILST(PLI01BM)             8        14
     5 PLI01M   PLI01CM         ATC.V1R5M0.SAMPLE.PLILST(PLI01CM)             5         8         9
     6          PROC1                                                        15
     7 PLI01M   PLI01DM         ATC.V1R5M0.SAMPLE.PLILST(PLI01DM)             3
     8          PROC1                                                         7
  ------------------------------------------------------------------------------------------------------------------------
```

Figure 11. Summary Report for PLI01M

Edit the CA Control File

CA uses assembler listings to determine where to insert breakpoints. You supply the names of the listing files in the CA control file (CACTL).

Make sure to use the compiler options specified in “Setup” on page 231.

The CACTL control file for the PL/I summary example is *hi_lev_qual*.V1R5M0.SAMPLE.CACTL(PLI01M). The file is shown in Figure 12 on page 62.

To edit the CACTL for the PLI01M summary example:

1. Select option `1` from the `ATC Primary Option Menu`.

   The `Coverage, Distillation and Unit Test Assistant` panel is displayed.

2. Select option `1`.

   The `Work with the CA/DA/UTA Control File` panel is displayed.

3. Specify the following:

Use Program Name for File Name	`YES`
Program Name	`PLI01M`
Listing Type	`PL/I`

   An ISPF edit session for the CA control file you requested is displayed.

   The data in the control file consists of the following:

   - The type of listing file (PL/I)
   - Names of the listing files you want to test
   - Names of the load modules that contain the code of each listing
   - Copy to/from information for making copies of the object modules or load modules into which the breakpoints are inserted

   If the control file you requested did not previously exist, it is created with comments in it to help you enter the appropriate information in the fields. The control file shown in Figure 12 on page 62 is the control file for PLI01M. This example shows how to instrument object modules. To instrument load modules, see “Instrumentation of Load Modules instead of Object Modules” on page 238.

4. Verify that the listing file names and the copy to/from object module names are correct.

5. Press the End key (PF3) to terminate the edit session.

For more information, see “Editing the Coverage Assistant Control File” on page 81.

```
Defaults ListDsn=ATC.V1R5M0.SAMPLE.PLILST(*),
         LoadMod=PLI01M,
         FromObjDsn=ATC.V1R5M0.SAMPLE.OBJ,
         ToObjDsn=ATC.V1R5M0.SAMPLE.ZAPOBJ

PL/I     ListMember=PLI01AM

PL/I     ListMember=PLI01BM

PL/I     ListMember=PLI01CM

PL/I     ListMember=PLI01DM
```

Figure 12. Control File for PLI01M

Create Setup JCL

Before test cases can be executed on the test program (PLI01M), CA must insert breakpoints into the test program. During Setup, the CA Setup program analyzes the assembler statements from the compiler listings and creates a table containing breakpoint data (address, op code, and so on). User SVC instructions are inserted for the instructions at the breakpoints in the object modules or load modules. If you instrument object modules, you then link these modified object modules into a modified PLI01M load module for CA to use.

To create the setup JCL:

1. Select option `1` from the `ATC Primary Option Menu`.

   The `Coverage, Distillation, and Unit Test Assistant` panel is displayed.

2. Select option `2`.

   The `Create JCL for Setup` panel is displayed. You create the JCL for the setup of PLI01M from this panel.

   All of the default values on this panel are correct for the PLI01M example. The defaults will usually be correct for your test coverage run. The only field that you may need to change is the `Program Name` field.

3. If necessary, change the program name to `PLI01M`.

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 identified on the panel using member name SPLI01M.

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

For more detailed information, see “CA, DA, and UTA Setup” on page 231.

Create JCL to Start a Monitor Session

JCL is required to start a monitor session.

To create the JCL to start a monitor session:

1. Select option `1` from the `ATC Primary Option Menu` and option `3` from the `Coverage, Distillation, and Unit Test Assistant` panel.

   The `Create JCL to Start the Monitor` panel is displayed. You create the JCL for the CA execution of PLI01M from this panel.

2. If necessary, change the program name to `PLI01M`.
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 identified on the panel using member name XPLI01M.
4. Press the End key (PF3) to exit the panel.

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

For more detailed information, see "Monitor Execution" on page 245.

Create JCL for a Summary Report

JCL is required to generate a summary report.

To create the summary report JCL:

1. Select option `1` from the `ATC Primary Option Menu`.

   The `Coverage, Distillation and Unit Test Assistant` panel is displayed.
2. Select option `4`.

   The `Coverage Reports` panel is displayed.
3. Select option `1`.

   The `Create JCL for Summary Report` panel is displayed. You create the JCL for generating the PLI01M summary report from this panel.
4. If necessary, change the program name to `PLI01M`.
5. Select option `1`.

   Informational messages are written to your screen as the JCL is created. The created JCL is put into the JCL library identified on the panel using member name TPLI01M.
6. Press the End key (PF3) to exit the panel.

For more detailed information, see "Summary Coverage Report" on page 83.

Create JCL to Link the Modified Object Modules

If you instrumented object modules, you must link the modified object modules (modified by the Setup step) into an executable program for testing. You can use the normal JCL that links your program, but be sure to specify the object module library that contains the modified object modules. Sample JCL to link the PLI01M example is provided in *hi_lev_qual*.V1R5M0.SAMPLE.JCL(LPLI01M).

Create JCL to Run the GO Step

You can use the normal JCL that executes your program, but be sure to specify the load module library that contains the link-edited modified object modules. Sample JCL to execute the GO step for the PLI01M example is provided in *hi_lev_qual*.V1R5M0.SAMPLE.JCL(GPLI01M).

Execute the JCL

When you have created all of the PLI01M JCL, you can run the PLI01M summary example by executing the following functions in the order listed. See Figure 7 on page 46 for a flow diagram of these steps.

1. SPLI01M[12]

   Performs the Setup step. All JCL steps should end with condition code 0.

2. LPLI01M[13]

   If you instrumented object modules, links the object modules that were modified with breakpoints in the Setup step into the PLI01M load module.

3. XPLI01M[12]

   Starts the monitor. JCL completes with condition code 0.

4. GPLI01M[13]

   Runs sample program PLI01M. PLI01M runs to completion with condition code 0.

5. CASTATS[14]

   Displays statistics with the CASTATS command. You should see a nonzero EVNTS count in the TOTALS line. (This is an optional step for illustrative purposes.)

6. CASTOP[14]

   Stops the monitor session. CA writes the statistics to disk.

7. TPLI01M[12]

   Creates the summary of PLI01M. The summary is in data set proj_qual.PLI01M.SUMMARY.

Annotated PL/I Listings

To produce an annotated PL/I listing of PLI01M, perform the following steps. **Steps 2a through 6 are described in more detail in topics that follow in this chapter.**

1. Compile the PL/I source you want to test. This produces listings that include the assembler statements needed by CA. (This has already been done for the PLI01M example. The listings are in *hi_lev_qual*.V1R5M0.SAMPLE.PLILST.)

   Make sure to use the compiler options specified in “Setup” on page 231.

2. Start the ATC ISPF dialog by entering the following from ISPF option 6:

   ```
   EX 'hi_lev_qual.V1R5M0.REXX(ATSTART)'
   ```

[12] JCL created from the panels and put into the JCL library.

[13] JCL supplied with the installation materials in *hi_lev_qual*.V1R5M0.SAMPLE.JCL. (Sample JCL for all of the steps can be found in this partitioned data set [PDS].)

[14] Monitor commands issued from either the `Control the CA/DA/UTA Monitor` panel or the TSO command processor (ISPF option 6) by entering:

```
EX 'hi_lev_qual.V1R5M0.REXX(cacmd)'
```

where `cacmd` is the command issued (such as, CASTATS, CASTOP, and so on).

The `ATC Primary Option Menu` is displayed (shown in Figure 2 on page 27).

a. Edit the CA control file.

   Verify that the control file includes the listings of the object modules you want to test.

b. Create the setup JCL.

   Create the JCL that enables the Setup job to produce a file containing breakpoint data and to modify copies of your object modules by inserting breakpoints.

c. Create the JCL to start a monitor session.

d. Create the JCL for an annotated listing.

   Create the JCL to produce the annotated listing after PLI01M has executed.

3. End the ATC ISPF dialog by pressing the End key (PF3) on the `ATC Primary Option Menu`.
4. Create the JCL to link the modified object modules.

   If you instrumented object modules, you must link the modified object modules into an executable program 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. Create the JCL to run the GO step.

   Create the JCL to run your program. Specify the same modified load module as in step 4.

6. Execute the JCL.

   Execute the created JCL files for PLI01M in the correct order. (This order is shown in “Execute the JCL” on page 64.)

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

& 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.

: Non-branch instruction has executed.

¬ Instruction has not executed.

Figure 13 on page 66 shows a sample annotated listing.

```
 1      0  PLI01AM:PROC OPTIONS(MAIN);         /* PL/I FOR MVS & VM TEST
           /*********************************************************************/
           /*                                                                   */
           /* Licensed Materials - Property of IBM                              */
           /*                                                                   */
           /* 5799-GBN                                                          */
           /*                                                                   */
           /* (C) Copyright IBM Corp. 1997, 1998 All Rights Reserved            */
           /*                                                                   */
           /* US Government Users Restricted Rights - Use, duplication or       */
           /* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*/
           /*                                                                   */
           /*********************************************************************/
 2   1  0  DCL EXPARM1 FIXED BIN(31) INIT(5);
 3   1  0  DCL EXPARM2 FIXED BIN(31) INIT(2);
 4   1  0  DCL PARM2   FIXED BIN(31) INIT(2);
 5   1  0  DCL PLI01BM EXTERNAL ENTRY;        /*                               */
 6&  1  0  DO WHILE (EXPARM1 > 0);            /*  THIS DO LOOP EXECUTED 5 TIMES*/
 7:  1  1     EXPARM1 = EXPARM1 -1;           /*
 8:  1  1     CALL  PLI01BM(PARM2);           /* PLI01BM CALLED 5 TIMES
 9:  1  1  END;
10>  1  0  IF (EXPARM2 = 0) THEN             /* THIS BRANCH ALWAYS TAKEN      */
              CALL PROC2A(EXPARM2);          /* PROC2A NEVER  CALLED          */
11&  1  0  DO WHILE (EXPARM2 > 0);           /* DO LOOP EXECUTED TWICE        */
12:  1  1    EXPARM2 = EXPARM2 - 1;
13:  1  1  END;
14:  1  0  RETURN;

15   1  0  PROC2A: PROCEDURE(P1PARM1);         /* THIS PROCEDURE NEVER EXECU
16   2  0  DCL P1PARM1 FIXED BIN(31);
17¬  2  0  P1PARM1 = 10;
18¬  2  0  END PROC2A;
19   1  0  END PLI01AM;
```

Figure 13. Annotated PL/I Listing

Edit the CA Control File

This step is identical to the corresponding procedure for summary reports at “Edit the CA Control File” on page 61.

Create Setup JCL

This step is identical to the corresponding procedure for summary reports at “Create Setup JCL” on page 62.

Create JCL to Start a Monitor Session

This step is identical to the corresponding procedure for summary reports at “Create JCL to Start a Monitor Session” on page 62.

Create JCL for an Annotated Listing

To create the annotated listing JCL:

1. Select option `1` from the `ATC Primary Option Menu`.

   The `Coverage, Distillation and Unit Test Assistant` panel is displayed.

2. Select option `4`.

   The `Coverage Reports` panel is displayed.

3. Select option 2.

   The Create JCL for Summary and Annotation Report panel is displayed. You create the JCL for printing the annotated listing (along with a summary) for PLI01M from this panel.

4. If necessary, change the program name to PLI01M.

5. Select option 1.

   Informational messages are written to your screen as the JCL is created. The created JCL is put into the JCL library identified on the panel using member name RPLI01M.

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

For more detailed information, see "Annotated Listing Coverage Report" on page 95.

Create JCL to Link the Modified Object Modules

This step is identical to the corresponding procedure for summary reports at "Create JCL to Link the Modified Object Modules" on page 63.

Create JCL to Run the GO Step

This step is identical to the corresponding procedure for summary reports at "Create JCL to Run the GO Step" on page 63.

Execute the JCL

When you have created all of the PLI01M JCL, you can run the PLI01M example by executing the following JCL in the order listed. See Figure 7 on page 46 for a flow diagram of these steps.

1. SPLI01M[15]

   Performs the Setup step. All JCL steps should complete with condition code 0.

2. LPLI01M[16]

   If you instrumented object modules, links the object modules that were modified with breakpoints during the Setup step into the PLI01M load module.

3. XPLI01M[15]

   Starts the monitor. JCL completes with condition code 0.

4. GPLI01M[16]

   Runs sample program PLI01M. PLI01M runs to completion with condition code 0.

---

15 JCL created from the panels and put into the JCL library.

16 JCL supplied with the installation materials in *hi_lev_qual*.V1R5M0.SAMPLE.JCL. (Sample JCL for all of the steps can be found in this partitioned data set [PDS].)

5. CASTATS[17]

   Displays statistics with the CASTATS command. You should see a nonzero EVNTS count in the TOTALS line. (This is an optional step for illustrative purposes.)

6. CASTOP[17]

   Stops the monitor session. CA writes the statistics to disk.

7. RPLI01M[15]

   Creates the annotated listing (and summary) of PLI01M. The report is in data set proj_qual.PLI01M.REPORT.

Assembler Summary of Test Case Coverage

The content of the summary report is the same for the two assemblers supported: Assembler H and High Level Assembler. The breakpoints are inserted into the object modules during Setup. When you are ready to test your program, you link the object modules that have been modified with breakpoints.

Figure 14 on page 70 is a summary of a sample assembler program called ASM01L, a High Level Assembler program. The steps for Assembler H are identical. The Assembler H sample program is called ASM01H. For Assembler H, substitute ASM01H for ASM01L in the following steps. To produce a summary for ASM01L, perform the following steps as listed. **Steps 2a through 6 are described in more detail in topics that follow in this chapter.**

1. Assemble the assembler source you want to test. This produces the listings that include the assembler statements needed by CA. (This has already been done for the ASM01L example. The listings are in *hi_lev_qual*.V1R5M0.SAMPLE.ASMLLST for the High Level Assembler and *hi_lev_qual*.V1R5M0.SAMPLE.ASMHLST for the H Assembler.)

   Make sure to use the assembler options specified in “Setup” on page 231.

2. Start the ATC ISPF dialog by entering the following from ISPF option 6:

   ```
   EX 'hi_lev_qual.V1R5M0.REXX(ATSTART)'
   ```

   The `ATC Primary Option Menu` is displayed (shown in Figure 2 on page 27).

   a. Edit the CA control file.

      Verify that the control file includes the listings of the object modules or load modules you want to test. The samples shipped with the product show how to instrument object modules. To instrument load modules, see “Instrumentation of Load Modules instead of Object Modules” on page 238.

   b. Create the setup JCL.

      Create the JCL that enables the Setup job to produce a file containing breakpoint data and instrumented object modules or load modules by inserting breakpoints.

---

[17] Monitor commands issued from either the `Control the CA/DA/UTA Monitor` panel or the TSO command processor (ISPF option 6) by entering:

```
EX 'hi_lev_qual.V1R5M0.REXX(cacmd)'
```

where `cacmd` is the command issued (such as, CASTATS, CASTOP, and so on).

c. Create the monitor JCL.

   Create the JCL to start a monitor session.

d. Create the summary report JCL.

   Create the JCL to produce the test case coverage summary after ASM01L has executed.

3. End the ATC ISPF dialog by pressing the End key (PF3) on the `ATC Primary Option Menu`.
4. Create the JCL to link the modified object modules.

   If you instrumented object modules, you must link the modified object modules into an executable program 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. Create the JCL to run the GO step.

   Create the JCL to run your program. Specify the same modified load module as in step 4.

6. Execute the JCL.

   Execute the created JCL files for ASM01L in the correct order. (This order is shown in “Execute the JCL” on page 74.)

Figure 14 on page 70 shows a sample summary report. For details about the information included in the report, see “Summary Report for Assembler” on page 88.

```
1 *********  CA SUMMARY:                    PROGRAM AREA DATA                    ********
0               DATE: 04/22/1998
                TIME: 15:09.53
        TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                           |                                         |  STATEMENTS:          | BRANCHES:           |
  | PA LOAD MOD PROCEDURE     | LISTING NAME                            |   TOTAL     EXEC   %  |  CPATH   TAKEN   %  |
  ---------------------------------------------------------------------------------------------------------------------
     1 ASM01L   TEST2           ATC.V1R5M0.SAMPLE.ASMLLST(ASM01AL)      *    144      104  72.2       6       5  83.3
     2 ASM01L   TEST2B          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01BL)      *    164      144  87.8       6       4  66.7
     3 ASM01L   TEST2C          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01CL)      *    140      118  84.3       8       5  62.5
     4 ASM01L   TEST2D          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01DL)      *     92        0   0.0       4       0   0.0
  ---------------------------------------------------------------------------------------------------------------------
  Summary for all PAs:                                                       540      366  67.8      24      14  58.3

1 *********  CA SUMMARY:                    UNEXECUTED CODE                      ********
0               DATE: 04/22/1998
                TIME: 15:09.53
        TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                           |                                         |
  | PA LOAD MOD PROCEDURE     | LISTING NAME                            | start    end     start    end     start   end
  ---------------------------------------------------------------------------------------------------------------------
     1 ASM01L   TEST2           ATC.V1R5M0.SAMPLE.ASMLLST(ASM01AL)      000056 000062    000086 0000A0
     2 ASM01L   TEST2B          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01BL)      000050 00005A    000078 000082
     3 ASM01L   TEST2C          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01CL)      000034 00003A    000050 00005A    000092 000098
     4 ASM01L   TEST2D          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01DL)      000000 000000    000016 00006C
  ---------------------------------------------------------------------------------------------------------------------

1 *********  CA SUMMARY:                    BRANCHES THAT HAVE NOT GONE BOTH WAYS ********
0               DATE: 04/22/1998
                TIME: 15:09.53
        TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                           |                                         |
  | PA LOAD MOD PROCEDURE     | LISTING NAME                            |  stmt      stmt      stmt      stmt      stmt
  ---------------------------------------------------------------------------------------------------------------------
     1 ASM01L   TEST2           ATC.V1R5M0.SAMPLE.ASMLLST(ASM01AL)      000052
     2 ASM01L   TEST2B          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01BL)      000040    00008C
     3 ASM01L   TEST2C          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01CL)      000030    000064    00008E
     4 ASM01L   TEST2D          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01DL)      000030    00005E
  ---------------------------------------------------------------------------------------------------------------------
```

Figure 14. Summary Report for ASM01L

Edit the CA Control File

CA uses assembler listings to determine where to insert breakpoints. You supply the names of the listing files in the CA control file (CACTL).

Make sure to use the assembler options specified in “Setup” on page 231.

The CACTL control file for the assembler summary example is *hi_lev_qual*.V1R5M0.SAMPLE.CACTL(ASM01L). The file is shown in Figure 15 on page 72.

To edit the CACTL for the ASM01L summary example:

1. Select option `1` from the `ATC Primary Option Menu`.

   The `Coverage, Distillation and Unit Test Assistant` panel is displayed.

2. Select option `1`.

   The `Work with the CA/DA/UTA Control File` panel is displayed.

3. Specify the following:

Use Program Name for File Name	`YES`
Program Name	`ASM01L`
Listing Type	`ASM`

   An ISPF edit session for the CA control file you requested is displayed.

   The data in the control file consists of the following:

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

   If the control file you requested did not previously exist, it is created with comments in it to help you enter the appropriate information in the fields. The control file shown in Figure 15 on page 72 is the control file for ASM01L. This example shows how to instrument object modules. To instrument load modules, see “Instrumentation of Load Modules instead of Object Modules” on page 238.

4. Verify that the listing file names and the copy to/from object module names are correct.

5. Press the End key (PF3) to terminate the edit session.

For more information, see “Editing the Coverage Assistant Control File” on page 81.

```
Defaults ListDsn=ATC.V1R5M0.SAMPLE.ASMLLST(*),
         LoadMod=ASM01L,
         FromObjDsn=ATC.V1R5M0.SAMPLE.OBJ,
         ToObjDsn=ATC.V1R5M0.SAMPLE.ZAPOBJ

ASM      ListMember=ASM01AL

ASM      ListMember=ASM01BL

ASM      ListMember=ASM01CL

ASM      ListMember=ASM01DL
```

Figure 15. Control File for ASM01L

Create Setup JCL

Before test cases can be executed on the test program (ASM01L), CA must insert breakpoints into the test program. During Setup, the CA Setup program analyzes the assembler listings and creates a table containing breakpoint data (address, op code, and so on). User SVC instructions are inserted for the instructions at the breakpoints in the object modules or load modules. If you instrument object modules, you then link these modified object modules into a modified ASM01L load module for CA to use.

To create the setup JCL:

1. Select option `1` from the `ATC Primary Option Menu`.

   The `Coverage, Distillation, and Unit Test Assistant` panel is displayed.

2. Select option `2`.

   The `Create JCL for Setup` panel is displayed. You create the JCL for the setup of ASM01L from this panel.

   All of the default values on this panel are correct for the ASM01L example. The defaults will usually be correct for your test coverage run. The only field that you may need to change is the `Program Name` field.

3. If necessary, change the program name to `ASM01L`.

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 identified on the panel using member name SASM01L.

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

For more detailed information, see “CA, DA, and UTA Setup” on page 231.

Create JCL to Start a Monitor Session

JCL is required to start a monitor session.

To create the JCL to start a monitor session:

1. Select option 1 from the ATC Primary Option Menu and option 3 from the Coverage, Distillation, and Unit Test Assistant panel.

   The Create JCL to Start the Monitor panel is displayed. You create the JCL for the CA execution of ASM01L from this panel.

2. If necessary, change the program name to ASM01L.
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 identified on the panel using member name XASM01L.

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

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

For more detailed information, see "Monitor Execution" on page 245.

Create Summary Report JCL

JCL is required to generate a summary report.

To create the summary report JCL:

1. Select option 1 from the ATC Primary Option Menu.

   The Coverage, Distillation and Unit Test Assistant panel is displayed.

2. Select option 4.

   The Coverage Reports panel is displayed.

3. Select option 1.

   The Create JCL for Summary Report panel is displayed. You create the JCL for generating the ASM01L summary report from this panel.

4. If necessary, change the program name to ASM01L.
5. Select option 1.

   Informational messages are written to your screen as the JCL is created. The created JCL is put into the JCL library identified on the panel using member name TASM01L.

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

For more detailed information, see "Summary Coverage Report" on page 83.

Create JCL to Link the Modified Object Modules

If you instrumented object modules, you must link the modified object modules (modified by the Setup step) into an executable program for testing. You can use the normal JCL that links your program, but be sure to specify the object module library that contains the modified object modules. Sample JCL to link the ASM01L example is provided in *hi_lev_qual*.V1R5M0.SAMPLE.JCL(LASM01L).

Create JCL to Run the GO Step

You can use the normal JCL that executes your program, but be sure to specify the load module library that contains the link-edited modified object modules. Sample JCL to execute the GO step for the ASM01L example is provided in *hi_lev_qual*.V1R5M0.SAMPLE.JCL(GASM01L).

Execute the JCL

When you have created all of the ASM01L JCL, you can run the ASM01L summary example by executing the following functions in the order listed. See Figure 7 on page 46 for a flow diagram of these steps.

1. SASM01L[18]

   Performs the Setup step. All JCL steps should end with condition code 0.

2. LASM01L[19]

   If you instrumented object modules, links the object modules that were modified with breakpoints during the Setup step into the ASM01L load module.

3. XASM01L[18]

   Starts the monitor. JCL completes with condition code 0.

4. GASM01L[19]

   Runs sample program ASM01L. ASM01L runs to completion with condition code 0.

5. CASTATS[20]

   Displays statistics with the CASTATS command. You should see a nonzero EVNTS count in the TOTALS line. (This is an optional step for illustrative purposes.)

6. CASTOP[20]

   Stops the monitor session. CA writes the statistics to disk.

7. TASM01L[18]

   Creates the summary of ASM01L. The summary is in data set proj_qual.ASM01L.SUMMARY.

---

[18] JCL created from the panels and put into the JCL library.

[19] JCL supplied with the installation materials in *hi_lev_qual*.V1R5M0.SAMPLE.JCL. (Sample JCL for all of the steps can be found in this partitioned data set [PDS].)

[20] Monitor commands issued from either the `Control the CA/DA/UTA Monitor` panel or the TSO command processor (ISPF option 6) by entering:

```
EX 'hi_lev_qual.V1R5M0.REXX(cacmd)'
```

where `cacmd` is the command issued (such as, CASTATS, CASTOP, and so on).

Annotated Assembler Listings

To produce an annotated assembler listing of ASM01L, perform the following steps. **Steps 2a through 6 are described in more detail in topics that follow in this chapter.**

1. Assemble the assembler source you want to test. This produces assembler listings needed by CA. (This has already been done for the ASM01L example. The listings are in *hi_lev_qual*.V1R5M0.SAMPLE.ASMLLST for the High Level Assembler and *hi_lev_qual*.V1R5M0.SAMPLE.ASMHLST for the H Assembler.)

   Make sure to use the assembler options specified in "Setup" on page 231.

2. Start the ATC ISPF dialog by entering the following from ISPF option 6:

   ```
   EX 'hi_lev_qual.V1R5M0.REXX(ATSTART)'
   ```

   The `ATC Primary Option Menu` is displayed (shown in Figure 2 on page 27).

   a. Edit the CA control file.

      Verify that the control file includes the listings of the object modules you want to test.

   b. Create the setup JCL.

      Create the JCL that enables the Setup job to produce a file containing breakpoint data and to modify copies of your object modules or load modules by inserting breakpoints.

   c. Create the monitor JCL.

      Create the JCL to start a monitor session.

   d. Create the JCL for an annotated listing.

      Create the JCL to produce the annotated listing after ASM01L has executed.

3. End the ATC ISPF dialog by pressing the End key (PF3) on the `ATC Primary Option Menu`.

4. Create the JCL to link the modified object modules.

   If you instrumented object modules, you must link the modified object modules into an executable program you can test. Specify the library that will contain the modified object modules and the library that will contain the modified load module.

5. Create the JCL to run the GO step.

   Create the JCL to run your program. Specify the same modified load module as in step 4.

6. Execute the JCL.

   Execute the created JCL files for ASM01L in the correct order. (This order is shown in "Execute the JCL" on page 74.)

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

& 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.

: Non-branch instruction has executed.

¬ Instruction has not executed.

@ Data area in the assembler listing.

% Unconditional branch that has been executed in the assembler listing.

Figure 16 shows a sample annotated listing.

```
  ACTIVE USINGS: NONE
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    HLASM R2.0  1997/11/14 09.55
                                      1 ********************************************************************      00000100
                                      2 *                                                                  *      00000200
                                      3 * LICENSED MATERIALS - PROPERTY OF IBM                            *      00000300
                                      4 *                                                                  *      00000400
                                      5 * 5799-GBN                                                       *      00000500
                                      6 *                                                                  *      00000600
                                      7 * (C) COPYRIGHT IBM CORP. 1997, 1998 ALL RIGHTS RESERVED         *      00000700
                                      8 *                                                                  *      00000800
                                      9 * US GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION OR     *      00000900
                                     10 * DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE CONTRACT WITH IBM     *      00001000
                                     11 * CORP.                                                          *      00001100
                                     12 *                                                                *      00001200
                                     13 *                                                                *      00001300
                                     14 ********************************************************************      00001400
                                     15 ********************************************************************      00001500
                                     16 *                                                                *      00001600
                                     17 * HIGH LEVEL ASSEMBLER TEST.                                     *      00001701
                                     18 *                                                                *      00001800
                                     19 ********************************************************************      00001900
000000                               20 TEST2    CSECT ,                                               01S0001 00002000
                                      1 ********************************************************************      00000100
                                      2 *                                                                  *      00000200
                                      3 * LICENSED MATERIALS - PROPERTY OF IBM                            *      00000300
                                      4 *                                                                  *      00000400
                                      5 * 5799-GBN                                                       *      00000500
                                      6 *                                                                  *      00000600
                                      7 * (C) COPYRIGHT IBM CORP. 1997 ALL RIGHTS RESERVED               *      00000700
                                      8 *                                                                  *      00000800
                                      9 * US GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION OR     *      00000900
                                     10 * DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE CONTRACT WITH IBM     *      00001000
                                     11 * CORP.                                                          *      00001100
                                     12 *                                                                *      00001200
                                     13 *                                                                *      00001300
                                     14 ********************************************************************      00001400
```

Figure 16 (Part 1 of 3). Annotated Assembler Listing

```
                                    15 ********************************************************************      00001500
                                    16 *                                                                  *      00001600
                                    17 * HIGH LEVEL ASSEMBLER TEST.                                       *      00001701
                                    18 *                                                                  *      00001800
                                    19 ********************************************************************      00001900
000000                              21 @MAINENT DS    0H                                              01S0001 00003000
                R:F  00000          22          USING *,@15                                           01S0001 00004000
000000 47F0 F016            00016   23%         B     @PROLOG                                         01S0001 00005000
000004 10                           24@         DC    AL1(16)                                         01S0001 00006000
000005 E3C5E2E3F2404040             25@         DC    C'TEST2     97.295'                             01S0001 00007000
                                    26          DROP  @15                                                     00008000
000015 00
000016 90EC D00C            0000C   27:@PROLOG  STM   @14,@12,12(@13)                                  01S0001 00009000
00001A 18CF                         28:         LR    @12,@15                                         01S0001 00010000
                            00000   29 @PSTART  EQU   TEST2                                           01S0001 00011000
                R:C  00000          30          USING @PSTART,@12                                     01S0001 00012000
00001C 50D0 C0B0            000B0   31:         ST    @13,@SA00001+4                                  01S0001 00013000
000020 41E0 C0AC            000AC   32:         LA    @14,@SA00001                                    01S0001 00014000
000024 50E0 D008            00008   33:         ST    @14,8(,@13)                                     01S0001 00015000
000028 18DE                         34:         LR    @13,@14                                         01S0001 00016000
                                    35 *   DO WHILE(EXPARM1>0);           /* THIS DO LOOP EXECUTED 5 TIMES */ 00017000
00002A 47F0 C042            00042   36%         B     @DE00006                                        01S0006 00018000
00002E                              37 @DL00006 DS    0H                                              01S0007 00019000
                                    38 *     EXPARM1 = EXPARM1 - 1;        /*                           */ 00020000
00002E 5810 C100            00100   39:         L     @01,EXPARM1                                     01S0007 00021000
000032 0610                         40:         BCTR  @01,0                                           01S0007 00022000
000034 5010 C100            00100   41:         ST    @01,EXPARM1                                     01S0007 00023000
                                    42 *     CALL TEST2B(PARM2);           /* TEST2B CALLED 5 TIMES        */ 00024000
000038 58F0 C0F8            000F8   43:         L     @15,@CV00063                                    01S0008 00025000
00003C 4110 C0A4            000A4   44:         LA    @01,@AL00002                                    01S0008 00026000
000040 05EF                         45%         BALR  @14,@15                                         01S0008 00027000
                                    46 *   END;                                                       01S0009 00028000
000042 5800 C100            00100   47:@DE00006 L     @00,EXPARM1                                     01S0009 00029000
000046 1200                         48:         LTR   @00,@00                                         01S0009 00030000
000048 4720 C02E            0002E   49&         BP    @DL00006                                        01S0009 00031000
                                    50 *   IF (EXPARM2 = 0) THEN          /* THIS BRANCH ALWAYS TAKEN      */ 00032000
00004C 5810 C104            00104   51:         L     @01,EXPARM2                                     01S0010 00033000
000050 1211                         52:         LTR   @01,@01                                         01S0010 00034000
000052 4770 C06C            0006C   53>         BNZ   @RF00010                                        01S0010 00035000
000056 4110 C0A8            000A8   55¬         LA    @01,@AL00003                                    01S0011 00037000
00005A 45E0 C086            00086   56¬         BAL   @14,PROC1                                       01S0011 00038000
                                    57 *   DO WHILE(EXPARM2>0);           /* DO LOOP EXECUTED TWICE        */ 00039000
00005E 47F0 C06C            0006C   58¬         B     @DE00012                                        01S0012 00040000
000062                              59 @DL00012 DS    0H                                              01S0013 00041000
                                    60 *     EXPARM2 = EXPARM2 - 1;                                    01S0013 00042000
000062 5820 C104            00104   61:         L     @02,EXPARM2                                     01S0013 00043000
000066 0620                         62:         BCTR  @02,0                                           01S0013 00044000
000068 5020 C104            00104   63:         ST    @02,EXPARM2                                     01S0013 00045000
                                    64 *   END;                                                       01S0014 00046000
00006C 5830 C104            00104   65:@DE00012 L     @03,EXPARM2                                     01S0014 00047000
000070 1233                         66:         LTR   @03,@03                                         01S0014 00048000
000072 4720 C062            00062   67&         BP    @DL00012                                        01S0014 00049000
                                    68 *   RETURN CODE(0);                                            01S0015 00050000
000076 1FFF                         69:         SLR   @15,@15                                         01S0015 00051000
000078 58D0 D004            00004   70:         L     @13,4(,@13)                                     01S0015 00052000
00007C 58E0 D00C            0000C   71:         L     @14,12(,@13)                                    01S0015 00053000
000080 980C D014            00014   72:         LM    @00,@12,20(@13)                                 01S0015 00054000
000084 07FE                         73%         BR    @14                                             01S0015 00055000
                                    74 *   END TEST2;                                                 01S0020 00056000
                                    75 *PROC1:                                                        01S0016 00057000
                                    76 *   PROCEDURE(P1PARM1);            /* THIS PROCEDURE NEVER EXECUTED */ 00058000
000086 90EC D00C            0000C   77¬PROC1    STM   @14,@12,12(@13)                                 01S0016 00059000
00008A D203 C0F4 1000 000F4 00000   78¬         MVC   @PC00002(4),0(@01)                              01S0016 00060000
                                    79 *   P1PARM1 = 10;                                              01S0018 00061000
```

Figure 16 (Part 2 of 3). Annotated Assembler Listing

```
000090 5820 C0F4          000F4   80¬         L     @02,@PA00064                                  01S0018 00062000
000094 4130 000A          0000A   81¬         LA    @03,10                                        01S0018 00063000
000098 5030 2000          00000   82¬         ST    @03,P1PARM1(,@02)                             01S0018 00064000
                                  83 *   END PROC1;                                               01S0019 00065000
00009C                            84 @EL00002 DS    0H                                            01S0019 00066000
00009C                            85 @EF00002 DS    0H                                            01S0019 00067000
00009C 98EC D00C          0000C   86¬@ER00002 LM    @14,@12,12(@13)                               01S0019 00068000
0000A0 07FE                       87¬         BR    @14                                           01S0019 00069000
0000A2                            88 @DATA    DS    0H                                                    00070000
0000A4                            89          DS    0F                                                    00071000
0000A4                            90 @AL00002 DS    0A                                                    00072000
0000A4 00000108                   91@         DC    A(PARM2)                                              00073000
0000A8                            92 @AL00003 DS    0A                                                    00074000
0000A8 00000104                   93@         DC    A(EXPARM2)                                            00075000
0000AC                            94          DS    0F                                                    00076000
0000AC                            95@@SA00001 DS    18F                                                   00077000
0000F4                            96@@PC00002 DS    1F                                                    00078000
0000F8                            97          DS    0F                                                    00079000
0000F8 00000000                   98@@CV00063 DC    V(TEST2B)                                             00080000
000100                            99          LTORG                                                         00081000
000100                           100          DS    0D                                                    00082000
000100 00000005                  101@EXPARM1  DC    F'5'                                                  00083000
000104 00000002                  102@EXPARM2  DC    F'2'                                                  00084000
000108 00000002                  103@PARM2    DC    F'2'                                                  00085000
                          00000  104 @DYNSIZE EQU   0                                                     00086000
                          00000  105 @00      EQU   0                                                     00087000
                          00001  106 @01      EQU   1                                                     00088000
                          00002  107 @02      EQU   2                                                     00089000
                          00003  108 @03      EQU   3                                                     00090000
                          00004  109 @04      EQU   4                                                     00091000
                          00005  110 @05      EQU   5                                                     00092000
                          00006  111 @06      EQU   6                                                     00093000
                          00007  112 @07      EQU   7                                                     00094000
                          00008  113 @08      EQU   8                                                     00095000
                          00009  114 @09      EQU   9                                                     00096000
                          0000A  115 @10      EQU   10                                                    00097000
                          0000B  116 @11      EQU   11                                                    00098000
                          0000C  117 @12      EQU   12                                                    00099000
                          0000D  118 @13      EQU   13                                                    00100000
                          0000E  119 @14      EQU   14                                                    00101000
                          0000F  120 @15      EQU   15                                                    00102000
                          00000  121 P1PARM1  EQU   0,4,C'F'                                              00103000
                          000F4  122 @PA00064 EQU   @PC00002,4,C'F'                                       00104000
                          0006C  123 @RF00010 EQU   @DE00012                                              00105000
000110                           124          DS    0D                                                    00106000
                          00110  125 @ENDDATA EQU   *                                                     00107000
                          00110  126 @MODLEN  EQU   @ENDDATA-TEST2                                        00108000
                                 127          END   ,                                                     00109000
```

Figure 16 (Part 3 of 3). Annotated Assembler Listing

Edit the CA Control File

This step is identical to the corresponding procedure for summary reports at “Edit the CA Control File” on page 71.

Create Setup JCL

This step is identical to the corresponding procedure for summary reports at “Create Setup JCL” on page 72.

Create JCL to Start a Monitor Session

This step is identical to the corresponding procedure for summary reports at “Create JCL to Start a Monitor Session” on page 73.

Create JCL for an Annotated Listing

To create the annotated listing JCL:

1. Select option `1` from the `ATC Primary Option Menu`.

   The `Coverage, Distillation and Unit Test Assistant` panel is displayed.

2. Select option `4`.

   The `Coverage Reports` panel is displayed.

3. Select option `2`.

   The `Create JCL for Summary and Annotation Report` panel is displayed. You create the JCL for printing the annotated listing (along with a summary) for ASM01L from this panel.

4. If necessary, change the program name to `ASM01L`.

5. Select option `1`.

   Informational messages are written to your screen as the JCL is created. The created JCL is put into the JCL library identified on the panel using member name RASM01L.

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

For more detailed information, see “Annotated Listing Coverage Report” on page 95.

Create JCL to Link the Modified Object Modules

This step is identical to the corresponding procedure for summary reports at “Create JCL to Link the Modified Object Modules” on page 74.

Create JCL to Run the GO Step

This step is identical to the corresponding procedure for summary reports at “Create JCL to Run the GO Step” on page 74.

Execute the JCL

When you have created all of the ASM01L JCL, you can run the ASM01L example by executing the following JCL in the order listed. See Figure 7 on page 46 for a flow diagram of these steps.

1. SASM01L[21]

   Performs the Setup step. All JCL steps should complete with condition code 0.

2. LASM01L[22]

   If you instrumented object modules, links the object modules that were modified with breakpoints during the Setup step into the ASM01L load module.

3. XASM01L[21]

   Starts the monitor. JCL completes with condition code 0.

4. GASM01L[22]

   Runs sample program ASM01L. ASM01L runs to completion with condition code 0.

5. CASTATS[23]

   Displays statistics with the CASTATS command. You should see a nonzero EVNTS count in the TOTALS line. (This is an optional step for illustrative purposes.)

6. CASTOP[23]

   Stops the monitor session. CA writes the statistics to disk.

7. RASM01L[21]

   Creates the annotated listing (and summary) of ASM01L. The report is in data set proj_qual.ASM01L.REPORT.

---

[21] JCL created from the panels and put into the JCL library.

[22] JCL supplied with the installation materials in *hi_lev_qual*.V1R5M0.SAMPLE.JCL. (Sample JCL for all of the steps can be found in this partitioned data set [PDS].)

[23] Monitor commands issued from either the `Control the CA/DA/UTA Monitor` panel or the TSO command processor (ISPF option 6) by entering:

```
EX 'hi_lev_qual.V1R5M0.REXX(cacmd)'
```

where `cacmd` is the command issued (such as, CASTATS, CASTOP, and so on).

Editing the Coverage Assistant Control File

This chapter describes the function of the control file (CACTL) used by CA. The CACTL contains the names of the compiler listings and copy to/from information. CA, DA, and UTA share the CACTL file. See “CA, DA, and UTA Control File” on page 209 for a complete description of this control file. This chapter only explains how to use the control file with Coverage Assistant.

Contents of the Control File

The control file defines the programs to be analyzed. A sample control file (CACTL) used to produce a COBOL summary or an annotated COBOL listing is shown in Figure 9 on page 50. The sample control file for a PL/I summary or an annotated PL/I listing is shown in Figure 12 on page 62. The sample file for an ASM summary or an annotated ASM listing is shown in Figure 15 on page 72. The statements and operands used in these control files can be summarized as follows:

- The **DEFAULTS** statement specifies default values to be used by subsequent **COBOL**, **PL/I**, and **ASM** statements.
- The **COBOL** statement indicates a COBOL program specification.
- The **PL/I** statement indicates a PL/I program specification.
- The **ASM** statement indicates an assembler program specification.
- The operand of the **LISTDSN=** keyword specifies the data set containing the program's compiler listing.
- The operand of the **LISTMEMBER=** keyword specifies the member name of a particular program listing in the LISTDSN data set. This operand replaces the asterisk in the LISTDSN specification in the DEFAULTS statement.
- The operand of the **LOADMOD=** keyword specifies the load module member name.
- The operand of the **FROMOBJDSN=** keyword specifies the data set name of the partitioned data set that contains the object code.
- The operand of the **TOOBJDSN=** keyword specifies the data set name of the partitioned data set that is to contain the instrumented object code generated by the CA Setup job.
- The **OBJMEMBER=** keyword that indicates the member name of the object code within the FROMOBJDSN and TOOBJDSN data sets. If it is not specified, it defaults to the value of the LISTMEMBER operand.
- The **FROMLOADDSN=** keyword specifies the data set name of the partitioned data set that contains the input load module.
- The **TOLOADDSN=** keyword specifies the data set name of the partitioned data set that is to contain the instrumented load module generated by the CA Setup job.

Note: The FROMOBJDSN and TOOBJDSN keywords are mutually exclusive of the FROMLOADDSN and TOLOADDSN keywords.

© Copyright IBM Corp. 1997, 1999

See “Contents of the Control File” on page 211 for a description of the syntax of the control file.

Note: The control file shown in Figure 9 on page 50 contains both CA and UTA control information. Only the DEFAULTS, COBOL, PL/I, and ASM statements are processed by CA.

Coverage Assistant Reports

CA can generate three kinds of reports to describe test case coverage:

- **Summary**

  A summary of the test case coverage using statement numbers of a high-level language or offsets for the assembler language.

- **Annotated listings**

  Annotated compiler and assembler listings with a character on each break-pointed statement to describe how each statement was executed.

- **Targeted Summary**

  A summary report that provides coverage information on specific "target" statements, which can be identified by statement number or, for COBOL and PL/I, by statements that reference specific variables.

You can select the type of CA report you want from the `Coverage Reports` panel.

Summary Coverage Report

The summary coverage report gives statistics on the coverage of all program areas (PAs) during the test run. The summary report is divided into the following sections:

- Program Area Data
  Lists summary data on each PA. 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. See "PROGRAM AREA DATA" on page 86 for details.
- Unexecuted Code
  Lists the unexecuted code statements in each PA. See "UNEXECUTED CODE" on page 87 for details.
- Branches That Have Not Gone Both Ways
  Lists the conditional branches that have not executed in both directions for each PA. See "BRANCHES THAT HAVE NOT GONE BOTH WAYS" on page 87 for details.

Examples of summary reports are shown in Figure 17 on page 84 and Figure 18 on page 85.

Each section of a report contains the date, time, and test case ID (if provided) of the CA test run. You may provide the test case ID with the CAIDADD operator command during a monitor session. See "CAIDADD" on page 261 for details on this command. A summary of data for all PAs is displayed following the PA-specific data.

The INTERNAL option was used to create the report in Figure 17 on page 84. 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.

© Copyright IBM Corp. 1997, 1999

```
1 *********  CA SUMMARY:                    PROGRAM AREA DATA                    ********
0                DATE: 02/15/1999
                 TIME: 11:47.41
          TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                            |                                        |  STATEMENTS:          | BRANCHES:          |
  | PA LOAD MOD PROCEDURE      | LISTING NAME                           |   TOTAL     EXEC   %  |  CPATH   TAKEN   %  |
  ---------------------------------------------------------------------------------------------------------------------
     1 COB01M                   ATC.V1R5M0.SAMPLE.COBOLST(COB01AM)            6        6 100.0       0       0 100.0
     2          PROGA                                                         8        7  87.5       6       5  83.3
     3          PROCA                                                         1        0   0.0       0       0 100.0
     4          PROGB                                                         7        5  71.4       6       3  50.0
     5          PROCB                                                         2        2 100.0       0       0 100.0
     6 COB01M   PROGC            ATC.V1R5M0.SAMPLE.COBOLST(COB01CM)            7        5  71.4       6       5  83.3
     7          PROCC                                                         3        2  66.7       2       1  50.0
     8 COB01M   PROGD            ATC.V1R5M0.SAMPLE.COBOLST(COB01DM)            6        0   0.0       6       0   0.0
     9          PROCD                                                         1        0   0.0       0       0 100.0
  ---------------------------------------------------------------------------------------------------------------------
  Summary for all PAs:                                                        41       27  65.9      26      14  53.8

1 *********  CA SUMMARY:                    UNEXECUTED CODE                      ********
0                DATE: 02/15/1999
                 TIME: 11:47.41
          TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                            |                                        |
  | PA LOAD MOD PROCEDURE      | LISTING NAME                           | start    end     start    end     start   end
  ---------------------------------------------------------------------------------------------------------------------
     2 COB01M   PROGA            ATC.V1R5M0.SAMPLE.COBOLST(COB01AM)          67     67
     3          PROCA                                                        79     79
     4          PROGB                                                       118    119
     6 COB01M   PROGC            ATC.V1R5M0.SAMPLE.COBOLST(COB01CM)          39     40
     7          PROCC                                                        58     58
     8 COB01M   PROGD            ATC.V1R5M0.SAMPLE.COBOLST(COB01DM)          37     46
     9          PROCD                                                        51     51
  ---------------------------------------------------------------------------------------------------------------------

1 *********  CA SUMMARY:                    BRANCHES THAT HAVE NOT GONE BOTH WAYS ********
0                DATE: 02/15/1999
                 TIME: 11:47.41
          TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                            |                                        |
  | PA LOAD MOD PROCEDURE      | LISTING NAME                           |  stmt      stmt      stmt      stmt      stmt
  ---------------------------------------------------------------------------------------------------------------------
     2 COB01M   PROGA            ATC.V1R5M0.SAMPLE.COBOLST(COB01AM)          66
     4          PROGB                                                       113       118
     6 COB01M   PROGC            ATC.V1R5M0.SAMPLE.COBOLST(COB01CM)          37
     7          PROCC                                                        57
     8 COB01M   PROGD            ATC.V1R5M0.SAMPLE.COBOLST(COB01DM)          37        41        45
  ---------------------------------------------------------------------------------------------------------------------
```

Figure 17. Summary Coverage Report for COB01M in COBOL

The INTERNAL option was used to create the report in Figure 18 on page 85. 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.

```
1 *********  CA SUMMARY:                    PROGRAM AREA DATA                    ********
0                DATE: 02/15/1999
                 TIME: 12:28.34
         TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                            |                                        |  STATEMENTS:          | BRANCHES:          |
  | PA LOAD MOD PROCEDURE      | LISTING NAME                           |   TOTAL     EXEC   %  |  CPATH   TAKEN   %  |
  ---------------------------------------------------------------------------------------------------------------------
     1 PLI01M   PLI01AM          ATC.V1R5M0.SAMPLE.PLILST(PLI01AM)              9        9 100.0       6       5  83.3
     2          PROC2A                                                         2        0   0.0       0       0 100.0
     3 PLI01M   PLI01BM          ATC.V1R5M0.SAMPLE.PLILST(PLI01BM)             11        8  72.7       6       4  66.7
     4          PROC1                                                          2        2 100.0       0       0 100.0
     5 PLI01M   PLI01CM          ATC.V1R5M0.SAMPLE.PLILST(PLI01CM)              7        4  57.1       6       3  50.0
     6          PROC1                                                          4        3  75.0       2       1  50.0
     7 PLI01M   PLI01DM          ATC.V1R5M0.SAMPLE.PLILST(PLI01DM)              2        0   0.0       2       0   0.0
     8          PROC1                                                          4        0   0.0       2       0   0.0
  ---------------------------------------------------------------------------------------------------------------------
  Summary for all PAs:                                                         41       26  63.4      24      13  54.2

1 *********  CA SUMMARY:                    UNEXECUTED CODE                      ********
0                DATE: 02/15/1999
                 TIME: 12:28.34
         TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                            |                                        |
  | PA LOAD MOD PROCEDURE      | LISTING NAME                           | start    end     start    end     start   end
  ---------------------------------------------------------------------------------------------------------------------
     2 PLI01M   PROC2A           ATC.V1R5M0.SAMPLE.PLILST(PLI01AM)            17     18
     3 PLI01M   PLI01BM          ATC.V1R5M0.SAMPLE.PLILST(PLI01BM)             9      9        15     16
     5 PLI01M   PLI01CM          ATC.V1R5M0.SAMPLE.PLILST(PLI01CM)             6      6        10     11
     6          PROC1                                                        16     16
     7 PLI01M   PLI01DM          ATC.V1R5M0.SAMPLE.PLILST(PLI01DM)             3     11
     8          PROC1                                                         6     10
  ---------------------------------------------------------------------------------------------------------------------

1 *********  CA SUMMARY:                    BRANCHES THAT HAVE NOT GONE BOTH WAYS ********
0                DATE: 02/15/1999
                 TIME: 12:28.34
         TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                            |                                        |
  | PA LOAD MOD PROCEDURE      | LISTING NAME                           |  stmt      stmt      stmt      stmt      stmt
  ---------------------------------------------------------------------------------------------------------------------
     1 PLI01M   PLI01AM          ATC.V1R5M0.SAMPLE.PLILST(PLI01AM)            10
     3 PLI01M   PLI01BM          ATC.V1R5M0.SAMPLE.PLILST(PLI01BM)             8        14
     5 PLI01M   PLI01CM          ATC.V1R5M0.SAMPLE.PLILST(PLI01CM)             5         8         9
     6          PROC1                                                        15
     7 PLI01M   PLI01DM          ATC.V1R5M0.SAMPLE.PLILST(PLI01DM)             3
     8          PROC1                                                         7
  ---------------------------------------------------------------------------------------------------------------------
```

Figure 18. Summary Coverage Reports for PLI01M in PL/I

Areas of the Report

PROGRAM AREA DATA

The portion of the report called PROGRAM AREA DATA contains the following information:

- PROGRAM IDENTIFICATION
  - The load module name, paragraph name, and listing name (COBOL)
  - The load module name, the procedure, ON-unit, or Begin-block name, and the listing name (PL/I)
  - The load module name, CSECT name, and listing name (ASM)
- Coverage statistics

  Data on test case coverage for each PA.

PROGRAM IDENTIFICATION

The fields in the PROGRAM IDENTIFICATION area are:

PA	Number of the program area (PA)
LOAD MOD	Load module name.
PROCEDURE	COBOL: Paragraph name. These are shown in source order. 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 CA-generated name). These are shown in source order. ASM: CSECT name.
LISTING NAME	Listing name.

Coverage Statistics

The fields in Coverage Statistics are:

STATEMENTS: TOTAL	The statements of code for this test case run
STATEMENTS: EXEC	The statements of code that were executed
STATEMENTS: %	The percentage of statements that were executed
BRANCHES: CPATH	The number of conditional branch paths (number of conditional branches multiplied by 2)
BRANCHES: TAKEN	The number of conditional branch paths that were executed
BRANCHES: %	The percentage of conditional branch paths that were executed

UNEXECUTED CODE

The portion of the report called UNEXECUTED CODE contains the following information:

PROGRAM IDENTIFICATION	Described previously under PROGRAM IDENTIFICATION.
start	The line number of the first unexecuted instruction in this unexecuted segment.
end	The line number of the last unexecuted instruction in this unexecuted segment.

BRANCHES THAT HAVE NOT GONE BOTH WAYS

The portion of the report called BRANCHES THAT HAVE NOT GONE BOTH WAYS contains the following information:

PROGRAM IDENTIFICATION	Described previously under PROGRAM IDENTIFICATION.
stmt	The line number of the conditional branch instruction that did not execute in both directions.

Suppression of Conditional Branch Coverage with Performance Mode

When the Performance Mode (described at “Using the Performance Mode to Reduce Monitor Overhead” on page 253) is used, the breakpoints for conditional branches are not kept in storage, and the conditional branch coverage data is inaccurate. For summaries produced from test runs when the Performance Mode is enabled during SETUP, the conditional branch coverage data is suppressed. In the PROGRAM AREA DATA section of the summary report, the BRANCHES section is blank. In the BRANCHES THAT HAVE NOT GONE BOTH WAYS section, the conditional branches are not listed.

If the summary is performed on a test run where some object modules are tested with the Performance Option on, and some with it off, those object modules that are tested with it off have their conditional branch coverage listed.

Examples of Reports Using Listings

The ATC installation package contains the following examples:

- COB01M (COBOL for MVS & VM example)
- COB012 (VS COBOL II example)
- COB01O (OS/VS COBOL example)
- COB02M (COBOL for MVS & VM example)
- COB022 (VS COBOL II example)
- COB02O (OS/VS COBOL example)
- PLI01M (PL/I for MVS & VM example)
- PLI012 (PL/I 2.3.0 example)
- PLI011 (PL/I 1.5.1 example)
- ASM01L (High Level Assembler 1.2.0 example)
- ASM01H (Assembler H 2.1.0 example)

All of these examples include sample coverage reports. COB01M, COB02M, and PLI01M also include sample targeted coverage reports.

Summary Report for Assembler

The summary for a High Level Assembler or an Assembler H program differs from the high-level summaries described previously. For assembler, code references are in offsets within the listing, not references to statement numbers. For an example summary report, see Figure 19 on page 89.

Areas of the Report

EXECUTED CODE

STATEMENTS: TOTAL — The bytes of executable code in the program. Data areas occurring anywhere in the listing are excluded from this count.

STATEMENTS: EXEC — The bytes of executable code in the program that executed.

UNEXECUTED CODE

START — The offset within the PA of the first unexecuted instruction in this unexecuted segment

END — The offset within the PA of the last unexecuted instruction in this unexecuted segment

BRANCHES THAT HAVE NOT GONE BOTH WAYS

STMT — The offset within the PA of the conditional branch instruction that did not execute in both directions

ASTERISKS BY STATEMENT TOTALS

If your code contains relative branching, for example:

```
B   *+20
```

your coverage data may not be accurate (indicated by an * in the STATEMENT TOTALS column). To get accurate coverage in this case, enable the frequency mode flag in defaults and recreate your SETUP JCL (and rerun all steps).

```
1 ********* CASUMMARY:                    PROGRAM AREA DATA                    ********
0               DATE: 04/22/1998
                TIME: 15:09.53
        TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                           |                                         |  STATEMENTS:          | BRANCHES:           |
  | PA LOAD MOD PROCEDURE     | LISTING NAME                            |   TOTAL     EXEC   %  |  CPATH   TAKEN   %  |
  ---------------------------------------------------------------------------------------------------------------------
     1 ASM01L   TEST2           ATC.V1R5M0.SAMPLE.ASMLLST(ASM01AL)      *    144      104  72.2       6       5  83.3
     2 ASM01L   TEST2B          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01BL)      *    164      144  87.8       6       4  66.7
     3 ASM01L   TEST2C          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01CL)      *    140      118  84.3       8       5  62.5
     4 ASM01L   TEST2D          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01DL)      *     92        0   0.0       4       0   0.0
  ---------------------------------------------------------------------------------------------------------------------
  Summary for all PAs:                                                        540      366  67.8      24      14  58.3

1 ********* CASUMMARY:                    UNEXECUTED CODE                      ********
0               DATE: 04/22/1998
                TIME: 15:09.53
        TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                           |                                         |
  | PA LOAD MOD PROCEDURE     | LISTING NAME                            | start    end     start    end     start   end
  ---------------------------------------------------------------------------------------------------------------------
     1 ASM01L   TEST2           ATC.V1R5M0.SAMPLE.ASMLLST(ASM01AL)      000056 000062    000086 0000A0
     2 ASM01L   TEST2B          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01BL)      000050 00005A    000078 000082
     3 ASM01L   TEST2C          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01CL)      000034 00003A    000050 00005A    000092 000098
     4 ASM01L   TEST2D          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01DL)      000000 000000    000016 00006C
  ---------------------------------------------------------------------------------------------------------------------

1 ********* CASUMMARY:                    BRANCHES THAT HAVE NOT GONE BOTH WAYS ********
0               DATE: 04/22/1998
                TIME: 15:09.53
        TEST CASE ID:
0 |<--                    PROGRAM IDENTIFICATION                    -->|
  |                           |                                         |
  | PA LOAD MOD PROCEDURE     | LISTING NAME                            |  stmt      stmt      stmt      stmt      stmt
  ---------------------------------------------------------------------------------------------------------------------
     1 ASM01L   TEST2           ATC.V1R5M0.SAMPLE.ASMLLST(ASM01AL)      000052
     2 ASM01L   TEST2B          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01BL)      000040    00008C
     3 ASM01L   TEST2C          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01CL)      000030    000064    00008E
     4 ASM01L   TEST2D          ATC.V1R5M0.SAMPLE.ASMLLST(ASM01DL)      000030    00005E
  ---------------------------------------------------------------------------------------------------------------------
```

Figure 19. Summary Report for ASM01L

Creating Coverage Reports

To create a coverage report, select option 1 from the ATC Primary Option Menu, and then option 4 from the Coverage, Distillation, and Unit Test Assistant panel. The Coverage Reports panel, shown in Figure 20 is displayed. This panel allows you to select the kind of report you want.

```
------------------------------ Coverage Reports ------------------------------
Option ===>

1  Summary       Create JCL for Summary Report
2  Annotation    Create JCL for Summary and Annotation Report
3  Combine       Create JCL for Combining Multiple Runs
4  Targeted      Generate a Targeted Summary Report

Enter END to Terminate
```

Figure 20. Coverage Reports Panel

Summary Create JCL for a summary report.

Annotation Create JCL for both summary and annotated listing reports.

Combine Create JCL for combining multiple CA coverage runs.

Targeted Generate a targeted summary report from a coverage run.

Creating Summary and Annotated Listing Report JCL Using the Panels

To create a summary report, select option 1 on the Coverage Reports panel. The Create JCL for Summary Report panel, shown in Figure 21, allows you 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  COB01M

Input Files:
  Breakpoint Table Dsn. 'YOUNG.TEST.COB01M.BRKTAB'
  Breakout Dsn. . . . . 'YOUNG.TEST.COB01M.BRKOUT'

JCL Library and Member:
  JCL Dsn . . . . . 'YOUNG.TEST.JCL.CNTL(Txxxxxxx)'

Output Summary Type and File:
  Type. . . . . INTERNAL (Internal|External)
  Report Dsn  . . . . . 'YOUNG.TEST.COB01M.SUMMARY'
    (* for default sysout class)
```

Figure 21. Create JCL for Summary Report Panel

Generate

Generate JCL from the parameters you have specified on the panel.

Edit

Make changes to existing JCL.

Submit

Submit for execution the JCL specified in the JCL Dsn field on this panel.

Use Program Name for File Name

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, enter YES.

When you press Enter, the file names on the panel are changed automatically. Using the program name to construct the data set names is the normal CA procedure.

Program Name

Name to use for CA files if you enter `YES` in the `Use Program Name for File Name` field. Note that this 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.TEST.COB01M.BRKTAB'

- Partitioned data sets:

  `'proj_qual.file_type(program_name)'`

  For example: 'YOUNG.TEST.BRKTAB(COB01M)'

Input Files

Names of the breakpoint table and breakout data sets.

JCL Dsn

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

Note: 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 T*xxxxxxx*, where *xxxxxxx* is the first seven characters of the program name.

Output Summary Type and File

Type of summary report (internal or external) and the name of the data set containing the summary report.

To create a summary report and annotated listings, select option 2 on the Coverage Reports panel. The Create JCL for Summary and Annotation Report panel, shown in Figure 22, allows you to specify summary report and annotated listing 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  COB01M

Input Files:
  Control File Dsn. . . 'YOUNG.TEST.COB01M.CACTL'
  Breakpoint Table Dsn. 'YOUNG.TEST.COB01M.BRKTAB'
  Breakout Dsn. . . . . 'YOUNG.TEST.COB01M.BRKOUT'

JCL Library and Member:
  JCL Dsn . . . . . 'YOUNG.TEST.JCL.CNTL(Rxxxxxxx)'

Output Summary Type and Annotation File:
  Type. . . . . INTERNAL (Internal|External)
  Report Dsn  . . . . . 'YOUNG.TEST.COB01M.REPORT'
    (* for default sysout class)
```

Figure 22. Create JCL for Summary and Annotated Listing Report Panel

The panel's options and fields are as follows:

Generate

Generate JCL from the parameters you have specified on the panel.

Edit

Make changes to existing JCL.

Submit

Submit for execution the JCL specified in the JCL Dsn field on this panel.

Use Program Name for File Name

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, enter YES.

When you press Enter, the file names on the panel are changed automatically. Using the program name is the normal CA procedure.

Program Name

Name to use for CA files if you enter `YES` in the `Use Program Name for File Name` field. Note that this 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.TEST.COB01M.BRKTAB'

- Partitioned data sets:

  `'proj_qual.file_type(program_name)'`

  For example: 'YOUNG.TEST.BRKTAB(COB01M)'

Input Files

Names of the control file (CACTL), breakpoint table, and breakout data sets.

JCL Dsn

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

Note: 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 R*xxxxxxx*, where *xxxxxxx* is the first seven characters of the program name.

Output Summary Type and Annotation File

Type of summary report (internal or external) and the name of the data set containing the summary and annotated listing report.

Creating JCL for a Summary without Annotated Listings

If you only want a summary of test case coverage results, without the annotated listings, select option `1` from the `Coverage Reports` panel.

Annotated Listing Coverage Report

You can create two kinds of annotated listings to show code coverage:

All Every line of the listing is printed.

Unexecuted Only unexecuted instructions or conditional branch instructions that have not gone both ways are printed.

Each instruction line in the listings has an annotation character placed to the right of the statement number to indicate what happened during the test run:

& A conditional branch instruction has executed both ways.[24]

> A conditional branch instruction has branched, but not fallen through.[24]

V A conditional branch instruction has fallen through, but not branched.[24]

: 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 you prefer by supplying a parameter to the REPORT program.

Figure 23 on page 96 shows a sample COBOL annotated listing in which all lines have been printed.

Figure 24 on page 98 shows a sample PL/I annotated listing in which all lines have been printed.

Figure 25 on page 99 shows a sample ASM annotated listing in which all lines have been printed.

[24] The annotation of statements with conditional branches is affected by the Performance Mode. For more information, see "Changes in Annotation Symbols with Performance Mode" on page 104

```
000001                IDENTIFICATION DIVISION.
000002                PROGRAM-ID. COB01AM.
000003               ******************************************************************
000004               *                                                                *
000005               * LICENSED MATERIALS - PROPERTY OF IBM                           *
000006               *                                                                *
000007               * 5799-GBN                                                       *
000008               *                                                                *
000009               * (C) COPYRIGHT IBM CORP. 1997, 1998 ALL RIGHTS RESERVED         *
000010               *                                                                *
000011               * US GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION OR    *
000012               * DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE CONTRACT WITH IBM    *
000013               * CORP.                                                          *
000014               *                                                                *
000015               *                                                                *
000016               ******************************************************************
000017               ******************************************************************
000018               *                                                                *
000019               * COBOL FOR MVS & VM TEST.                                       *
000020               *                                                                *
000021               * MEMBER COB01AM HAS ENTRY POINT COB01AM.                        *
000022               * CALLS COB01BM, WHICH CALLS COB01CM, WHICH CALLS COB01DM.       *
000023               ******************************************************************
000024
000025                ENVIRONMENT DIVISION.
000026
000027                DATA DIVISION.
000028
000029                WORKING-STORAGE SECTION.
000030                01 TAPARM1      PIC 99 VALUE 5.
000031                01 TAPARM2      PIC 99 VALUE 2.
000032                01 COB01BM      PIC X(7) VALUE 'COB01BM'.
000033                01 P1PARM1      PIC 99 VALUE 0.
000034
000035                01 TASTRUCT.
000036                  05 LOC-ID.
000037                    10 STATE    PIC X(2).
000038                    10 CITY     PIC X(3).
000039                  05 OP-SYS     PIC X(3).
000040
000041                PROCEDURE DIVISION.
000042
000043               * THE FOLLOWING ALWAYS PERFORMED
000044
000045               * ACCESS BY TOP LEVEL QUALIFIER
000046 :                   MOVE 'ILCHIMVS' TO TASTRUCT.
000047
000048               * ACCESS BY MID LEVEL QUALIFIERS
000049 :                   MOVE 'ILSPR' TO LOC-ID.
000050 :                   MOVE 'AIX' TO OP-SYS.
000051
000052               * ACCESS BY LOW LEVEL QUALIFIERS
000053 :                   MOVE 'KY' TO STATE.
000054 :                   MOVE 'LEX' TO CITY.
000055 :                   MOVE 'VM ' TO OP-SYS.
000056
000057                PROGA.
000058
000059               * THIS PERFORM EXECUTED
000060 &                   PERFORM WITH TEST BEFORE UNTIL TAPARM1 = 0
000061 :    1                SUBTRACT 1 FROM TAPARM1
000062 :    1                CALL 'COB01BM'
000063                    END-PERFORM
000064
000065               * THIS IF ALWAYS FALSE
000066 >                   IF TAPARM2 = 0
000067 ¬    1                PERFORM PROCA
000068                    END-IF
```

Figure 23 (Part 1 of 2). Annotated COBOL Listing

```
000069
000070               * THIS PERFORM EXECUTED
000071 &                  PERFORM WITH TEST BEFORE UNTIL TAPARM2 = 0
000072 :    1               SUBTRACT 1 FROM TAPARM2
000073                   END-PERFORM
000074 :                 STOP RUN
000075                   .
000076
000077                PROCA.
000078               * PROCA NEVER CALLED
000079 ¬                 MOVE 10 TO P1PARM1
000080                   .
000081
000082               * START OF COB01BM NESTED IN COB01AM
000083
000084   1            IDENTIFICATION DIVISION.
000085   1            PROGRAM-ID. COB01BM.
000086   1           ****************************************************************
000087   1           *                                                              *
000088   1           * COBOL FOR MVS & VM TEST.                                     *
000089   1           *                                                              *
000090   1           * COB01BM, CALLED BY COB01AM.                                  *
000091   1           ****************************************************************
000092   1
000093   1            ENVIRONMENT DIVISION.
000094   1
000095   1            DATA DIVISION.
000096   1
000097   1            WORKING-STORAGE SECTION.
000098   1            01 TBPARM1      PIC 99 VALUE 5.
000099   1            01 TBPARM2      PIC 99 VALUE 0.
000100   1            01 COB01CM      PIC X(7) VALUE 'COB01CM'.
000101   1            01 P1PARM1      PIC 99 VALUE 0.
000102   1
000103   1            PROCEDURE DIVISION.
000104   1
000105   1            PROGB.
000106   1           * THIS PERFORM EXECUTED
000107 & 1                PERFORM WITH TEST BEFORE UNTIL TBPARM1 = 0
000108 : 1  1               SUBTRACT 1 FROM TBPARM1
000109 : 1  1               CALL 'COB01CM'
000110   1                END-PERFORM
000111   1
000112   1           * THIS IF EXECUTED
000113 V 1                IF TBPARM2 = 0
000114 : 1  1               PERFORM PROCB
000115   1                END-IF
000116   1
000117   1           * THIS PERFORM NOT EXECUTED
000118 ¬ 1                PERFORM WITH TEST BEFORE UNTIL TBPARM2 = 0
000119 ¬ 1  1               SUBTRACT 1 FROM TBPARM2
000120   1                END-PERFORM
000121   1                .
000122   1
000123   1            PROCB.
000124   1           * PROCB EXECUTED
000125 : 1                MOVE 10 TO P1PARM1
000126   1                .
000127   1
000128 : 1                EXIT PROGRAM.
000129   1
000130   1            END PROGRAM COB01BM.
000131                END PROGRAM COB01AM.
```

Figure 23 (Part 2 of 2). Annotated COBOL Listing

```
 1      0  PLI01AM:PROC OPTIONS(MAIN);         /* PL/I FOR MVS & VM TEST
           /*********************************************************************/
           /*                                                                   */
           /* Licensed Materials - Property of IBM                              */
           /*                                                                   */
           /* 5799-GBN                                                          */
           /*                                                                   */
           /* (C) Copyright IBM Corp. 1997, 1998 All Rights Reserved            */
           /*                                                                   */
           /* US Government Users Restricted Rights - Use, duplication or       */
           /* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.*/
           /*                                                                   */
           /*********************************************************************/
 2   1  0  DCL EXPARM1 FIXED BIN(31) INIT(5);
 3   1  0  DCL EXPARM2 FIXED BIN(31) INIT(2);
 4   1  0  DCL PARM2   FIXED BIN(31) INIT(2);
 5   1  0  DCL PLI01BM EXTERNAL ENTRY;        /*                               */
 6&  1  0  DO WHILE (EXPARM1 > 0);            /*  THIS DO LOOP EXECUTED 5 TIMES*/
 7:  1  1     EXPARM1 = EXPARM1 -1;           /*
 8:  1  1     CALL  PLI01BM(PARM2);           /* PLI01BM CALLED 5 TIMES
 9:  1  1  END;
10>  1  0  IF (EXPARM2 = 0) THEN             /* THIS BRANCH ALWAYS TAKEN      */
              CALL PROC2A(EXPARM2);          /* PROC2A NEVER  CALLED          */
11&  1  0  DO WHILE (EXPARM2 > 0);           /* DO LOOP EXECUTED TWICE        */
12:  1  1    EXPARM2 = EXPARM2 - 1;
13:  1  1  END;
14:  1  0  RETURN;

15   1  0  PROC2A: PROCEDURE(P1PARM1);        /* THIS PROCEDURE NEVER EXECU
16   2  0  DCL P1PARM1 FIXED BIN(31);
17¬  2  0  P1PARM1 = 10;
18¬  2  0  END PROC2A;
19   1  0  END PLI01AM;
```

Figure 24. Annotated PL/I Listing

```
 ACTIVE USINGS: NONE
 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                  HLASM R2.0  1997/11/14 09.55
                                    1 ********************************************************************      00000100
                                    2 *                                                                  *      00000200
                                    3 * LICENSED MATERIALS - PROPERTY OF IBM                            *      00000300
                                    4 *                                                                  *      00000400
                                    5 * 5799-GBN                                                       *      00000500
                                    6 *                                                                  *      00000600
                                    7 * (C) COPYRIGHT IBM CORP. 1997, 1998 ALL RIGHTS RESERVED         *      00000700
                                    8 *                                                                  *      00000800
                                    9 * US GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION OR     *      00000900
                                   10 * DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE CONTRACT WITH IBM     *      00001000
                                   11 * CORP.                                                           *      00001100
                                   12 *                                                                  *      00001200
                                   13 *                                                                  *      00001300
                                   14 ********************************************************************      00001400
                                   15 ********************************************************************      00001500
                                   16 *                                                                  *      00001600
                                   17 * HIGH LEVEL ASSEMBLER TEST.                                       *      00001701
                                   18 *                                                                  *      00001800
                                   19 ********************************************************************      00001900
000000                             20 TEST2    CSECT ,                                                 01S0001 00002000
                                    1 ********************************************************************      00000100
                                    2 *                                                                  *      00000200
                                    3 * LICENSED MATERIALS - PROPERTY OF IBM                            *      00000300
                                    4 *                                                                  *      00000400
                                    5 * 5799-GBN                                                       *      00000500
                                    6 *                                                                  *      00000600
                                    7 * (C) COPYRIGHT IBM CORP. 1997 ALL RIGHTS RESERVED               *      00000700
                                    8 *                                                                  *      00000800
                                    9 * US GOVERNMENT USERS RESTRICTED RIGHTS - USE, DUPLICATION OR     *      00000900
                                   10 * DISCLOSURE RESTRICTED BY GSA ADP SCHEDULE CONTRACT WITH IBM     *      00001000
                                   11 * CORP.                                                           *      00001100
                                   12 *                                                                  *      00001200
                                   13 *                                                                  *      00001300
                                   14 ********************************************************************      00001400
                                   15 ********************************************************************      00001500
                                   16 *                                                                  *      00001600
                                   17 * HIGH LEVEL ASSEMBLER TEST.                                       *      00001701
                                   18 *                                                                  *      00001800
                                   19 ********************************************************************      00001900
000000                             21 @MAINENT DS    0H                                            01S0001 00003000
                R:F  00000         22          USING *,@15                                         01S0001 00004000
000000 47F0 F016            00016  23%         B     @PROLOG                                       01S0001 00005000
000004 10                          24@         DC    AL1(16)                                       01S0001 00006000
000005 E3C5E2E3F2404040            25@         DC    C'TEST2     97.295'                           01S0001 00007000
                                   26          DROP  @15                                                   00008000
000015 00
000016 90EC D00C            0000C  27:@PROLOG  STM   @14,@12,12(@13)                               01S0001 00009000
00001A 18CF                        28:         LR    @12,@15                                       01S0001 00010000
                            00000  29 @PSTART  EQU   TEST2                                       01S0001 00011000
                R:C  00000         30          USING @PSTART,@12                                   01S0001 00012000
00001C 50D0 C0B0            000B0  31:         ST    @13,@SA00001+4                                01S0001 00013000
000020 41E0 C0AC            000AC  32:         LA    @14,@SA00001                                  01S0001 00014000
000024 50E0 D008            00008  33:         ST    @14,8(,@13)                                   01S0001 00015000
000028 18DE                        34:         LR    @13,@14                                       01S0001 00016000
                                   35 *   DO WHILE(EXPARM1>0);           /* THIS DO LOOP EXECUTED 5 TIMES */ 00017000
00002A 47F0 C042            00042  36%         B     @DE00006                                      01S0006 00018000
00002E                             37 @DL00006 DS    0H                                            01S0007 00019000
                                   38 *     EXPARM1 = EXPARM1 - 1;        /*                          */ 00020000
00002E 5810 C100            00100  39:         L     @01,EXPARM1                                   01S0007 00021000
000032 0610                        40:         BCTR  @01,0                                         01S0007 00022000
000034 5010 C100            00100  41:         ST    @01,EXPARM1                                   01S0007 00023000
                                   42 *     CALL TEST2B(PARM2);           /* TEST2B CALLED 5 TIMES        */ 00024000
000038 58F0 C0F8            000F8  43:         L     @15,@CV00063                                  01S0008 00025000
00003C 4110 C0A4            000A4  44:         LA    @01,@AL00002                                  01S0008 00026000
000040 05EF                        45%         BALR  @14,@15                                       01S0008 00027000
                                   46 *   END;                                                      01S0009 00028000
```

Figure 25 (Part 1 of 3). Annotated ASM Listing

```
000042 5800 C100            00100    47:@DE00006 L     @00,EXPARM1                                    01S0009 00029000
000046 1200                          48:         LTR   @00,@00                                        01S0009 00030000
000048 4720 C02E            0002E    49&         BP    @DL00006                                       01S0009 00031000
                                     50 *   IF (EXPARM2 = 0) THEN          /* THIS BRANCH ALWAYS TAKEN      */ 00032000
00004C 5810 C104            00104    51:         L     @01,EXPARM2                                    01S0010 00033000
000050 1211                          52:         LTR   @01,@01                                        01S0010 00034000
000052 4770 C06C            0006C    53>         BNZ   @RF00010                                       01S0010 00035000
000056 4110 C0A8            000A8    55¬         LA    @01,@AL00003                                   01S0011 00037000
00005A 45E0 C086            00086    56¬         BAL   @14,PROC1                                      01S0011 00038000
                                     57 *   DO WHILE(EXPARM2>0);           /* DO LOOP EXECUTED TWICE        */ 00039000
00005E 47F0 C06C            0006C    58¬         B     @DE00012                                       01S0012 00040000
000062                               59 @DL00012 DS    0H                                             01S0013 00041000
                                     60 *     EXPARM2 = EXPARM2 - 1;                                 01S0013 00042000
000062 5820 C104            00104    61:         L     @02,EXPARM2                                    01S0013 00043000
000066 0620                          62:         BCTR  @02,0                                          01S0013 00044000
000068 5020 C104            00104    63:         ST    @02,EXPARM2                                    01S0013 00045000
                                     64 *   END;                                                   01S0014 00046000
00006C 5830 C104            00104    65:@DE00012 L     @03,EXPARM2                                    01S0014 00047000
000070 1233                          66:         LTR   @03,@03                                        01S0014 00048000
000072 4720 C062            00062    67&         BP    @DL00012                                       01S0014 00049000
                                     68 *   RETURN CODE(0);                                        01S0015 00050000
000076 1FFF                          69:         SLR   @15,@15                                        01S0015 00051000
000078 58D0 D004            00004    70:         L     @13,4(,@13)                                    01S0015 00052000
00007C 58E0 D00C            0000C    71:         L     @14,12(,@13)                                   01S0015 00053000
000080 980C D014            00014    72:         LM    @00,@12,20(@13)                                01S0015 00054000
000084 07FE                          73%         BR    @14                                            01S0015 00055000
                                     74 *   END TEST2;                                             01S0020 00056000
                                     75 *PROC1:                                                     01S0016 00057000
                                     76 *   PROCEDURE(P1PARM1);            /* THIS PROCEDURE NEVER EXECUTED */ 00058000
000086 90EC D00C            0000C    77¬PROC1    STM   @14,@12,12(@13)                                01S0016 00059000
00008A D203 C0F4 1000 000F4 00000    78¬         MVC   @PC00002(4),0(@01)                             01S0016 00060000
                                     79 *   P1PARM1 = 10;                                          01S0018 00061000
000090 5820 C0F4            000F4    80¬         L     @02,@PA00064                                   01S0018 00062000
000094 4130 000A            0000A    81¬         LA    @03,10                                         01S0018 00063000
000098 5030 2000            00000    82¬         ST    @03,P1PARM1(,@02)                              01S0018 00064000
                                     83 *   END PROC1;                                             01S0019 00065000
00009C                               84 @EL00002 DS    0H                                             01S0019 00066000
00009C                               85 @EF00002 DS    0H                                             01S0019 00067000
00009C 98EC D00C            0000C    86¬@ER00002 LM    @14,@12,12(@13)                                01S0019 00068000
0000A0 07FE                          87¬         BR    @14                                            01S0019 00069000
0000A2                               88 @DATA    DS    0H                                                     00070000
0000A4                               89          DS    0F                                                     00071000
0000A4                               90 @AL00002 DS    0A                                                     00072000
0000A4 00000108                      91@         DC    A(PARM2)                                               00073000
0000A8                               92 @AL00003 DS    0A                                                     00074000
0000A8 00000104                      93@         DC    A(EXPARM2)                                             00075000
0000AC                               94          DS    0F                                                     00076000
0000AC                               95@@SA00001 DS    18F                                                    00077000
0000F4                               96@@PC00002 DS    1F                                                     00078000
0000F8                               97          DS    0F                                                     00079000
0000F8 00000000                      98@@CV00063 DC    V(TEST2B)                                              00080000
000100                               99          LTORG                                                        00081000
000100                              100          DS    0D                                                     00082000
000100 00000005                     101@EXPARM1  DC    F'5'                                                   00083000
000104 00000002                     102@EXPARM2  DC    F'2'                                                   00084000
000108 00000002                     103@PARM2    DC    F'2'                                                   00085000
                            00000   104 @DYNSIZE EQU   0                                                    00086000
                            00000   105 @00      EQU   0                                                    00087000
                            00001   106 @01      EQU   1                                                    00088000
                            00002   107 @02      EQU   2                                                    00089000
                            00003   108 @03      EQU   3                                                    00090000
                            00004   109 @04      EQU   4                                                    00091000
```

Figure 25 (Part 2 of 3). Annotated ASM Listing

```
                    00005   110 @05      EQU   5                                  00092000
                    00006   111 @06      EQU   6                                  00093000
                    00007   112 @07      EQU   7                                  00094000
                    00008   113 @08      EQU   8                                  00095000
                    00009   114 @09      EQU   9                                  00096000
                    0000A   115 @10      EQU   10                                 00097000
                    0000B   116 @11      EQU   11                                 00098000
                    0000C   117 @12      EQU   12                                 00099000
                    0000D   118 @13      EQU   13                                 00100000
                    0000E   119 @14      EQU   14                                 00101000
                    0000F   120 @15      EQU   15                                 00102000
                    00000   121 P1PARM1  EQU   0,4,C'F'                           00103000
                    000F4   122 @PA00064 EQU   @PC00002,4,C'F'                    00104000
                    0006C   123 @RF00010 EQU   @DE00012                           00105000
000110                      124          DS    0D                                 00106000
                    00110   125 @ENDDATA EQU   *                                  00107000
                    00110   126 @MODLEN  EQU   @ENDDATA-TEST2                     00108000
                            127          END   ,                                  00109000
```

Figure 25 (Part 3 of 3). Annotated ASM Listing

Selecting Specific Listings to Annotate

You do not always have to produce annotated listings. You may create a Summary report first, and then, based on the summary, decide to produce annotated listings on only a few selected program areas.

To select the specific listings you want to annotate after completing a test case run:

1. Edit the CACTL, leaving in the names of the listings you want to annotate:
   a. Select option `1` from the `ATC Primary Option Menu`, then select option `1` from the `Coverage, Distillation, and Unit Test Assistant` panel.
   b. On the `Work with the CA/DA/UTA Control File` panel, select option `1`, which displays an ISPF edit session to allow you to modify the control file. (You may also edit the control file directly, 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. See “Creating Summary and Annotated Listing Report JCL Using the Panels” on page 91 for details.
3. Submit the report JCL:

   Select option `3` from the `Create JCL for Summary and Annotation Report` panel.

   CA produces an Annotated Listing for each listing file specified in the CACTL.

Reducing the Size of Annotated Listings

To save paper when printing annotated listings, you may want to set the USEROPT parameter in your defaults to U (display only unexecuted code) rather than A (display all code). The report then lists only those lines that were not executed or that had conditional branches that did not go both ways.

Displaying Execution Counts in an Annotated Listing

The number of times each statement was executed can be displayed in an annotated listing. To do so, change the Debug Mode and Frequency Count Mode flags in your ATC defaults before generating the JCL for the SETUP step.

To change your ATC defaults:

1. Select option 0 from the `ATC Primary Option Menu`.

   The `Manipulate ATC Defaults` panel is displayed.

2. Select option 1.

   The `ATC Defaults` panel is displayed.

3. In the Setup Defaults area of the panel, change `Debug Mode` to YES and `Frequency Count Mode` to YES.

Every time SETUP JCL is created, these flags will be set. You can also change these flags in SETUP JCL that has already been created. This may be simpler than changing the defaults (and then changing them back) if you want to get execution counts for just one test case. To identify these flags in the parameters passed to the SETUP program, see the comments in the created SETUP JCL.

When Debug Mode flag is set to YES, breakpoints are left in storage for the entire test run instead of being removed after their first execution. Each time the breakpoint is executed, the count field is incremented.

When the Frequency Count Mode flag is YES, the execution counts are saved in the BRKOUT file of coverage results. The annotated listing report will display them on the right hand side of the listing, as shown in Figure 26:

```
000060 &                 PERFORM WITH TEST BEFORE UNTIL TAPARM1 = 0       30      >0006 0000 <
000061 :    1              SUBTRACT 1 FROM TAPARM1                       30      >0005 0000 <
000062 :    1              CALL 'COB01BM'                                85      >0005 0000 <
000063                   END-PERFORM
000064
000065              * THIS IF ALWAYS FALSE
000066 >                 IF TAPARM2 = 0                                   31      >0001 0000 <
000067 [    1              PERFORM PROCA                                 77      >0000 0000 <
000068                   END-IF
000069
000070              * THIS PERFORM EXECUTED
000071 &                 PERFORM WITH TEST BEFORE UNTIL TAPARM2 = 0       31      >0003 0000 <
000072 :    1              SUBTRACT 1 FROM TAPARM2                       31      >0002 0000 <
000073                   END-PERFORM
000074 :                 STOP RUN                                                 >0001 0000 <
000075                   .
```

Figure 26. Annotated Listing with Execution Counts

The execution counts for each statement are on the right between the right arrow (>) and the left arrow (<). For example, statement 60 was executed six times and statement 61 was executed five times. The other field between the arrows (always zeros in this example) is used to display the first key number that executed this statement when Distillation Assistant is used. For more information about key numbers, see “Recording Which Keys Execute a Statement” on page 142.

Note: Enabling Debug Mode and Frequency Count Mode may significantly degrade the performance of the monitored program.

Differences in CA Reports When DA and UTA Are Used

When DA or UTA is used with CA, extra breakpoints are inserted. Some of these extra breakpoints can be on conditional branch instructions, which will alter the conditional branch coverage in the summary. For example, if a test case were set up to read variables with UTA as well as to measure code coverage, its conditional branch coverage could be as follows:

```
| BRANCHES:          |
|  CPATH   TAKEN   %  |
----------------------
      18       9  50.0
----------------------
      18       9  50.0
```

Figure 27. Sample Conditional Branch Coverage with UTA

If you run the sample test case with UTA disabled, the extra breakpoints for reading variables are not inserted, and the conditional branch coverage is as follows:

```
| BRANCHES:          |
|  CPATH   TAKEN   %  |
----------------------
       2       2 100.0
----------------------
       2       2 100.0
```

Figure 28. Sample Conditional Branch Coverage without UTA

Note: Statement coverage is not affected if you enable UTA and DA.

The annotated listing will show the added conditional branches. For example, a test case with UTA enabled produced the following in the annotated listing:

```
000089      * OPERATIONS ON A DATE
000090 VV¬    COMPUTE INTEGER-DATE = FUNCTION INTEGER-OF-DATE(CURR-DATE)
```

Figure 29. Sample Annotated Listing with UTA

Because variables are read on statement 90, any conditional branches in the statement are monitored for coverage. Without UTA, no conditional branches are monitored on assignment statements, as shown in the following:

```
000089      * OPERATIONS ON A DATE
000090 :      COMPUTE INTEGER-DATE = FUNCTION INTEGER-OF-DATE(CURR-DATE)
```

Figure 30. Sample Annotated Listing without UTA

Changes in Annotation Symbols with Performance Mode

When the Performance Mode (described at “Using the Performance Mode to Reduce Monitor Overhead” on page 253) is used, the breakpoints for conditional branches are not kept in storage, and the conditional branch coverage data is inaccurate. For reports produced from test runs when the Performance Mode is enabled during SETUP, the following change to annotation occurs.

Conditional statements (IF, PERFORM(COBOL), DO WHILE(PL/I), and others) that were annotated with one conditional annotation symbol (>,V,&) per conditional branch within the statement, are now annotated with one : (a colon), if the statement is executed, or one ¬ (a not symbol), if the statement is not executed.

Parameters for the SUMMARY and REPORT Programs

This section describes the input parameters specified as the first character of the PARM field on the EXEC JCL statement for the SUMMARY and REPORT programs.

SUMMARY

The SUMMARY program produces a summary report from test case results. Its parameter is built automatically by the ISPF dialog. The parameter and its values are:

Internal/external

- **I** Summary with each PA listed separately.
- **E** Combine all PAs into one entry per object module.

PL/X Inline/NoInline (Optional unless Target specified)

- **I** Include PL/X inlined code.
- **N** Do not include PL/X inlined code.

Target/NoTarget (Optional)

- **Y** Indicates targeted summary.
- **N** Indicates normal summary.

Note: This parameter is ignored for assembler.

REPORT

The REPORT program produces annotated source listings from test case results. Its parameters are built automatically by the ISPF dialog. The types of parameters and their values are:

User code to display This flag indicates which part of the user code is displayed in the report.

- **A** All user code displayed.
- **U** Only unexecuted user code displayed.

Listing Type

- **B** COBOL listing.
- **P** PL/I listing.
- **A** assembler listing.

Annotation characters Optional list of annotation characters. The default list is as follows:
¬ **:** > **V % @** and **&**
For a description of each of these default characters, see "Annotated Listing Coverage Report" on page 95.

Printing Reports

You can reduce the size of annotated listings by printing only the lines with unexecuted code. To do this, specify `U` instead of `A` as the User Options variable in the Coverage Report Defaults section of the `ATC Defaults` panel, shown in Figure 4 on page 29.

Targeted Summary Reports

The targeted summary report is similar to a summary report except that it is based on a specific subset of statements. You can select statements by specifying the statement numbers directly, or by specifying that all statements that reference given COBOL or PL/I variables are of interest. You can also specify a range of statements to target.

The targeted summary report includes a header section containing a stylized version of the target control cards to generate the report. This provides an easy means for identifying the control cards used to generate the report in case you choose to produce multiple different Targeted Summary Reports from a single coverage run.

The other difference between the targeted summary report and the summary report is in the unexecuted code section. The targeted summary report lists all targeted unexecuted statements rather than the starting and ending statement numbers of unexecuted code segments. For a description of the summary format, see "Summary Coverage Report" on page 83.

To create a targeted summary report, you must supply:

- Breakpoint table (BRKTAB) and breakout (BRKOUT) data sets from a coverage run
- A target control data set
- Complier listings for the programs of interest

The target control cards, which specify the statements to be targeted, can be included by the user in the same CA control file used for the coverage run, or can be in a separate target control file. The SAA post processor can also be used to produce a template target control file based on an SAA comparison report, which targets all statements containing variables which appeared on changed lines. For details, see "SAA Postprocessor Outputs" on page 302.

Target Control File

A target control file is used to specify the statements that are of interest in the targeted summary report. It must be a sequential file or a member of a partitioned data set, and can have any DCB attributes. It is generally free-form and subject to the following requirements:

- Statements are free form (not column dependent)
- An asterisk in column 1 indicates a comment.
- The characters // (two consecutive slashes) indicate that the rest of the line following the two slashes is a comment.
- Lines containing nothing but blanks are ignored.
- Keywords and operands may be coded in any combination of upper and lower case.
- Operands may appear in any order
- Operands must be separated by a comma
- One or more blanks can appear between keywords and the corresponding equal sign as well as between the equal sign and the operand and between 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[25] 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 may be continued, if desired, by interrupting the line after a comma and continuing the statement on the next line.
- Labels, if present, are specified preceding the statement name and must be immediately followed by a colon. Labels cannot contain embedded blanks, commas, parentheses, or equal signs.
- Labels specified on COBOL, PL/I, and PLI statements cannot be repeated on any of those statements. Likewise, labels on SCOPE statements cannot be repeated on another SCOPE statement and labels on TARGETVAR statements cannot be repeated on another TARGETVAR statement.
- Operands shown in the syntax diagrams as being enclosed in parentheses, need not be enclosed in these parentheses if the operand contains no embedded blanks or commas.

[25] The DEFAULTS statement is described in "DEFAULTS Statement" on page 213.

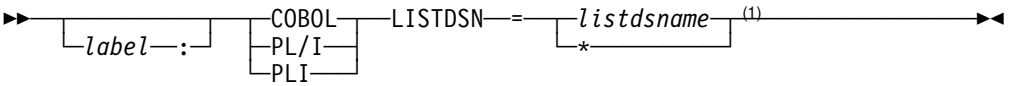
DBCS support for the control file statements is explained in Appendix C, “DBCS Support” on page 391.

The format of these statements is shown in the following sections.

Compilation Unit Definition (COBOL or PL/I) Statement

The COBOL and PL/I statements identify the compilation unit to which subsequent Target Control statements apply.

Note: The purpose and syntax of the COBOL and PL/I control statements is the same as that of the corresponding CA/DA/UTA control statement. However, not all information that is required for CA/DA/UTA is required in the Target Control statement.



Note:

[1] All other operands that can be specified on the corresponding CA control file statement (see “Control File Statement Syntax” on page 212) may be specified but are ignored.

where:

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

COBOL
indicates that this compilation unit is for a COBOL program.

PL/I
PLI
indicates that this compilation unit is for a PL/I program.

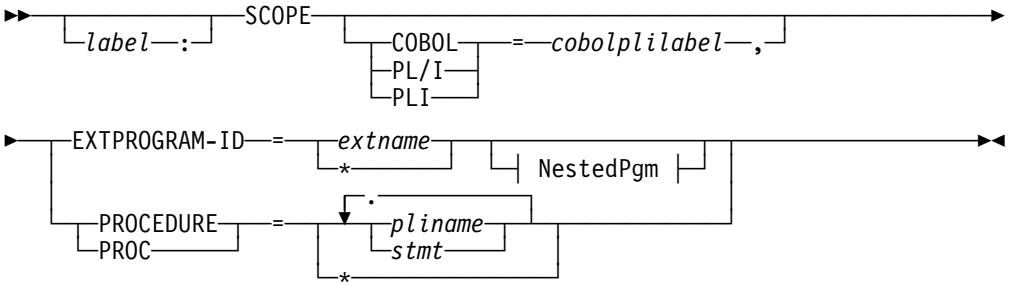
listdsname
the data set name of the data set which contains the compiler listing for this program.

***** indicates that all listings in the coverage run are of interest. (* cannot be specified in a COBOL or PL/I statement referenced by a TARGETVAR statement.)

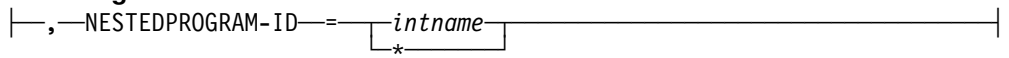
SCOPE Statement

The SCOPE statement identifies a name scope in a COBOL or PL/I program which contains variables of interest.

Note: The purpose and syntax of the SCOPE target control statement is the same as that of the SCOPE CA/DA/UTA control statement.



NestedPgm:



where:

cobolplilabel
: a label on the COBOL or PL/I statement that defines the compilation unit containing this scope. If this operand is omitted, the default is the previous COBOL or PL/I statement.

extname
: the COBOL program-id of the external COBOL program which contains items to be referenced.

*
: indicates that the search for the referenced items is to be done through all external COBOL programs in the specified listing.

intname
: the COBOL program-id of the internal (nested) COBOL program which contains items to be referenced. This operand should be specified only if the variables of interest are defined in a nested COBOL program. If this keyword is not specified, the variable is assumed to be defined in the external COBOL program.

*
: indicates that the search for the referenced items is to be done through all internal program-ids in the specified external COBOL program-id or through all procedures in the specified external PL/I procedure in the specified listing.

pliname
: the PL/I procedure or BEGIN block which contains items to be referenced. For PL/I internal procedures, the form PROC1.PROC2 must be used where PROC1 is the external procedure and PROC2 is an internal procedure contained in PROC1. Also, in the case of PL/I, named BEGIN blocks are considered to be equivalent to named procedures and are specified in exactly the same way. For unnamed BEGIN blocks, the statement number (*stmt*) where the BEGIN block is defined is used in place of the procedure name.

  For example, **PROCEDURE=P0.P1.B1.2451.P3** would specify an external procedure named P0 which contains an internal procedure or named BEGIN block named P1 which contains an internal procedure or named BEGIN block named B1 which contains an unnamed BEGIN block defined in statement 2451 which contains an internal procedure or named BEGIN block named P3.

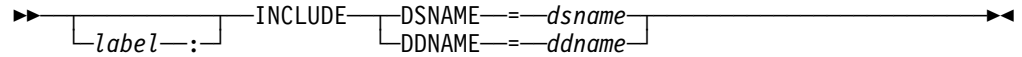
stmt
see the description of *pliname* above.

***** indicates that the search for the referenced items is to be done through all PL/I procedures and BEGIN blocks in the specified listing.

INCLUDE Statement

The INCLUDE statement can be used to include information from a CA/UTA/DA control file. When such a control file is included, all statements except DEFAULTS[25], COBOL, PL/I, and SCOPE are ignored.

Note: The INCLUDE statement allows labels defined on these statements to be referenced on subsequent TARGETVAR and TARGETSTMT control statements.



where:

dsname
specifies the data set name of the CA/DA/UTA control file to be included.

ddname
specifies a ddname which has been previously allocated to a CA/DA/UTA control file.

TARGETVAR Statement

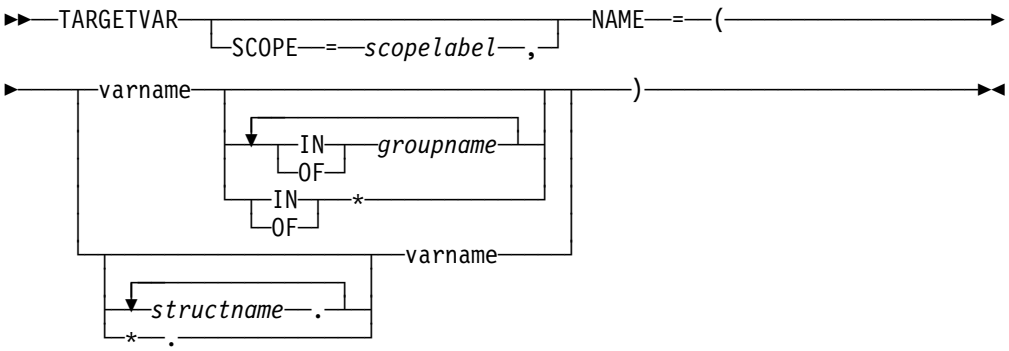
The TARGETVAR statement specifies that all COBOL or PL/I statements that reference a specific variable are of interest.

For COBOL variables, all statements that refer to the specified variable either directly (via the variable's name) or indirectly (via the name of a containing or contained group) are considered to be of interest.

For PL/I variables, only statements that refer to the specified variable directly (via the variable's name) are considered to be of interest.

The SCOPE specified for a PL/I variable must be the procedure or begin block in which the variable is defined (either explicitly or implicitly) although it may be referenced in other, contained blocks. This means that the scope for implicitly declared variables (other than parameters) must be defined as the external procedure.

The syntax of the TARGETVAR statement is:



where:

scopelabel — a label on a SCOPE statement that defines the scope containing the specified variable. If this operand is omitted, the default is the previous SCOPE statement.

varname — the name of the variable of interest.

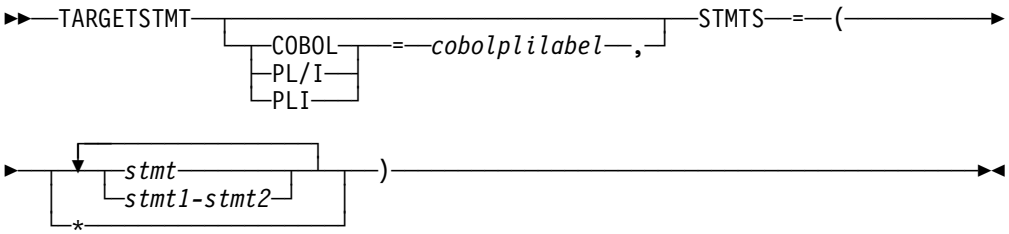
groupname — the name of a group containing the referenced variable.

* — specifies that the variable is an unqualified name. In this case, all occurrences of this name will be selected irrespective of the containing group name(s).

structname — the name of a structure containing the referenced variable.

TARGETSTMT Statement

The TARGETSTMT statement identifies specific statements of interest for Targeted Summary. It has the following syntax:



where:

cobolplilabel — a label on a COBOL or PL/I statement that defines the compile unit containing the specified statements. If this operand is omitted, the previous COBOL or PL/I statement will be used.

* — indicates that all statements in the source listing are to be selected.

stmt — one or more statement numbers from the executable statements in the source listing.

For COBOL listings, use the line number field shown in the compiler listing (under the LINEID header in the more recent compilers). For statements that span more than one line in the listing, use the first line number on which the statement occurs.

For PL/I listings, use the statement number field (STMT header) if the STMT compiler option is used or use the line number field (NUMBER header) if the NUMBER compiler option is used.

stmt1-stmt2 one or more ranges of statement numbers from the source listing. All statements within the range are selected. *stmt1* must not be greater than *stmt2*.

Example

```
        Cobol ListDsn=ATC.V1R5M0.Sample.CobolLst(Cob01M),
              FromObjDsn=ATC.V1R5M0.Sample.Obj,ObjMember=Cob01M,ToObjDsn=ATC.V1R5M0.Sample.Zapped.Obj
        Scope ExtProgram-Id=*
*
        TargetVar Name=(Date1 In MyFile In Record1)
        TargetStmt Stmts=(21,33)
```

Figure 31. Target Control File

Creating Targeted Summary Reports

To create a Targeted Summary report, select option `4` on the `Coverage Reports` panel. The `Create JCL for Targeted Summary Report` panel, shown in Figure 32, allows you to specify targeted summary report parameters.

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

1  Generate      Generate JCL from parameters
2  Edit          Edit JCL
3  Submit        Submit JCL
4  Foreground    Run Targeted Summary in the Foreground

Enter END to Terminate

Use Program Name for File Name  YES (Yes|No) Program Name  COB01M

Input Files:
  Control File Dsn. . . 'YOUNG.TEST.TARGCTL(COB01M)'
  Breakpoint Table Dsn. 'YOUNG.TEST.COB01M.BRKTAB'
  Breakout Dsn. . . . . 'YOUNG.TEST.COB01M.BRKOUT'

JCL Library and Member:
  JCL Dsn . . . . . 'YOUNG.TEST.JCL.CNTL(QCOB01M)'

Output Summary Type and File Name:
  Type. . . . . INTERNAL (Internal|External)
  Report Dsn  . . . . . 'YOUNG.TEST.COB01M.TARGREP'
    (* for default sysout class)
```

Figure 32. Create JCL for Targeted Summary Report

Generate
: Generate JCL from the parameters you have specified on the panel.

Edit
: Make changes to existing JCL.

Submit
: Submit for execution the JCL specified in the `JCL Dsn` field on this panel.

Foreground
: Run Targeted Coverage in the foreground using the parameters you have specified on the panel.

Use Program Name for File Name
: To construct all data set names from the default high-level qualifier, the specified program name, and the default low-level qualifier for each data set, enter `YES`.

  When you press Enter, the file names on the panel are changed automatically.

Program Name

Name to use to construct data set names if you enter `YES` in the `Use Program Name for File Name` field. Note that this 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.TEST.COB01M.BRKTAB'

- Partitioned data sets:

  `'proj_qual.file_type(program_name)'`

  For example: 'YOUNG.TEST.BRKTAB(COB01M)'

Control File Dsn

Name of the targeted coverage control data set.

Breakpoint Table Dsn

Name of the breakpoint (BRKTAB) data set created by the coverage run.

BreakOut Dsn

Name of the breakout (BRKOUT) data set created by the coverage run.

JCL Dsn

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

Note: 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 Qxxxxxxx, where xxxxxxx is the first seven characters of the program name.

Type

Type of targeted summary report to produce. An Internal summary lists totals for each PA separately. An External summary combines all PAs into one entry per object module.

Report Dsn

Name of the targeted coverage report data set. This data set will be created if it does not currently exist.

Using Coverage Assistant in a Large Project Environment

Typically, large projects involve 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 CA), and a testing interval that may last many days or weeks. This chapter explains how CA supports this type of complex testing environment.

Creating CA Files during Code Development

The creation of the listings and modified object modules used by CA can be incorporated into the coder's standard development procedure.

In this chapter:

- *Coder* refers to the code developer(s) developing multiple compilable object modules for a product.
- *Tester* refers to the person(s) running test cases on the product to obtain test case coverage data.
- *Module* refers to a separately compilable object module of the project that has a listing.

A flow diagram of test case coverage in a large project environment is shown in Figure 33 on page 116.

To create CA files during code development, complete the following steps:

1. Each coder should generate breakpoint data on object modules when they are ready to be tested. The breakpoint data could be created automatically by a CA step included in the coder's compile procedure or JCL. At that time, a breakpoint table would be created to match the object module. In the CA step, you should include a ZAPTXT program that uses the breakpoint data and modifies the object module by inserting the breakpoints.
2. The tester selects the object modules to be measured by CA (created in step 1) and links them with the remaining *unmodified* object modules to create the executable load module.
3. The tester enters the names of the modules to be measured by CA into the CACTL.
4. The tester starts a monitor session, runs test cases, and stops the monitor. When the monitor is stopped, it writes its output data to a results file.
5. The tester produces a report on selected modules from the test case coverage results.
6. The tester performs steps 2 through 4 for as many coders or test case coverage runs as desired.
7. For overall test case coverage, the tester combines results from each coder or test case coverage run into one report.

© Copyright IBM Corp. 1997, 1999

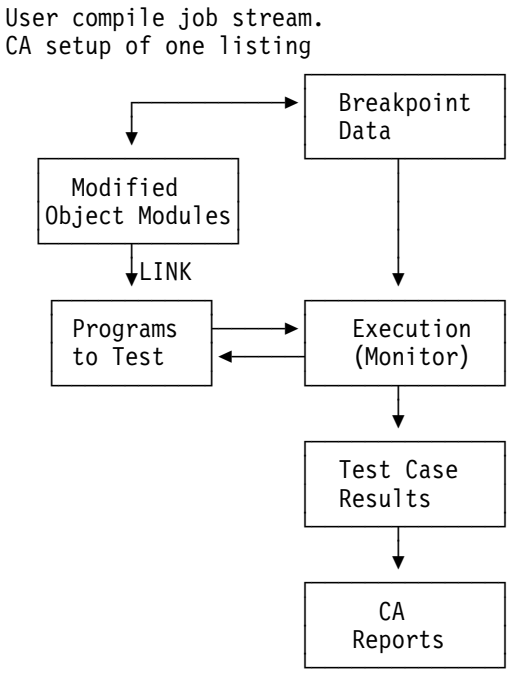


Figure 33. Using CA in a Large Project Environment—Flow Diagram

Breakpoint Data

You can generate breakpoint data for an object module every time you compile the module or for any number of listings at the same time. To update the breakpoint data for a module every time you compile, insert a CA Setup step into the coder's compile job stream.

When the tester is ready to do a test case coverage run, the tester edits the CACTL by entering the file names of each object module to be measured. When the monitor JCL is run, the breakpoint files need to be concatenated as DD statements for use by the monitor during the test case coverage run.

Test Case Coverage Results

If you stop a monitor session, it writes the test case coverage results for each module to a test case results (BRKOUT) file. To create a summary and annotated listings for selected modules, the tester puts their names in the CACTL. These names can be any modules that have been tested in any test coverage run, as long as the results are in a BRKOUT file.

Combining Test Case Coverage Results

To combine individual test case coverage runs into a report showing overall test case coverage, the tester executes the CA test case coverage combine program. For an example of combining test case results, see Figure 34 on page 117.

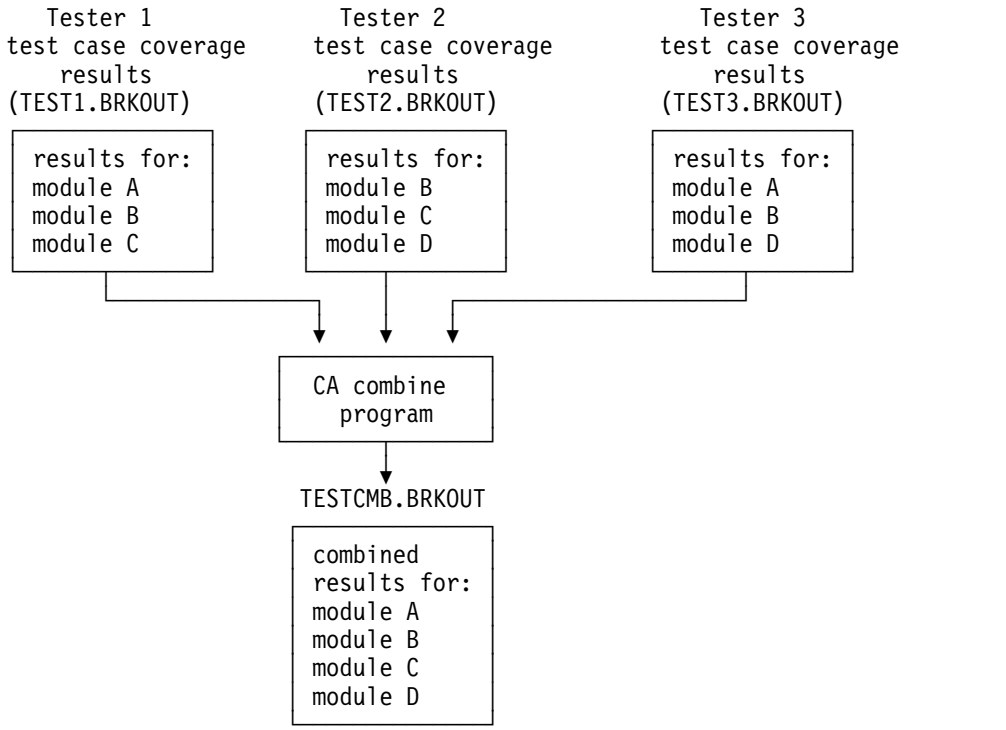


Figure 34. Combining Results of Multiple Testers—Flow Diagram

Three test case coverage runs 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. Run the CA combine program to combine the results from the three test case coverage runs into one file. The combined results show the overall test case coverage. A summary or annotated listing report can then be produced 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). If the BRKTABs for modules A through D exist in different files, you can concatenate them in the BRKTAB DD JCL statement in the report JCL.

Creating the Combine JCL Using the Panels

To combine test case coverage results using the panels:

1. Select option `1` from the `ATC Primary Option Menu` to display the `Coverage, Distillation and Unit Test Assistant` panel.
2. Select option `4` to display the `Coverage Reports` panel.
3. Select option `3` to display the `Create JCL for Combining Multiple Runs` panel, shown in Figure 35 on page 118.

```
-------------------- 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.TEST.CBCTL(COB06M)'

JCL Library and Member:
  JCL Dsn . . . . . . . 'YOUNG.TEST.JCL.CNTL(CCOB06M)'

Combined Breakout File:
  Breakout Dsn. . . . . 'YOUNG.TEST.COB06M.CMBOUT'
```

Figure 35. Create JCL for Combining Multiple Runs Panel

4. Select option `1` and change the values entered in each field as appropriate.

   The panel's options and fields are as follows:

   EditCtl
   Edit the combined control file.

   ResetCtl
   Reset the combined control file.

   Generate
   Generate JCL using the parameters you have specified on the panel.

   Edit
   Make changes to existing JCL.

   Submit
   Submit for execution the JCL specified in the `JCL Dsn` field on this panel.

   Use Program Name for File Name
   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, enter `YES`. Using the program name is the normal CA procedure.

   When you press Enter, the file names on the panel are changed automatically.

Program Name

Name to use for CA data sets if you enter `YES` in the `Use Program Name for File Name` field. Note that this 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.TEST.COB01M.BRKTAB'
- Partitioned data sets:

  `'proj_qual.file_type(program_name)'`

  For example: 'YOUNG.TEST.BRKTAB(COB01M)'

Combined Cntl Dsn

Name of the data set containing the list of breakout data sets you want to combine.

JCL Dsn

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

Note: If `Use Program Name for File Name` is set to YES, then the member name or program name qualifier of the data set will be C*xxxxxxx*, where *xxxxxxx* is the first seven characters of the program name.

Breakout Dsn

Name of the combined BRKOUT data set created by running the combine JCL.

5. The `EditCtl` option puts you into an ISPF edit session in which you list the data sets to be combined (see Figure 36 for an example). You must enter the complete data set name for each data set.

```
EDIT       YOUNG.TEST.CBCTL(COB06M)                        Columns 00001 00072
Command ===>                                                 Scroll ===> CSR
********************************** Top of Data ****************************
000001 * List the input files (PDS or SEQ) you wish to combine (one per line),
000002 * using JCL naming conventions.  All comment lines must start with '*'.
000003 YOUNG.TEST.COB06M1.BRKOUT                               <= Input DSN 1
000004 YOUNG.TEST.COB06M3.BRKOUT                               <= Input DSN 2
000005 YOUNG.TEST.COB06M5.BRKOUT                               <= Input DSN 3
000006 YOUNG.TEST.COB06M7.BRKOUT                               <= Input DSN 4
000007                                                         <= Input DSN 5
000008                                                         <= Input DSN 6
000009                                                         <= Input DSN 7
000010                                                         <= Input DSN 8
000011                                                         <= Input DSN 9
000012                                                         <= Input DSN 10
******************************** Bottom of Data ***************************
```

Figure 36. ISPF Edit Screen for Combine

Rules for Combining Results

Two coverage runs will be combined only if they were created using the same BRKTAB file. The following coverage runs will be combined:

1. Run setup, create BRKTAB file
2. Coverage run 1 uses BRKTAB file
3. Coverage run 2 uses same BRKTAB file

The following coverage runs will not be combined because the setup step was rerun:

1. Run setup, create BRKTAB1 file
2. Coverage run 1 uses BRKTAB1 file
3. Rerun setup, create new BRKTAB2 file
4. Coverage run 2 uses BRKTAB2 file

Sample Combine Test Case

A sample test case (COB06M) is provided in the samples shipped with ATC that illustrates combining test case results.

COB06M was executed several times with a different numerical (1-8) input parameter each time. Each number caused a different set of statements to be executed. The results of running COB06M under CA with each of the input parameters specified are in BRKOUT files *hi_lev_qual*.V1R5M0.SAMPLE.BRKOUT(COB06M*x*), where *x*=1 to 8.

For an example of JCL that runs COB06M, see *hi_lev_qual*.V1R5M0.SAMPLE.JCL(GCOB06M).

Running a report with one of these BRKOUT files shows only the coverage of that input number. Four of the eight BRKOUT files (inputs 1, 3, 5, and 7) were combined. The JCL to combine these BRKOUT files is in *hi_lev_qual*.V1R5M0.SAMPLE.JCL(CCOB06M).

Executing this JCL produces BRKOUT file *hi_lev_qual*.V1R5M0.SAMPLE.BRKOUT(COB06M). Executing the report program, whose JCL is in *hi_lev_qual*.V1R5M0.SAMPLE.JCL(RCOB06M), produces a combined report that shows the coverage of running COB06M with inputs 1, 3, 5, and 7. For an example combined report, see the *hi_lev_qual*.V1R5M0.SAMPLE.REPORT(COB06M) file.

Measuring Coverage for Individual Test Cases

Testers may want to keep coverage results on a test case basis. This allows them to run test cases for regression testing of 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** the tester issues either the CADATA or CASTOP command. When these commands are executed, the tester can select the file name of the BRKOUT file.

For example:

```
XTEST2        Start the monitor
TEST2 parm1   Run TEST2 for test case 1
CADATA TC1    Save the coverage results in file proj_qual.TC1.BRKOUT
CARESET       Reset statistics
TEST2 parm2   Run TEST2 for test case 2
CADATA TC2    Save the coverage results in file proj_qual.TC2.BRKOUT
CARESET       Reset statistics
TEST2 parm3   Run TEST2 for test case 3
CASTOP TC3    Save the coverage results in file proj_qual.TC3.BRKOUT
              and stop the monitor session
```

The BRKOUT files TC1, TC2, and TC3 contain coverage results for their respective test cases. Testers can run the CA report program on the BRKOUT files to obtain coverage results for the specific test cases. To obtain overall coverage, execute the Combine program on TC1, TC2, and TC3.

Using Distillation Assistant

© Copyright IBM Corp. 1997, 1999

Introduction

This chapter contains the following topics:

- What Is Distillation Assistant?
- What Does DA Require?
- How Does DA Work?
- Where Can You Get Further Details?

What Is Distillation Assistant?

Distillation is the reduction of a data set to the minimum size that provides the same test coverage as the complete data set. Distillation Assistant (DA) supports distillation of QSAM or VSAM input data sets that are processed by applications generated with the following IBM COBOL and PL/I compilers:

- IBM COBOL for OS/390 & VM 2.1 (plus Millennium Language Extensions [MLE])
- IBM COBOL for MVS & VM 1.2 (plus Millennium Language Extensions [MLE])
- IBM VS COBOL II Release 4.0
- IBM OS/VS COBOL Release 2.4
- IBM PL/I for MVS & VM 1.1.1 (plus Millennium Language Extensions [MLE])
- IBM OS PL/I Optimizing Compiler 2.3.0
- IBM PL/I Optimizing Compiler 1.5.1

While the program under test is running, DA records all logical keys (for data read from a specified input data set) that cause new coverage. When you have completed testing, you can run a DA program that makes a copy of your input data set, but includes only the records that have these keys. Further testing using this distilled input data set results in much faster test runs with coverage that is equivalent to that produced by using the complete input data set.

For example, your input master data set may contain thousands of employee or customer records. DA may distill the input master data set to several hundred records that caused new coverage during your original test run. Further testing time is greatly reduced by using the distilled input master data set.

The distillation process consists of two steps:

Step 1 Logical Distillation

This step consists of instrumenting your object code and executing the instrumented code under the DA monitor. As the instrumented code reads records from the specified input master data set, the monitor determines which “keys” in the input master data set caused new code coverage in the instrumented code. The list of these keys is then saved for the second step.

Step 2 Physical Distillation

This step consists of creating a new master data set by reading the list of keys produced in the first step and the input master data set. The new master data set consists of only those records in the input master data set whose logical key appears in the list of keys.

© Copyright IBM Corp. 1997, 1999

DA has the following characteristics:

- Low overhead

  For a test case run, DA typically adds very little to the execution time of the program. DA inserts user SVC instructions as breakpoints and then intercepts the interrupts.

- Panel-driven user interface

  You can use an ISPF panel driven interface to create JCL for executing DA programs.

- Simple, flexible control

  The control file used to define your input master data set provides a simple method of controlling DA operations.

What Does DA Require?

DA runs under MVS. Detailed MVS system resource requirements for DA are described in "DA and UTA Resources" on page 388. DA uses ISPF services to display dialogs and to produce the JCL to run the DA steps.

Input Master Data Set Restrictions

The input master data set can be any sequential or VSAM data set which contains logical[26] keys. However, note the following input master data set restrictions:

- If the data set is sequential, the RECFM may be any valid MVS RECFM except VS and VBS. Spanned records are not supported.
- VSAM data sets with either KSDS or ESDS organizations can be distilled. However, VSAM data sets that have alternate indexes will not have the corresponding alternate indexes built into the new master data set.
- Any type of VSAM data set that cannot be distilled directly, can be copied using the IDCAMS REPRO function to a sequential data set, which can be distilled and copied back to a VSAM data set.

In addition to the previously described restrictions, the logical keys:

- Cannot exceed 126 bytes in length
- Must appear at the same offset in each record
- Must be the same length in each record

[26] A logical key is simply a field which can be used to identify the record. These need not be defined as physical keys. For example, a VSAM ESDS or a sequential data set can be distilled as long as a field exists that can be logically used as a key.

Logical Distillation Requirements

DA requires, as part of its input, listings created by the application program compilers that it supports. These compilers offer options that allow you to include assembler statements, data maps, and data cross-references in the listings, all of which DA uses.

DA also requires the application program object modules as input. DA creates copies of these object modules with breakpoints inserted into them.

See "Execution" on page 130 for a description of how the DA authorized programs intercept breakpoints.

Notes:

1. **Logical distillation cannot be performed on more than one physical file at a time.** This means that if more than one FILE control statement is specified for a logical distillation run, all of the specified COBOL FDs or PL/I file constants must resolve to the same DDNAME.
2. UTA variable monitoring should not be done in conjunction with DA key gathering. If the two are performed at the same time, the variable reads are intermixed with the keys, and no process is provided for separating them.
3. PL/I distillation does not support the following constructs:
   - Stream I/O. Only record I/O is supported.
   - Files which have the ENVIRONMENT(TOTAL) attribute.
   - READ operations on file variables. Only READs of file constants are supported.
   - READ statements that specify the EVENT option.
4. COBOL distillation is not supported under CICS® for routines compiled with the OS/VS COBOL compiler. *Reentrant* COBOL routines are only supported for compilers that support the RENT compiler option.

How Does DA Work?

Running DA consists of the following steps. This list is an overview of the process. **Each activity is described in more detail in topics that follow in this chapter.**

Step 1 Setup

a. Compile the source code that you want to analyze, using the required compiler options.

b. Generate DA JCL using the DA ISPF dialog:

   1) Edit the DA control file.
   2) Create the setup JCL.
   3) Create the monitor JCL.
   4) Create the distillation JCL.

c. If you instrumented object modules, edit your program's link-edit JCL to pick up the modified object modules, which are created by the Setup step.

d. Edit your program's GO JCL (or program invocation) to point to the new load module that will be created when you run the JCL created in step 1c.

Step 2 Execution

a. Run the Setup JCL (created at step 1b2).

b. Run the link-edit JCL (created at step 1c).

c. Run the monitor JCL to start a monitor session (created at step 1b3).

d. Run your application using the load module(s) created in step 2b.

e. Stop the monitor session (using the CASTOP command).

Step 3 Distillation

a. Run the distillation JCL (created at 1b4 on page 127).

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.

Figure 37 on page 129 shows a diagram of the entire process.

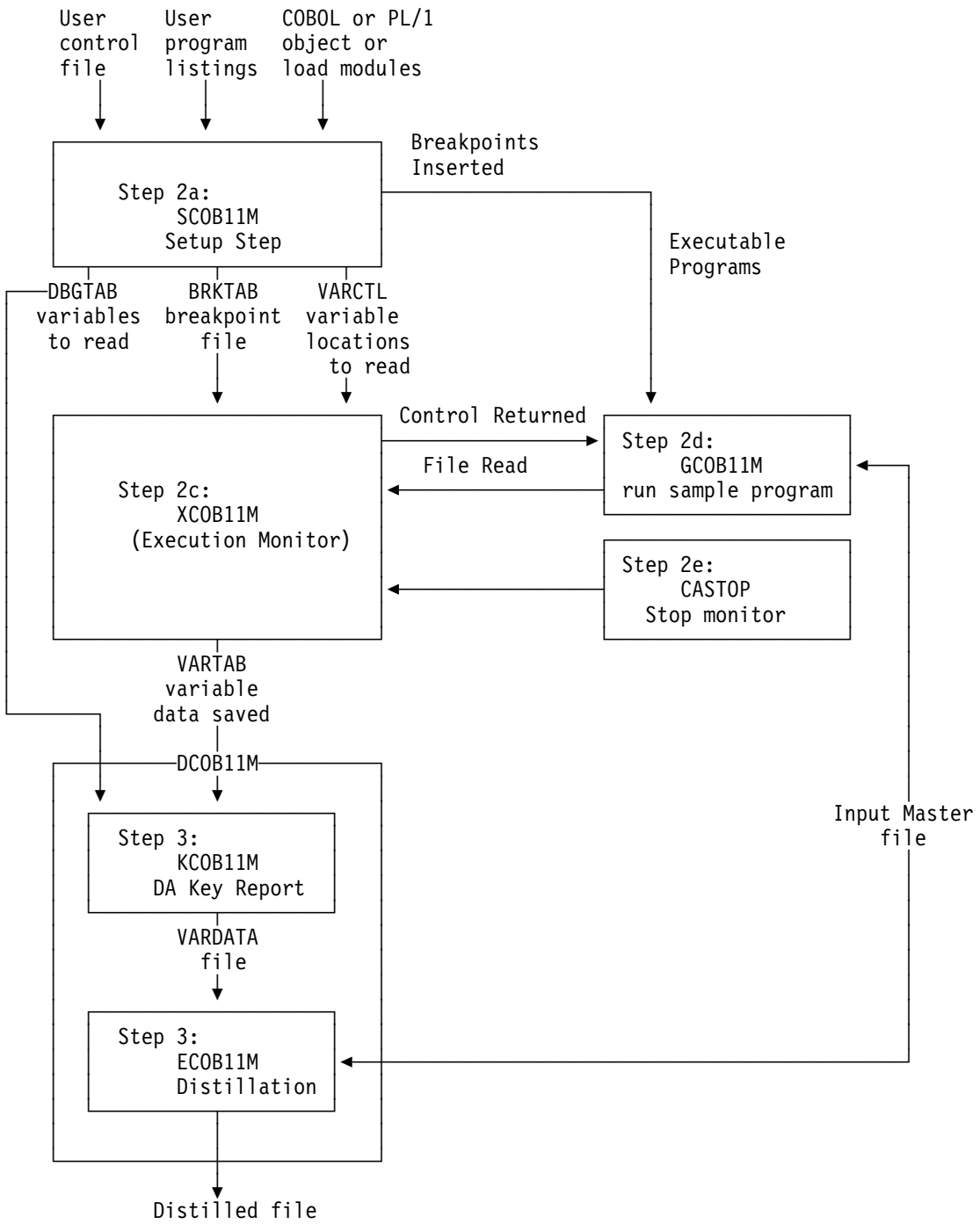


Figure 37. DA—Flow Diagram

Setup

DA Setup analyzes assembler statements and cross-reference information included in the compiler output listings to determine where to insert breakpoints into disk-resident copies of the object modules. It then inserts the breakpoints.

Setup runs in MVS. To run it, you need:

1. Compiler listings of the object modules
2. The object modules or load modules you want to test
3. The user control file listing the input data set to distill

The Setup step produces:

1. Modified test programs (object modules or load modules) containing breakpoints
2. A file of breakpoint-related information (called *BRKTAB* in this User's Guide) required for the monitor program in the Execution step.
3. A file of data for the data set to be monitored (called *DBGTAB* in this User's Guide).
4. A file of locations where the input data set is read by the monitor (called *VARCTL* in this User's Guide).

Execution

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

Then start a DA monitor session and run your test case programs. As the selected breakpoints are encountered, the monitor gains control, logs a key if it caused new coverage, then returns control to your program. After your test cases have completed, use CASTOP to stop the monitor session. It writes the results (the list of keys that caused new coverage) to a file called VARTAB.

The monitor inserts reserved supervisor call (SVC) instructions as breakpoints and is given control by MVS when these SVC instructions are executed in a program. Using SVCs as breakpoints is the architected way to receive control from MVS, and requires no modification to MVS. This technique is called user SVCs.

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.

Physical Distillation

DA physical distillation has two steps:

1. Format the list of keys that caused new coverage (the VARTAB file created by execution step).
2. Distill the input master data set.

The key list from step 1 is used in step 2 to make a distilled copy of your input master data set.

You can create JCL to run these two steps consecutively, or you can create JCL to run each step separately. If you run the steps separately, you can inspect and edit the key list to produce a customized distillation data set.

For example, you may want to include keys in a distilled data set even though they did not cause new coverage, or you may want to remove keys that caused new coverage, but are not important to your testing.

Where Can You Get Further Details?

Refer to the following sections for additional information.

For information about...	See...
Samples of DA test case coverage, including sample reports	“Distillation Assistant Samples” on page 133
Editing the CACTL file that contains the names of the listing data sets	“Editing the Distillation Assistant Control File” on page 145
Setting up the table of breakpoints from the listings	“CA, DA, and UTA Setup” on page 231
Starting the DA monitor session and running test cases on your programs	“Monitor Execution” on page 245
Commands that control the DA monitor program	“Monitor Commands” on page 255
Distilling your data	“Physical Distillation” on page 149
System resources needed by DA	“DA and UTA Resources” on page 388

Distillation Assistant Samples

This chapter describes a sample distillation using an example provided with the ATC package.

Distillation Assistant (DA) performs the distillation based on information that you specify in the control file.

This sample distills data read by the COBOL and PL/I test cases:

- COB11M (COBOL for IBM MVS & VM 1.2 sample)[27]
- COB112 (IBM VS COBOL II Release 4.0 sample)
- COB11O (IBM OS/VS COBOL Release 2.4 sample)
- PLI11M (IBM PL/I for MVS & VM 1.1.1 sample)
- PLI112 (IBM OS PL/I Optimizing Compiler 2.3.0 sample)
- PLI111 (IBM PL/I Optimizing Compiler 1.5.1 sample)

A flow diagram of the steps required to run these samples is shown in Figure 38 on page 134. The names in the steps are the member names of the JCL executed for the step. For COBOL, the steps are SCOB*nnx*, where *nn* is the test case number and *x* is *M* for COBOL for MVS & VM, *2* for VS COBOL II, or *O* for OS/VS COBOL. The PL/I steps are SPLI*nnx*. For PL/I the compiler identification is *M* for PL/I for MVS & VM, *2* for PL/I 2.3.0, or *1* for PL/I 1.5.1. In the flow diagram, *ccc* is the compiler identification: *COB* for COBOL (for example, SCOB11M) or *PLI* for PL/I (for example, SPLI11M).

The following DA samples use the ATC ISPF dialog to create the JCL to run the DA steps. The ATC ISPF dialog is provided as an aid in creating the JCL. Once the JCL is created for a test environment, it does not have to be recreated from the dialog. In a typical user test environment, the creation of the JCL can be incorporated into the user's procedures. You do not have to use the ISPF dialog to use DA.

[27] You can also test the DA installation for the COBOL for OS/390 & VM compiler by copying the JCL members and making minor edits to the compiler, link-edit, and runtime library names.

© Copyright IBM Corp. 1997, 1999

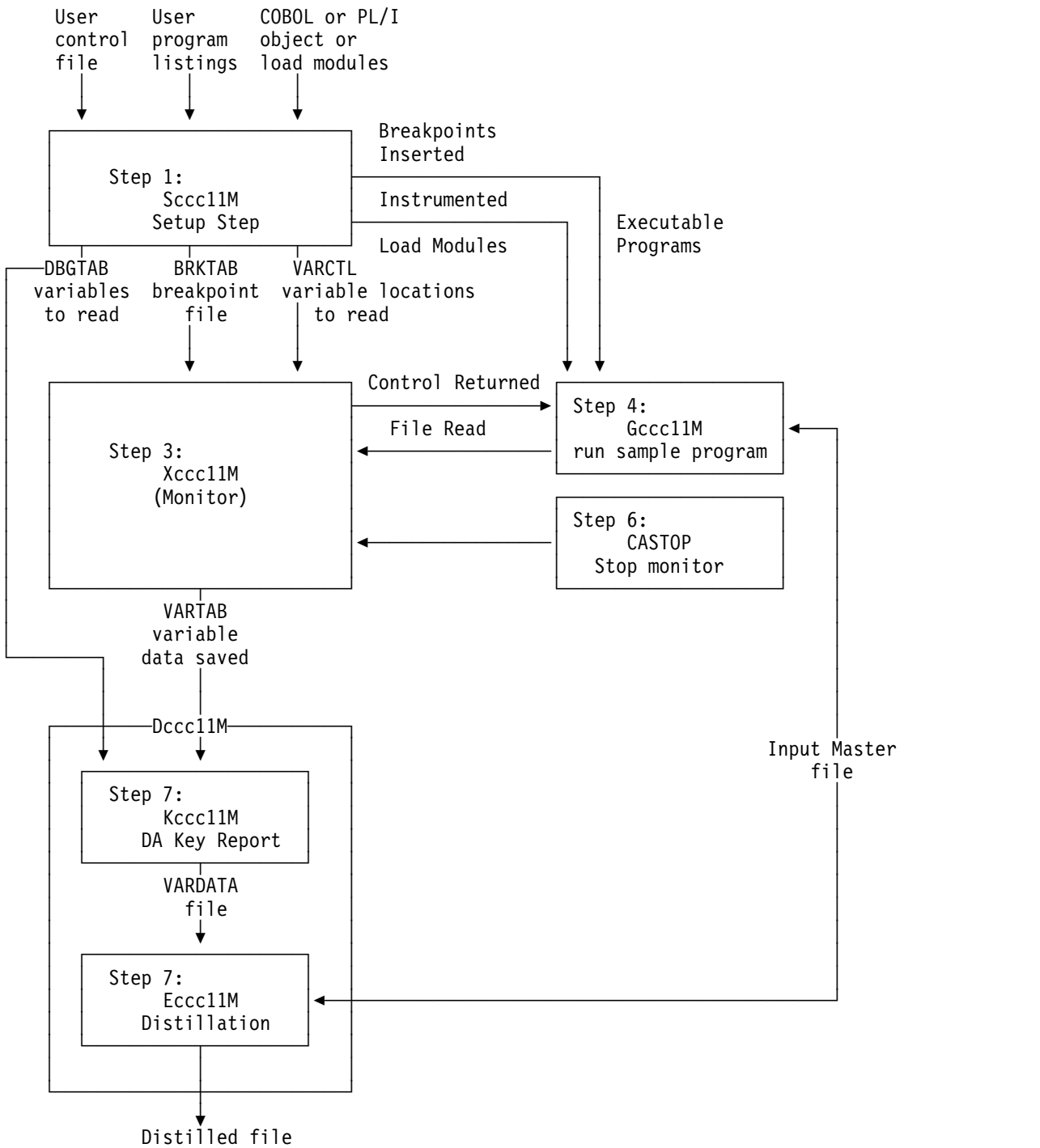


Figure 38. Sample Run—Flow Diagram

COB11x and PLI11x Test Cases

The COB11x (for COBOL) and PLI11x (for PL/I) test cases are examples of reading a file from one compile unit. The control files are shown in Figure 41 on page 138 and Figure 42 on page 138, the input master data set is shown in Figure 39 on page 136, and the new master data set is shown in Figure 40 on page 136. In the following descriptions, the test cases are referred to as *ccc*11x, where *ccc* is the compiler (*COB* for COBOL or *PLI* for PL/I).

Breakpoints are inserted into the object modules during Setup. These breakpoints are positioned so that the reads of the specified file can be monitored when the breakpoint interrupt occurs. When you are ready to test your program, you link the object modules that have been modified with the breakpoints.

To distill the input master data set read by *ccc*11*x*, perform the following steps. **Steps 2a through 6 are described in more detail in topics that follow in this chapter.**

1. Compile the source you want to test. This produces listings that include the assembler statements needed by DA. (This has already been done for the examples. For COBOL, the listings are in *hi_lev_qual*.V1R5M0.SAMPLE.COBOLST for the COBOL for MVS & VM and VS COBOL II compilers, and in *hi_lev_qual*.V1R5M0.SAMPLE.COSVSLST for the OS/VS compiler. For PL/I, the listings are in *hi_lev_qual*.V1R5M0.SAMPLE.PLILST).

   Make sure to use the compiler options specified in "Setup" on page 231.

2. Start the ATC ISPF dialog by entering the following from ISPF option 6:

   ```
   EX 'hi_lev_qual.V1R5M0.REXX(ATSTART)'
   ```

   The `ATC Primary Option Menu` is displayed.

   a. Edit the DA control file.

      Verify that the control file includes the listings of the object modules you want to test and information about the file that you want to distill.

   b. Create the setup JCL.

      Create the JCL that enables the Setup job to produce files containing breakpoint data, file read data, and instrumented programs. You can instrument either object modules or load modules.

   c. Create the monitor JCL.

      Create the JCL to start a monitor session.

   d. Create the JCL to perform the physical distillation.

      Create the JCL to perform the physical distillation after the *ccc*11*x* test case has executed.

3. End the ATC ISPF dialog by pressing the End key (PF3) on the `ATC Primary Option Menu`.

4. If you instrumented object modules, create the JCL to link the modified object modules.

   After the setup step, and before starting the monitor session, you must link the modified object modules into an executable program you can test. Specify the

library that will contain the modified object modules and the library that will contain the modified load module.

5. Create the JCL to run the GO step.

   Create the JCL to run your program. Specify the same modified load module as in step 4 on page 135.

6. Execute the JCL.

   Execute the created JCL files for *ccc*11*x* in the correct order. (This order is shown in "Execute the JCL" on page 140.)

Figures 39 and Figure 40 show portions of the input master data set and new master data set for this example.

```
%%Lincoln's Ge4a+++++++++++++++0402ttysburg Address, given November 19, 1863
%%Four score a8b+++++++++++++++5154nd seven years ago, our fathers brought fo
%%dedicated to9c+++++++++++++++1108 the proposition that all men are created
%%Now we are e5d+++++++++++++++8036ngaged in a great civil war. . .testing wh
%%can long end1e+++++++++++++++6086ure.  We are met on a great battlefield of
%%We have come5f+++++++++++++++9096 to dedicate a portion of that field as a
%%It is altoge7g+++++++++++++++5008ther fitting and proper that we should do
%%But, in a la1h+++++++++++++++5706rger sense, we cannot dedicate. . .we cann
%%who struggle8i+++++++++++++++8587d here have consecrated it, far above our
%%what we say 3j+++++++++++++++5249here, but it can never forget what they di
%%It is for us2k+++++++++++++++4243 the living, rather, to be dedicated here
%%It is rather4l+++++++++++++++2039 for us to be here dedicated to the great
%%to that caus6m+++++++++++++++0068e for which they gave the last full measur
%%that this na2n+++++++++++++++3536tion, under God, shall have a new birth of
%%shall not pe9o+++++++++++++++4977rish from this earth.
%%Lincoln's Ge9p+++++++++++++++2638ttysburg Address, given November 19, 1863
%%Four score a1q+++++++++++++++0065nd seven years ago, our fathers brought fo
%%dedicated to3r+++++++++++++++2252 the proposition that all men are created
%%Now we are e7s+++++++++++++++3705ngaged in a great civil war. . .testing wh
%%can long end5t+++++++++++++++1680ure.  We are met on a great battlefield of
%%We have come6u+++++++++++++++4811 to dedicate a portion of that field as a
%%It is altoge3v+++++++++++++++7672ther fitting and proper that we should do
%%But, in a la8w+++++++++++++++0965rger sense, we cannot dedicate. . .we cann
%%who struggle1x+++++++++++++++9577d here have consecrated it, far above our
%%what we say 2y+++++++++++++++6030here, but it can never forget what they di
%%It is for us1z+++++++++++++++0649 the living, rather, to be dedicated here
%%It is rather9+++++++++++++++8834 for us to be here dedicated to the great
%%to that caus7a+++++++++++++++7844e for which they gave the last full measur
%%that this na3b+++++++++++++++1954tion, under God, shall have a new birth of
  ...
```

Figure 39. Input Master Data Set for COB11M and PLI11M

```
%%Lincoln's Ge4a+++++++++++++++0402ttysburg Address, given November 19, 1863
%%Four score a8b+++++++++++++++5154nd seven years ago, our fathers brought fo
%%It is for us1z+++++++++++++++0649 the living, rather, to be dedicated here
%%shall not pe8f+++++++++++++++4843rish from this earth.
  ...
```

Figure 40. New (Distilled) Master Data Set for COB11M and PLI11M

Edit the DA Control File

DA uses assembler listings to determine where to insert breakpoints to monitor file reads. You supply the names of the listing files and data on the variables to read in each listing in the DA control file (CACTL).

DA uses the assembler listings produced by the COBOL or PL/I compiler. Make sure to use the compiler options specified in "Setup" on page 231.

The CACTL control file for this example is *hi_lev_qual*.V1R5M0.SAMPLE.CACTL(ccc11M). The file is shown in Figure 41 on page 138.

To edit the CACTL for the ccc11x example:

1. Select option `1` from the `ATC Primary Option Menu`.

   The `Coverage, Distillation and Unit Test Assistant` panel is displayed.

2. Select option `1` from the `Coverage, Distillation and Unit Test Assistant` panel.

   The `Work with the CA/DA/UTA Control File` panel is displayed.

3. Enter option `1` and specify the following:

Use Program Name for File Name	`YES`
Program Name	`ccc11x`
Listing Type	`COBOL` or `PLI`

4. Press Enter.

   An ISPF edit session for the DA control file you requested is displayed.

   The data in the control file consists of the following types of statements:

   COBOL or **PLI**
   : Indicates that the program is a COBOL (or PL/I) program and specifies information about the compiler listing, object module, and load module.

   SCOPE
   : Specifies information about the scope of the program in which the file of interest is defined.

   FILE
   : Specifies information about the file that is to be distilled, including logical key information.

   If the control file you requested did not previously exist, it is created with comments in it to help you enter the appropriate information in the fields.

   If you want to use the shipped sample CACTL member as a template for your control file, delete the existing lines in the member and copy in *hi_lev_qual*.V1R5M0.SAMPLE.CACTL(ccc11x). Change ATC. to *hi_lev_qual*. for all occurrences. Change the value of the ToObjDsn operand to proj_qual.ZAPOBJ. If you have recompiled the samples into proj_qual data sets, you will also have to edit the ListDsn and FromObjDsn keyword operands to point to your proj_qual listing and object data set names.

   The control file shown in:

   - Figure 41 on page 138 is the control file for *COB*11*x*.
   - Figure 42 on page 138 is the control file for *PLI*11*x*.

5. Verify that the listing file names and the copy to/from object module names are correct for your installation.
6. Press the End key (PF3) to terminate the edit session.

For more detailed information about the control file, see “Editing the Distillation Assistant Control File” on page 145.

```
COBOL ListDsn=ATC.V1R5M0.SAMPLE.COBOLST(COB11M),
      LoadMod=COB11M,
      FromObjDsn=ATC.V1R5M0.SAMPLE.OBJ,
      ToObjDsn=ATC.V1R5M0.SAMPLE.ZAPOBJ

Scope ExtProgram-Id=COB11M
File  File=QSAMIN,KeyPosition=15,KeyLen=20
```

Figure 41. Control File for COB11x

```
PL/I ListDsn=ATC.V1R5M0.SAMPLE.PLILST(PLI11M),
     LoadMod=PLI11M,
     FromObjDsn=ATC.V1R5M0.SAMPLE.OBJ,
     ToObjDsn=ATC.V1R5M0.SAMPLE.ZAPOBJ

     Scope PROCEDURE=PLI11M
     File  File=QSAMIN,KeyPosition=15,KeyLen=20,Stmts=38
```

Figure 42. Control File for PLI11x

Create Setup JCL

Before test cases can be executed on the ccc11x test program, DA must insert breakpoints into the test program. DA does this using the Setup program. When you execute the Setup JCL, the DA Setup program analyzes the assembler listings and creates a table containing breakpoint data (address, op code, and so on). User SVC instructions are inserted for the instructions at the breakpoints in the object modules. You then link the modified object modules into an executable load module for DA to use.

To create the Setup JCL:

1. Select option `1` from the `ATC Primary Option Menu`.

   The `Coverage, Distillation and Unit Test Assistant` panel is displayed.

2. Select option `2`.

   The `Create JCL for Setup` panel is displayed. You create the JCL for the setup of *ccc*11*x* from this panel. All of the default values on the panel are correct for the *ccc*11*x* examples except for possibly the `Enable DA` and the `Program Name` fields.

3. Ensure that `Enable DA` is set to `Yes`, that `Enable UTA` is set to `No`, and that the entry in the `Program Name` field is correct.

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 identified on the panel using member name S*ccc*11*x*.

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

For more detailed information, see “CA, DA, and UTA Setup” on page 231.

Create JCL to Start a Monitor Session

To create the JCL to start a monitor session:

1. Select option 1 from the ATC Primary Option Menu.

   The Coverage, Distillation and Unit Test Assistant panel is displayed.

2. Select option 3.

   The Create JCL to Start the Monitor panel is displayed. You create the JCL for the DA execution of ccc11x from this panel.

3. Ensure that Enable DA is set to YES and that the entry in the Program Name field is correct.

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 identified on the panel using member name X*ccc*11*x*.

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

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

For more detailed information, see “Monitor Execution” on page 245.

Create Physical Distillation JCL

To create the physical distillation JCL:

1. Select option 1 from the ATC Primary Option Menu.

   The Coverage, Distillation and Unit Test Assistant panel is displayed.

2. Select option 5.

   The Distillation panel is displayed.

3. Select option 3.

   The Generate JCL to Generate Key List and Distill Data panel is displayed.

4. Specify the following:

   - Enter '*hi_lev_qual*.V1R5M0.SAMPLE.ccc11.MASTER' in the Master Data Dsn field.
   - Enter 15 in the Key Position field.
   - Enter 20 in the Key Length field.
   - Enter the name you would like to give to the new (distilled) master data set in the Distilled Master Dsn field.
   - Enter *ccc*11*x* in the Program Name field.

5. Select option 1.

   Informational messages are written to your screen as the JCL is created. The created JCL is put into the JCL library identified on the panel using member name D*ccc*11*x*.

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

Create JCL to Link the Modified Object Modules

You must link the modified object modules (modified by the Setup step) into an executable program for testing. You can use the normal JCL that links your program, but be sure you use the object module library that contains the modified object modules. Sample JCL to link the *ccc*11*x* example is provided in *hi_lev_qual*.V1R5M0.SAMPLE.JCL(L*ccc*11*x*).

Create JCL to Run the GO Step

You can use the normal JCL that executes your program, but be sure to specify the load module library that contains the link-edited modified object modules. Sample JCL to execute the GO step for the *ccc*11*x* example is provided in *hi_lev_qual*.V1R5M0.SAMPLE.JCL(G*ccc*11*x*).

Execute the JCL

When you have created all of the *ccc*11*x* JCL, you can run the *ccc*11*x* summary example by executing the following functions in the order listed. To see a flow diagram of these steps, see Figure 38 on page 134

1. S*ccc*11*x*[28]

   Performs the Setup step. All JCL steps should complete with condition code 0.

2. L*ccc*11*x*[29]

   Links the object modules that have been modified with breakpoints in the Setup step into the *ccc*11*x* load module.

3. X*ccc*11*x*[28]

   Starts a monitor session. For DA, a program continuously runs to write variable data to disk. The JCL does not complete until your session is stopped by step 6.

4. G*ccc*11*x*[29]

   Runs sample program *ccc*11*x*. *ccc*11*x* runs to completion with condition code 0.

5. CASTATS[30]

   Displays statistics using the CASTATS command. You should see a nonzero EVNTS count in the TOTALS line. (This is an optional step for illustrative purposes.)

6. CASTOP[30]

   Stops the monitor session. DA writes the variable data to disk.

7. D*ccc*11*x*[28]

   Performs the physical distillation for *ccc*11*x*.

---

[28] JCL created from the panels and put into the JCL library.

[29] JCL supplied with the installation materials in *hi_lev_qual*.V1R5M0.SAMPLE.JCL. (Sample JCL for all of the steps can be found in this partitioned data set [PDS].)

[30] Monitor commands issued from either the `Control the CA/DA/UTA Monitor` panel or the TSO command processor (ISPF option 6) by entering:

```
EX 'hi_lev_qual.V1R5M0.REXX(cacmd)'
```

where `cacmd` is the command issued (such as, CASTATS, CASTOP, and so on).

Logical Distillation

This chapter contains information about logical distillation. Logical distillation is the process of instrumenting your object code and executing the instrumented code under the Distillation Assistant monitor. As the instrumented code reads records from the specified input master data set, the monitor determines which keys in the input master data set caused new code coverage in the instrumented code. The list of these keys is then saved for the physical distillation process, which is described in “Physical Distillation” on page 149.

Description of Reading Input Data Sets

After each read of the input data set that you specified in the Setup step, DA reads and saves the associated key. Breakpoints are inserted in your program so that new statement and conditional branch coverage can be measured. Whenever a statement is executed for the first time, the last key read is saved (if not previously saved). Whenever a condition in a conditional statement (for COBOL: IF, PERFORM, EVALUATE, and GO; for PL/I: IF, DO/END, and SELECT) takes a new path, the last key read is saved (if not previously saved). After termination of your test run using the DA command CASTOP, this list of saved keys is written to disk.

Coverage of the Distilled Data Set

No distilled data set will give equivalent coverage in all cases.

For example, your input data set may be customer purchase records. You may do special processing for customers whose total purchases exceed $1000. DA will record the key of the first customer record that caused the $1000 total to be exceeded. However the distilled data set will probably not contain all of the purchase records that totaled $1000. Therefore the path to process the $1000 customer totals will never be executed while running the distilled data set. In general, any “quantity” type paths executed while processing your input master data set will not be executed while processing the distilled data set.

PL/I ON-Units

If an ON-unit that handles an input condition in a PL/I program returns control to a statement following a READ for which distillation data is being recorded, the monitor cannot tell whether valid data was read or not. This may result in the previous record being included in the distilled output, even if it did not cause any unique coverage.

For example, an ON ENDFILE condition that returns control to the point after the READ causing the condition will cause the last record in the master data set to be included in the distilled data set, even though the end of file occurred after it had been processed.

© Copyright IBM Corp. 1997, 1999

How Much Data Can Be Read

One read of the input data set can save up to 126 bytes of key data.

Keys that cause new coverage are kept in storage buffers while your program is running, and periodically the keys are written to disk. Two buffers are used: one is written to disk while the other is being used. It is possible that the buffers could be full when a new key is to be written to the buffer. However, this is unlikely because:

1. Very few keys cause new coverage, and little data is written to the buffers. The size of each buffer is 65536 bytes. Each data entry contains the key plus eight bytes of other data.
2. To cause a new key to be read, the user program must do I/O. This allows time for the buffer monitor program to write any full buffers to disk.

If a buffer overrun occurs and causes loss of new key data, you will see a statement similar to the following in the VARDATA file of key data:

```
11915        1        1        38     KYLEX  3722AIX.
11916        1        1        38     KYLEX  3723AIX.
******** !!! BUFFER OVERFLOW !!! ********
11917        1        1        38     KYLEX  0000MVS.
```

If a buffer overflow occurs, the program that creates the VDR report of variable data (stepname EXVAR) completes with a return code of 4.

Recording Which Keys Execute a Statement

You can produce an annotated listing (similar to the Coverage Assistant report described in “Annotated COBOL Listings” on page 54, or “Annotated PL/I Listings” on page 64) that contains the key numbers that first execute each statement. Figure 43 shows the portion of an annotated listing that lists these numbers.

```
00150
00151          101-CASE.
00152  :           ADD 1 TO WS-COUNT-101.         >0001 0001 <
00153
00154          102-CASE.
00155  :           ADD 1 TO WS-COUNT-102.         >0001 0016 <
00156
00157          103-CASE.
00158  :           ADD 1 TO WS-COUNT-103.         >0001 0017 <
00159
00160          104-CASE.
00161  :           ADD 1 TO WS-COUNT-104.         >0001 0010 <
00162
00163          105-CASE.
00164  :           ADD 1 TO WS-COUNT-105.         >0001 0013 <
00165
00166          106-CASE.
00167  :           ADD 1 TO WS-COUNT-106.         >0001 0008 <
00168
00169          107-CASE.
00170  :           ADD 1 TO WS-COUNT-107.         >0001 0003 <
00171
```

Figure 43. Partial Annotated Listing for COB011 Showing Key Numbers

In addition to the annotation symbols on each executable statement that appear on the left, two columns of numbers appear to the right of each statement within brackets:

- The number in the first of these columns is the number of times the statement was executed
- The number in the second column is the key number of the key that first caused the statement to execute.

In most cases, DA removes the breakpoint the first time that the statement is executed, therefore the count on most lines will be one. However, the breakpoint on the file read statement in the file being distilled is left in for the entire execution. The count shown on this line will show how many reads to the file were executed.

The key number refers to the keys listed in the DA Key List of keys that caused new coverage.

In Figure 43 on page 142, key 1 (with key 4a) caused 101-CASE to execute first, and key 16 (with key 7n) caused 102-CASE to execute first.

You create an annotated listing with key data by changing your defaults before generating the setup JCL. To change your defaults:

1. Select option `0` from the `ATC Primary Option Menu`.
   This displays the `Manipulate ATC Defaults` panel.
2. Select option `1`.
   The `ATC Defaults` panel is displayed.
3. In the `Setup Defaults` area of the panel, change `Frequency Count Mode` to YES.

Complete the remaining DA steps to run the program under test.

Then generate the annotated listing with key data by executing a CA annotated listing report, as described in “Annotated COBOL Listings” on page 54 and “Annotated PL/I Listings” on page 64.

Editing the Distillation Assistant Control File

This chapter describes the function of the control file (CACTL) used by DA. The CACTL contains information that tells DA what compile units to analyze and the file that is to be monitored. CA, DA, and UTA share the CACTL file. See “CA, DA, and UTA Control File” on page 209 for a complete description of this control file. This chapter only explains how to use the control file with Distillation Assistant.

Contents of the Control File

The control file consists of a series of lines that specify information about the file to be distilled. The following describes the contents of the control file as it is used by DA. See “Contents of the Control File” on page 211 for a description of the syntax of the control file.

- The COBOL statement specifies the following:
  - The source language (COBOL)
  - The data set containing the compiler listing file
  - The name of the load module containing the program
  - The data set containing either the object code generated by the complier 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
- The PL/I statement specifies the following:
  - The source language (PL/I)
  - The data set containing the compiler listing file
  - The name of the load module containing the program
  - The data set containing either the object code generated by the complier 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

© Copyright IBM Corp. 1997, 1999

- The SCOPE statement specifies the following:
  - COBOL
    - The PROGRAM-ID of the external program in which the file is defined. The external program ID is the first program ID in the listing file.
    - The PROGRAM-ID of the nested (internal) program in which the file is defined. This operand is required only if the file is defined within a nested program.
  - PL/I
    - The name of the procedure or begin block in which the file is defined.
- The FILE statement specifies the following:
  - That all reads of the specified file are to be monitored
  - For COBOL, the FD name of the file to be monitored
  - For PL/I, the name of the file constant to be monitored
  - The position and length of the logical key within the file
  - For PL/I, the statement number where the file is read.

When performing distillation, you would normally want to monitor all reads of the file as is done using the statements previously described. If, however, for some reason you only want to monitor reads into a specific variable or group, the FILE statement previously described could be replaced by a VARIABLE and COVERAGE statement specifying the FILE option (for COBOL only). The syntax of these statements is described in “Contents of the Control File” on page 211.

Examples

The following figure shows an example of a file read in a COBOL program:

```
   COBOL ListDsn=ATC.V1R5M0.SAMPLE.COBOLST(COB11M),
         LoadMod=COB11M,
         FromObjDsn=ATC.V1R5M0.SAMPLE.OBJ,
         ToObjDsn=ATC.V1R5M0.SAMPLE.ZAPOBJ

   Scope ExtProgram-Id=COB11M
   File  File=QSAMIN,KeyPosition=15,KeyLen=20
```

Figure 44. CACTL Statements for Distillation (COBOL)

The example in Figure 44 is based on the following COBOL declarations:

```
 IDENTIFICATION DIVISION.
 ...
 DATA DIVISION.
 FILE SECTION.

 FD  QSAMIN
     RECORDING MODE IS F
     BLOCK CONTAINS 0 RECORDS
     LABEL RECORDS ARE STANDARD.
 01  INPUT-RECORD               PIC X(76).

     EJECT
 WORKING-STORAGE SECTION.
 ...
```

Figure 45. COBOL File Definition. All reads for the file QSAMIN are monitored. Whenever a record causes new code coverage, the key field located in columns 15 to 34 of the record is saved.

The following figure shows an example of a file read in a PL/I program:

```
  PL/I ListDsn=ATC.V1R5M0.SAMPLE.PLILST(PLI11M),
       LoadMod=PLI11M,
       FromObjDsn=ATC.V1R5M0.SAMPLE.OBJ,
       ToObjDsn=ATC.V1R5M0.SAMPLE.ZAPOBJ

       Scope PROCEDURE=PLI11M
       File  File=QSAMIN,KeyPosition=15,KeyLen=20,Stmts=38
```

Figure 46. CACTL Statements for Distillation (PL/I)

Physical Distillation

This chapter describes physical distillation. Physical distillation is the process of creating a new master data set by reading the list of keys produced in the logical distillation step (described in “Logical Distillation” on page 141) and the input master data set. The new master data set consists of only those records in the input master data set whose logical keys appear in the list of keys.

Physical Distillation Summary

The physical distillation process has two steps:

1. Format the list of keys that caused new coverage from the data saved by the execution monitor.
2. Use this key list to distill the input master data set.

You can create JCL to run these two steps consecutively, or you can create JCL to run each step separately. If you run the steps separately, you can inspect and edit the key list to produce a customized distillation data set. For example, you may want to include keys in a distilled data set even though they did not cause new coverage, or you may want to remove other keys that caused new coverage, but are not important to your testing.

Before the physical distillation begins, the new master data set is, in most cases, deleted if it currently exists, and allocated with the attributes of the input master data set. The exceptions to this rule are:

- If the new master data set is a member of a partitioned data set, the data set is not deleted and reallocated. However, if the attributes of the new master partitioned data set do not match those of the input master data set, the physical distillation process is ended.
- If a volume is specified for the new master data set, the data set is not deleted and reallocated. However, if the attributes of the new master data set do not match those of the input master data set, the physical distillation process ends.
- If the new master data set is on tape, no check is made to see if the data set exists; the specified file on the tape is simply rewritten.

© Copyright IBM Corp. 1997, 1999

The following chart shows the input/output combinations that are supported:

Table 1. Supported Input/Output Combinations

Input	Output					
	seq		pds(mem)		vsam	
	Exists	New	Exists	New	Exists	New
seq	del/realloc[a]	alloc like	add/replace	no	no	no
pds(mem)	no[b]	no	add/replace	alloc like add	no	no
vsam	no	no	no	no	del/realloc[a]	alloc like

Notes:

del/realloc Delete and reallocate like input data set.
add/replace Add or replace member in existing PDS.
add Add member in existing PDS.
alloc like Allocate like input data set.

a If the volume/unit is specified for the output data set, no del/realloc will be done (the output data set will be reused).
b If the volume/unit is specified for the output data set, pds(mem) to seq is allowed.

Parameters Used by Physical Distillation

The inputs to the physical distillation process are:

- Input master data set
- Input master data set's volume
- Input master data set's unit
- Key position in master data set
- Key length in master data set

Outputs from the logical distillation process are:

- Debug table
- Variable table

The following intermediate files are created during the physical distillation process:

- Variable ID file
- Variable data file

The physical distillation process creates the new master data set.

Running Physical Distillation

You generate the JCL to run the physical distillation from the Distillation panel. To display the Distillation panel:

1. Select option 1 from the ATC Primary Option Menu.

   The Coverage, Distillation and Unit Test Assistant panel is displayed.
2. Select option 5.

   The Distillation panel is displayed.
3. Select option 3.[31]

   The Generate JCL to Generate Key List and Distill Data panel, shown in Figure 47, is displayed. From this panel, you can generate JCL, edit JCL, and submit JCL for physical distillation.

```
-------------- Generate JCL to Generate Key List and Distill Data ------
Option ===>

1  Generate DASD Generate JCL for distilling master file on DASD
2  Generate Tape Generate JCL for distilling master file on TAPE
3  Edit          Edit JCL
4  Submit        Submit JCL

Enter END to Terminate
```

Figure 47. Generate JCL to Generate Key List and Distill Data Panel

The panel's options are as follows:

Generate DASD
: Generates the JCL required to perform both steps in the physical distillation. Use this option when the new distilled master data set will be stored on a DASD (direct access storage device).

Generate Tape
: Generates the JCL required to perform both steps in the physical distillation. Use this option when the input master data set is on tape and the new distilled master data set will be stored on tape.

Edit
: Starts an ISPF edit session for the JCL created by either Generate option.

Submit
: Submits the JCL created by either Generate option for physical distillation.

[31] If you would like to edit the key list after it is produced, you can select the KeyList option (option 1) to generate the key list and edit it. Select the DistillKey option (option 2) to complete the distillation. Because the fields used in these options are very similar to those used in the Distill option, they are not discussed here.

Generating JCL for Physical Distillation

To generate the JCL required for physical distillation, select the appropriate generate option (option 1 or 2) from the `Generate JCL to Generate Key List and Distill Data` panel. Either of these options allow you to perform both steps of the physical distillation.

If you select option 1, the `Generate JCL to Generate Key List and Distill DASD Data` panel, shown in Figure 48, is displayed. If you select option 2, the `Generate JCL to Generate Key List and Distill Tape Data` panel, shown in Figure 49 on page 154, is displayed.

The following topics describe how to complete these panels.

Generating JCL for DASD

If the new distilled master data set will be stored on DASD, complete the `Generate JCL to Generate Key List and Distill DASD Data` panel as described in this topic, and then press Enter. This panel allows you to generate the JCL required for both steps of the physical distillation: (1) generate a key list, and (2) distill the new master data set from the input master data set.

```
----------- Generate JCL to Generate Key List and Distill DASD Data ----
Command ===>

Enter END to Terminate

Input Master File:
  Master Data Dsn . . . 'ATC.V1R5M0.SAMPLE.COB11.MASTER'
  Volume Serial . . . .           Unit . .          (If not cataloged)
  Key Position  . . . . 15        (1 is first character of record)
  Key Length  . . . . . 20

Output Master File:
  Distilled Master Dsn  'YOUNG.TEST.COB11.DISTFILE'
  Volume Serial . . . .           Unit . .          (If not cataloged)

Use Program Name for File Name  YES (Yes|No) Program Name  COB11M

Input Files:
  Debug Table Dsn . . . 'YOUNG.TEST.COB11M.DBGTAB'
  Variable Table Dsn. . 'YOUNG.TEST.COB11M.VARTAB'

JCL Library and Member:
  JCL Dsn . . . . . . . 'YOUNG.TEST.JCL.CNTL(DCOB11M)'

Output Files:
  Variable ID Dsn . . . 'YOUNG.TEST.COB11M.VARID'
    (* for default sysout class)
  Variable Data Dsn . . 'YOUNG.TEST.COB11M.VARDATA'
```

Figure 48. Generate JCL to Generate Key List and Distill DASD Data Panel

The panel's fields are as follows:

Master Data Dsn
Specifies the data set name of the input master data set to be distilled.

Volume Serial
Specifies the volume on which the data set resides.

Unit
Specifies the device on which the data set resides.

Key Position
Specifies the position of the logical key in each input master data set record. Position 1 is the first byte of the record.

Key Length
Specifies the length of each logical key in the input master data set.

Distilled Master Dsn
The name of the new master data set (the result of the distillation).

Volume Serial
Specifies the volume on which the new data set will be stored.

Unit
Specifies the device on which the new data set will be stored.

Use Program Name for File Name
If you want to construct the subsequent data set names from the default high-level qualifier, the specified program name, and the default low-level qualifier for each data set, enter `YES`.

When you press Enter, the file names on the panel are changed automatically. Using the program name is the normal DA procedure.

Program Name
Specifies the next-to-last qualifier to be used when the `Use Program Name for File Name` field is set to `YES`.

Debug Table Dsn
Specifies the name of the debug table produced by the DA logical distillation.

Variable Table Dsn
Specifies the name of the variable table produced by the DA logical distillation.

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

Note: 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 D*xxxxxxx*, where *xxxxxxx* is the first seven characters of the program name.

Variable ID Dsn

Specifies the name of the data set that will contain the variable ID intermediate file created during the physical distillation process.

Variable Data Dsn

Specifies the name of the data set that will contain the variable data intermediate file created during the physical distillation process.

Generating JCL for Tape

If the new distilled master data set will be stored on tape, complete the `Generate JCL to Generate Key List and Distill Tape Data` panel as described in this topic, and then press Enter. This panel allows you to generate the JCL required for both steps of the physical distillation: (1) generate a key list, and (2) distill the new master data set from the input master data set.

```
----------- Generate JCL to Generate Key List and Distill Tape Data ----
Command ===>

Enter END to Terminate

Input Master File:
  Master Data Dsn . . . 'ATC.V1R5M0.SAMPLE.COB11.MASTER'
  Volume Serial . . . . DADST1    Unit . . 3480     (If not cataloged)
  File Number . . . . . 1
  Key Position  . . . . 15        (1 is first character of record)
  Key Length  . . . . . 20

Output Master File:
  Distilled Master Dsn  'YOUNG.TEST.COB11.DISTFILE'
  Volume Serial . . . . DADST2    Unit . . 3480
  File Number . . . . . 1         Expiration Date
  Catalog Data Set  . . No        (Yes|No)

Use Program Name for File Name  YES (Yes|No) Program Name  COB11M

Input Files:
  Debug Table Dsn . . . 'YOUNG.TEST.COB11M.DBGTAB'
  Variable Table Dsn. . 'YOUNG.TEST.COB11M.VARTAB'

JCL Library and Member:
  JCL Dsn . . . . . . . 'YOUNG.TEST.JCL.CNTL(DCOB11M)'

Output Files:
  Variable ID Dsn . . . 'YOUNG.TEST.COB11M.VARID'
    (* for default sysout class)
  Variable Data Dsn . . 'YOUNG.TEST.COB11M.VARDATA'
```

Figure 49. Generate JCL to Generate Key List and Distill Tape Data Panel

The panel's fields are as follows:

Master Data Dsn
Specifies the name of the input master data set to be distilled.

Volume Serial
Specifies the volume on which the data set resides.

Unit
Specifies the device on which the data set resides.

File Number
Specifies the position count of the data set relative to other data sets on the tape. The first data set is number 1.

Key Position
Specifies the position of the logical key in each input master data set record. Position 1 is the first byte of the record.

Key Length
Specifies the length of each logical key in the input master data set.

Distilled Master Dsn
The name of the new master data set (the result of the distillation).

Volume Serial
Specifies the volume on which the new data set will be stored.

Unit
Specifies the device on which the new data set will be stored.

File Number
Specifies the position count of the data set relative to other data sets on the tape. The first data set is number 1.

Expiration Date
Specifies the expiration date for the new data set. The recommended format is YYYY/DDD.

Catalog Data Set
Specifies whether the system is to make an entry pointing to the data set in the system or user catalog.

Use Program Name for File Name
If you want to construct the subsequent data set names from the default high-level qualifier, the specified program name, and the default low-level qualifier for each data set, enter `YES`.

When you press Enter, the file names on the panel are changed automatically. Using the program name is the normal DA procedure.

Program Name
Specifies the next-to-last qualifier to be used when the `Use Program Name for File Name` field is set to `YES`.

Debug Table Dsn
Specifies the name of the debug table produced by the DA logical distillation.

Variable Table Dsn
Specifies the name of the variable table produced by the DA logical distillation.

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

Note: 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 D*xxxxxxx*, where *xxxxxxx* is the first seven characters of the program name.

Variable ID Dsn
Specifies the name of the data set that will contain the variable ID intermediate file created during the physical distillation process.

Variable Data Dsn
Specifies the name of the data set that will contain the variable data intermediate file created during the physical distillation process.

Editing Distillation JCL

To start an ISPF edit session for the distillation JCL you created with either the `Generate DASD` option or the `Generate Tape` option:

1. Select option `2` from the `Generate JCL to Generate Key List and Distill Data` panel.

   The `Edit Distillation JCL` panel, shown in Figure 50, is displayed.

```
---------------------------- Edit Distillation JCL ---------------------
Command ===>

Enter END to Terminate

Use Program Name for File Name  YES (Yes|No) Program Name  COB11M

JCL Library and Member:
  JCL Dsn . . . . . . . 'YOUNG.TEST.JCL.CNTL(DCOB11M)'
```

Figure 50. Edit Distillation JCL Panel

2. Complete the panel, and press Enter.

The panel's fields are as follows:

Use Program Name for File Name
If you want to construct the subsequent data set names from the default high-level qualifier, the specified program name, and the default low-level qualifier for each data set, enter `YES`.

When you press Enter, the file names on the panel are changed automatically. Using the program name is the normal DA procedure.

Program Name
Specifies the next-to-last qualifier to be used when the `Use Program Name for File Name` field is set to `YES`.

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

Note: 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 D*xxxxxxx*, where *xxxxxxx* is the first seven characters of the program name.

Submitting JCL for Physical Distillation

To submit JCL that you created with either the `Generate DASD` option or the `Generate Tape` option:

1. Select option `4` from the `Generate JCL to Generate Key List and Distill Data` panel.

   The `Submit Distillation JCL` panel, shown in Figure 51, is displayed.

```
--------------------------- Submit Distillation JCL --------------------
Command ===>

Enter END to Terminate

Use Program Name for File Name  YES (Yes|No) Program Name  COB11M

JCL Library and Member:
  JCL Dsn . . . . . . . 'YOUNG.TEST.JCL.CNTL(DCOB11M)'
```

Figure 51. Submit Distillation JCL Panel

2. Complete the panel, and press Enter to submit the batch job.

The panel's fields are as follows:

Use Program Name for File Name
If you want to construct the subsequent data set names from the default high-level qualifier, the specified program name, and the default low-level qualifier for each data set, enter `YES`.

When you press Enter, the file names on the panel are changed automatically. Using the program name is the normal DA procedure.

Program Name
Specifies the next-to-last qualifier to be used when the `Use Program Name for File Name` field is set to `YES`.

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

Note: 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 D*xxxxxxx*, where *xxxxxxx* is the first seven characters of the program name.

3. Inspect the output to ensure that the distillation ran successfully.

Physical Distillation Return Codes

When physical distillation is done successfully, the return codes for all steps are zero. The following information is available in the SYSPRINT DD name of the DISTILL step:

- Number of keys: 0000000019
- Number of input master records: 0000001440
- Number of output master records: 0000000019

The previous values are for the distillation test case shipped with ATC.

When physical distillation fails, the return code of the DISTILL step is nonzero. The return code is the number of an FCV*xxx* error. The error message is printed in the SYSPRINT DDNAME of the DISTILL step. For more information about the error, see Appendix A, “Problem Determination” on page 307.

Using Unit Test Assistant

© Copyright IBM Corp. 1997, 1999

Introduction

This chapter contains the following topics:

- What Is Unit Test Assistant?
- What Does UTA Require?
- How Does UTA Work?
- Where Can You Get Further Details?

What Is Unit Test Assistant?

Unit Test Assistant (UTA) allows you to capture and log the values assigned to selected variables in your application programs at selected points during their execution. This is called *unit testing*. Unit testing allows you to confirm the effectiveness of Year 2000 changes that have been made to an application program.

In addition, Unit Test Assistant offers the ability to perform data *warping*. This means that variables can be modified automatically as they are encountered during program execution. UTA will intercept data entering or leaving a program at I/O time (or at other times where application logic dictates) and change the value of that data in a manner that you specify. This feature is especially useful when doing future date testing of Year 2000 remediated code.

UTA provides two types of data warping:

1. File warping
2. Dynamic data warping

A standard data warping process is to age, or warp, occurrences of dates in the input data files. UTA's file warp feature copies VSAM or QSAM files and warps record data fields in the copied files under user control. File warping is described in more detail in “Unit Test Assistant File Warping” on page 197. The remainder of this chapter describes logging and dynamic runtime warping of variables.

UTA supports applications generated by the following compilers:

- IBM COBOL for OS/390 & VM 2.1 (plus Millennium Language Extensions [MLE])
- IBM COBOL for MVS & VM 1.2 (plus Millennium Language Extensions [MLE])
- IBM VS COBOL II Release 4.0
- IBM OS/VS COBOL Release 2.4
- The following PL/I compilers are supported for data warping of file input buffers **only**:
  - IBM PL/I for MVS & VM 1.1.1 (plus Millennium Language Extensions [MLE])
  - IBM OS PL/I Optimizing Compiler 2.3.0
  - IBM PL/I Optimizing Compiler 1.5.1

For variables that you select to monitor in any compile units in any of your programs, UTA saves the contents of the variables while your program is executing, and writes this data to a log file for later analysis. (Warped variables are only logged if there is an error.)

© Copyright IBM Corp. 1997, 1999

UTA has the following characteristics:

- Low overhead. For a test case run, UTA typically adds very little to the execution time of the program. UTA inserts SVC instructions as *breakpoints* and then is given control of MVS when these SVCs are executed.
- Panel-driven user interface. You can use an ISPF panel-driven interface to create JCL for executing UTA programs.
- Simple, flexible control. The control file used to define what variables to monitor provides a simple method of controlling UTA operations.

What Does UTA Require?

UTA has the following requirements:

- UTA runs under MVS. Detailed MVS system resource requirements for UTA are described at "DA and UTA Resources" on page 388. UTA uses ISPF services to display dialogs and to produce the JCL to run the UTA steps.
- As part of its input, UTA requires listings created by the application program compilers that it supports. These compilers offer options that allow you to include assembler statements, data maps, and data cross-references in the listings, all of which UTA uses.
- UTA also requires the application program object modules as input. UTA creates copies of these object modules with breakpoints inserted into them.

See "Execution" on page 165 for a description of how the UTA authorized programs intercept breakpoints.

How Does UTA Work?

Running UTA consists of the following steps: This list is an overview of the process. **Each activity is described in more detail in topics that follow in this chapter.**

Step 1 Setup

a. Compile the source code that you want to analyze, using the required compiler options.

b. Generate UTA JCL using the UTA ISPF dialog:

   1) Edit the UTA control file.
   2) Create the setup JCL.
   3) Create the monitor JCL.
   4) Create the variable report JCL.

c. If you instrumented object modules, edit your program's link-edit JCL to pick up the modified object modules, which are created by the Setup step.

d. Edit your program's GO JCL (or program invocation) to point to the new load module that is created when you run the JCL created in step 1c.

Step 2 Execution

a. Run the setup JCL (created at step 1b2).
b. Run the link-edit JCL (created at step 1c).
c. Run the monitor JCL to start a monitor session (created at step 1b3).
d. Run your application using the load module(s) created in step 2b.
e. Stop the monitor session (using the CASTOP command).

Step 3 Report

a. Run the variable report JCL (created at 1b4 on page 162).

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.

Figure 52 shows a diagram of the entire process.

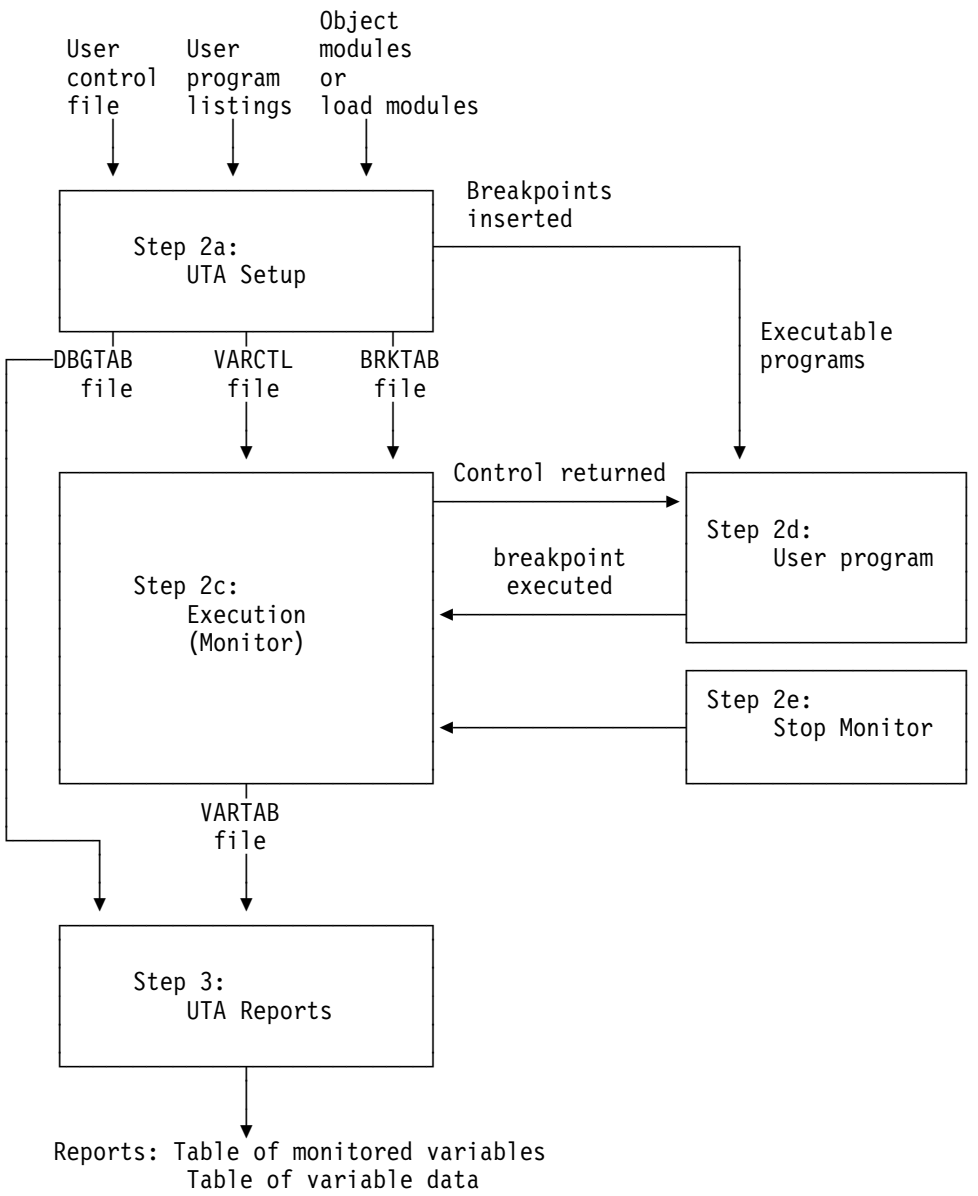


Figure 52. UTA—Flow Diagram

Setup

UTA Setup analyzes assembler statements and cross-reference information included in the compiler output listings to determine where to insert breakpoints into disk-resident copies of the object modules containing variables you want to examine. It then inserts the breakpoints.

Setup runs in MVS. To run it, you need:

1. Compiler listings of the object modules
2. The object modules or load modules you want to test
3. The user control file listing variables to monitor

The Setup step produces:

1. Modified test programs (object modules or load modules) containing breakpoints
2. A file of breakpoint-related information (called *BRKTAB* in this User's Guide) required for the monitor program in the Execution step.
3. A file of variable control data on variables to monitor (called *DBGTAB* in this User's Guide).
4. A file of variable locations to be used by the monitor (called *VARCTL* in this User's Guide).

Execution

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

Start the UTA monitor program and run your test case programs. As the selected breakpoints are encountered, the monitor gains control, logs the predefined selected variable values, changes (warps) predefined variables or buffers, and then returns control to your program. After your test cases have completed, use CASTOP to stop the monitor session. It writes the results (variable values) to a file called VARTAB.

The monitor inserts reserved supervisor call (SVC) instructions as breakpoints and is given control by MVS when these SVC instructions are executed in a program. Using SVCs as breakpoints is the architected way to receive control from MVS, and requires no modification to MVS. This technique is called user SVCs.

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 UTA Report program uses the results from the monitor and from DBGTAB to produce a report of all logged user-defined variables.

To run reports, you need:

1. The DBGTAB data set from the Setup step
2. The VARTAB data set from the Execution step

The reports produced are as follows:

- Monitored Variables Report (MVR). A table of monitored variables and their read specifications. The MVR is always generated.
- Either of the following reports:
  - Variable Data Report (VDR). A table of data that is read during execution with references to the MVR for variable identification.
  - Combined Variable Data Report (CVDR). A variation of the VDR, which includes a fully-qualified variable name for each entry.

See pages 191 and 192 for examples of these reports.

Where Can You Get Further Details?

Refer to the following sections for additional information.

For information about...	See...
Samples of UTA test case coverage, including sample reports	“Unit Test Assistant Samples” on page 167
Editing the CACTL file that contains the names of the listing data sets	“Editing the Unit Test Assistant Control File” on page 187
Setting up the table of breakpoints from the listings	“CA, DA, and UTA Setup” on page 231
Starting a UTA monitor session and running test cases on your programs	“Monitor Execution” on page 245
Reports on the test run	“Unit Test Assistant Reports” on page 189
Commands that control the UTA monitor program	“Monitor Commands” on page 255
System resources needed by UTA	“DA and UTA Resources” on page 388

Unit Test Assistant Samples

This chapter describes samples of Unit Test Assistant (UTA) variable monitoring support using examples provided with the ATC package.

UTA monitors or warps the values of the variables that you select in the control file.

The samples comprise the monitored variable report (MVR) and the variable data report (VDR) for the COBOL test cases:

- COB02M (COBOL for IBM MVS & VM 1.2 sample)[32]
- COB022 (IBM VS COBOL II Release 4.0 sample)
- COB02O (IBM OS/VS COBOL Release 2.4 sample)
- COB01M (COBOL for IBM MVS & VM 1.2 sample)
- COB012 (IBM VS COBOL II Release 4.0 sample)
- COB01O (IBM OS/VS COBOL Release 2.4 sample)

A flow diagram of the steps required to run these samples is shown in Figure 53 on page 168. The names in the steps are the member names of the JCL executed for the step. For example, SCOB*nnx*, where *nn* is the test case number and *x* is *M* for COBOL for MVS & VM, *2* for VS COBOL II, or *O* for OS/VS COBOL.

The following UTA samples use the ATC ISPF dialog to create the JCL to run the UTA steps. The ATC ISPF dialog is provided as an aid in creating the JCL. Once the JCL is created for a test environment, it does not have to be recreated from the dialog. In a typical user test environment, the creation of the JCL can be incorporated into the user's procedures. You do not have to use the ISPF dialog to use UTA.

[32] You can also test the CA and UTA installation for the COBOL for OS/390 & VM compiler by copying the JCL members and making minor edits to the compiler, link-edit, and runtime library names.

© Copyright IBM Corp. 1997, 1999

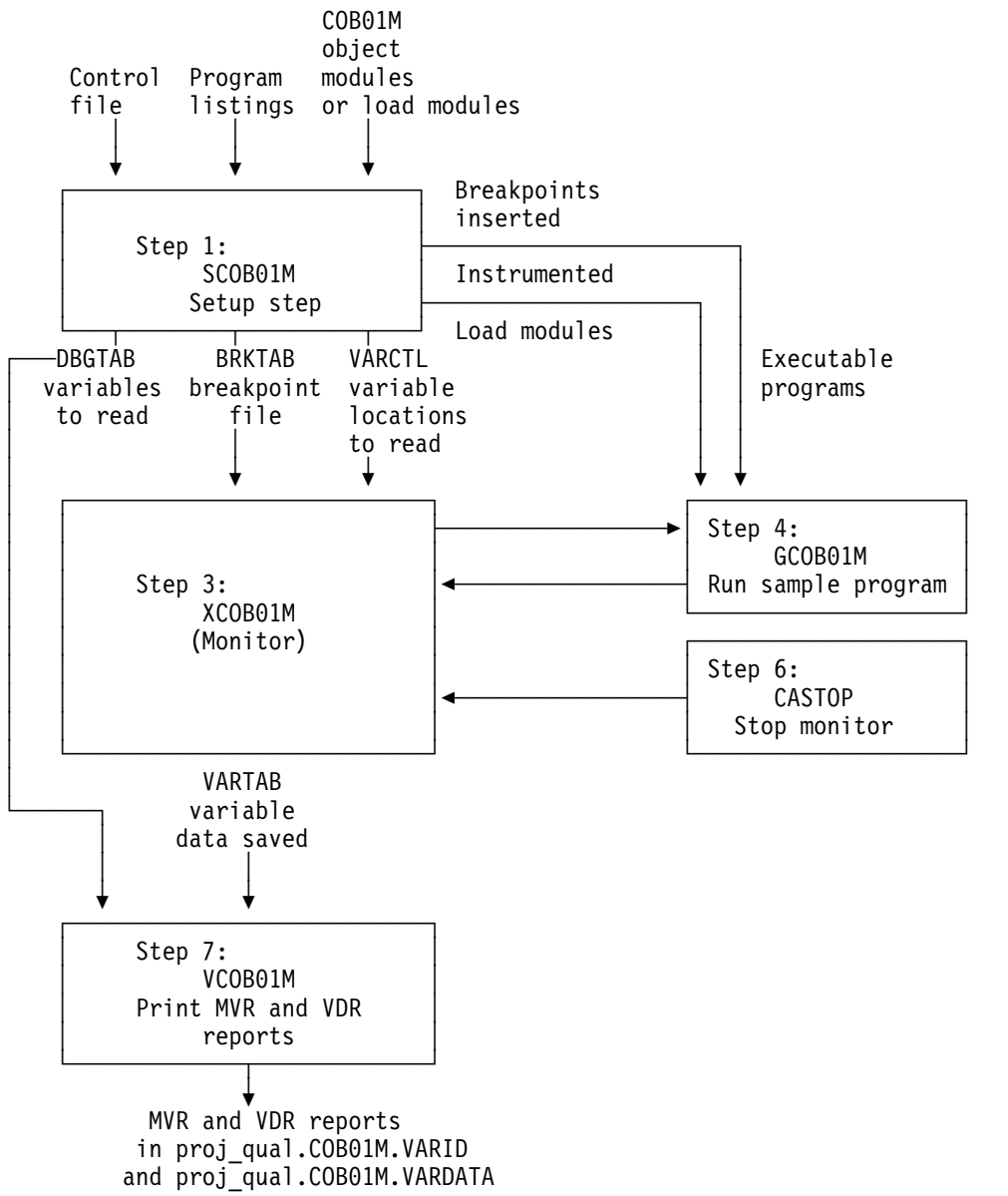


Figure 53. Sample Run—Flow Diagram

COB02x Test Cases

The COB02M, COB022, and COB020 test cases are examples that include reading variables from a single compile unit and using data warping to set, increment, and decrement variables. The control file is shown in Figure 56 on page 173, the MVR is shown in Figure 54 on page 171, and the VDR is shown in Figure 55 on page 171.

A total of eight variables are monitored in this test case. The first two are each set using data warping, and are read on subsequent statements. JULIAN-DATE is set to 0099365 on statement #87 as variable 2, and read on statement #94 as variable 1. CURR-DATE is set to 00991231 on statement #97 as variable 4, and read on statement #103 as variable 3. These values represent the appropriate format of the date December 31,1999 with a two-digit year. The variable numbers are those from the VARID report. Note that the variable numbers representing warp statements do not appear in the VARDATA report unless an error occurs.

The next entry, YEAR4, is incremented by three on statement #106, and read on statements #106 and #114 as variables 5 and 6. YEAR of YEAR-BY-FIELD of DATE-BY-FIELD is decremented by one on statement #106 as variable 7. This variable is subsequently copied into variable 8, YEAR2, which is read on statement #117.

The next two entries, variables 9 and 10, are both for BEGIN-DATE of LOAN, and demonstrate the difference between the NAME and FULL options in the control file coverage line. Variable 9 in the MVR uses the NAME option, and variable 10 uses the FULL option. As can be seen from the MVR, when using the NAME option, the variable is read only on statements #97 and #117, where it is referenced by the level name BEGIN-DATE. Using the FULL option, the variable is also read at all statements on which the high-level qualifier LOAN is accessed.

The next entry, variable 11, indicates that JULIAN-DATE is to be read only once on statement #143.

The final entry, variable 12, indicates that the variable J-DAY of J-DATE should be read once every 100th time that statement #153 is executed, for a maximum of five reads.

The reports for OS/VS COBOL have fewer statements listed for the variable BEGIN-DATE of LOAN than the reports from the other compilers. Other than that, the reports should be the same regardless of the compiler that is used for the programs under test.

For COBOL programs, breakpoints are inserted into the object modules during Setup. These breakpoints are positioned so that the requested variable data can be read when the breakpoint interrupt occurs. When you are ready to test your program, you link the object modules that have been modified with the breakpoints.

To produce the reports for COB02*x*, you need to perform the following steps. **Steps 2a through 6 are described in more detail in topics that follow in this chapter.**

1. Compile the COBOL source you want to test. This produces listings that include the assembler statements needed by UTA. (This has already been done for the examples. The listings are in *hi_lev_qual*.V1R5M0.SAMPLE.COBOLST for the COBOL for MVS & VM and VS COBOL II compilers, and in *hi_lev_qual*.V1R5M0.SAMPLE.COSVSLST for the OS/VS compiler.)

   Make sure to use the compiler options specified in "Setup" on page 231.

2. Start the ATC ISPF dialog by entering the following from ISPF option 6:

   ```
   EX 'hi_lev_qual.V1R5M0.REXX(ATSTART)'
   ```

   The `ATC Primary Option Menu` is displayed.

   a. Edit the UTA control file.

      Verify that the control file includes the listings of the object modules you want to test and information on the variables you want to monitor.

   b. Create the setup JCL.

      Create the JCL that enables the Setup job to produce files containing breakpoint data and variable read data, and to modify copies of your object modules by inserting breakpoints.

   c. Create the start monitor session JCL.

      Create the JCL to start a monitor session.

   d. Create the variable report JCL.

      Create the JCL to produce the MVR and VDR reports after the COB02*x* test case has executed.

3. End the ATC ISPF dialog by pressing the End key (PF3) on the `ATC Primary Option Menu`.

4. Create the JCL to link the modified object modules.

   After the Setup step, and before the Execution step, you must link the modified object modules into an executable program 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. Create the JCL to run the GO step.

   Create the JCL to run your program. Specify the same modified load module as in step 4.

6. Execute the JCL.

   Execute the created JCL files for COB02*x* in the correct order. (This order is shown in "Execute the JCL" on page 175.)

Figures 54 and 55 on page 171 show sample variable reports. For details about the information included in the report, see "Unit Test Assistant Reports" on page 189.

```
* DATE: 12/12/1998
* TIME: 05:58:16
*
*
*CU Name:              ATC.V1R5M0.SAMPLE.COBOLST(COB02M)
*External Program Id: COB02M
*
*    CU    var    var   data   read   read   read   read   read   prog      var
* numbr     ID   mode   type  offst  lngth   freq    max  stmts     id      name
*-------------------------------------------------------------------------------------------------------------
      1      1   I      C         0      7      1            94   COB02M    01 JULIAN-DATE
      1      2   I      Z         0      7      1            87   COB02M    01 JULIAN-DATE
      1      3   I      C         0      8      1           103   COB02M    01 CURR-DATE
      1      4   I      Z         0      8      1            97   COB02M    01 CURR-DATE
      1      5   I      C         0      4      1           114   COB02M    01 YEAR4
      1      6   I      Z         0      4      1           106   COB02M    01 YEAR4
      1      7   I      Z         0      2      1           106   COB02M    01 DATE-BY-FIELD 02 YEAR-BY-FIELD 03 YEAR
      1      8   I      C         0      2      1           117   COB02M    01 YEAR2
      1      9   I      C         0      8      1            97   COB02M    01 LOAN 02 BEGIN-DATE
                                                           117
      1     10   I      C         0      8      1            97   COB02M    01 LOAN 02 BEGIN-DATE
                                                           117
                                                           124
                                                           127
                                                           130
                                                           134
      1     11   I      C         0      7      1           143   COB02M    01 JULIAN-DATE
      1     12   I      C         0      3    100      5    153   COB02M    01 J-DATE 02 J-DAY
```

Figure 54. MVR for COB02M

```
* DATE: 12/12/1998
* TIME: 05:58:16
*
*
*    seq      CU     var      at         data
*  numbr   numbr      ID    stmt  stat   read
*-----------------------------------------------
       1       1       1      94         1999365
       2       1      10      97
       3       1       9      97
       4       1       3     103         00991231
       5       1       5     114         2002
       6       1      10     117         00991231
       7       1       9     117         00991231
       8       1       8     117         98
       9       1      10     124         19981231
      10       1      10     127         19981231
      11       1      10     130         19981231
      12       1      10     134         19981231
      13       1      12     153         098
      14       1      12     153         198
      15       1      12     153         298
      16       1      12     153         032
      17       1      12     153         132
      18       1      11     143         2002267
```

Figure 55. VDR for COB02M

Edit the UTA Control File

UTA uses assembler listings to determine where to insert breakpoints to monitor variables. You supply the names of the listing files and data on the variables to read in each listing in the UTA control file (CACTL).

Make sure to use the compiler options specified in "Setup" on page 231.

The CACTL control file for the COBOL one compile unit example is *hi_lev_qual*.V1R5M0.SAMPLE.CACTL(COB02M). The file is shown in Figure 56 on page 173.

To edit the CACTL for the COB02x one compile unit example:

1. Select option `1` from the `ATC Primary Option Menu`.

   The `Coverage, Distillation and Unit Test Assistant` panel is displayed.

2. Select option `1` from the `Coverage, Distillation and Unit Test Assistant` panel.

   The `Work with the CA/DA/UTA Control File` panel is displayed.

3. Enter option `1` and specify the following:

   Control File Dsn `COB02x`
   Listing Type `COBOL`

4. Press Enter.

   An ISPF edit session for the UTA control file you requested is displayed.

   The data in the control file consists of the following types of statements:

   DEFAULTS Specifies default values for operands not specified on subsequent COBOL statements.

   COBOL Specifies the data sets containing the compiler listing, object module, instrumented object module, and load module for the program that is to be tested.

   SCOPE Specifies the program scope in which the variables to be tested are defined.

   VARIABLE Specifies the name of the variables to be tested.

   COVERAGE Specifies detailed information about the type of coverage testing to be performed on a particular variable.

   WARP Specifies detailed information about the type of data warping to be performed on a particular variable or buffer.

   If the control file you requested did not previously exist, it is created with comments in it to help you enter the appropriate information in the fields.

   If you want to use the shipped sample CACTL member as a template for your control file, delete the existing lines in the member and copy in *hi_lev_qual*.V1R5M0.SAMPLE.CACTL(COB02X). Change ATC. to *hi_lev_qual*. for all occurrences. Change the value of the ToObjDsn operand to proj_qual.ZAPOBJ. If you have recompiled the samples into proj_qual data sets, you will also have to edit the ListDsn and FromObjDsn keyword operands to point to your proj_qual listing and object data set names.

   The control file shown in Figure 56 on page 173 is the control file for COB02x.

5. Verify that the data set names are correct for your installation.
6. Press the End key (PF3) to terminate the edit session.

For more detailed information about the control file, see “Editing the Unit Test Assistant Control File” on page 187.

```
     Defaults ListDsn=ATC.V1R5M0.SAMPLE.COBOLST(*),
              LoadMod=COB02M,
              FromObjDsn=ATC.V1R5M0.SAMPLE.OBJ,
              ToObjDsn=ATC.V1R5M0.SAMPLE.ZAPOBJ

     COBOL ListMember=COB02M

     Scope ExtProgram-Id=COB02M

     Variable Name=JULIAN-DATE
*                 set JULIAN-DATE using Data Warping
         Warp Action=Set, Value=0099365,
              Datatype=Zoned,Unsigned,Stmts=(87)

         Coverage Stmts=(94) // read JULIAN-DATE after it is warped and modified

     Variable Name=CURR-DATE
*                 set CURR-DATE using Data Warping
         Warp Action=Set, Value=00991231,
              Datatype=Zoned,Unsigned,Stmts=(97)

         Coverage Stmts=(103) // read CURR-DATE after it is warped

     Variable Name=YEAR4
*                 increment YEAR4 by 3 using Data Warping
         Warp Action=Increment,Value=3,
              Datatype=Zoned,Unsigned,Stmts=(106)

         Coverage Length=4,
              Stmts=(114) // read YEAR4 after it is warped and modified

     Variable Name=YEAR IN YEAR-BY-FIELD IN DATE-BY-FIELD
*                 decrement YEAR by 1 using Data Warping
         Warp Action=Decrement, Value=1,
              Datatype=Zoned,Unsigned,Stmts=(106)

     Variable Name=YEAR2
         Coverage Length=2,
              Stmts=(117) // read YEAR2 after it gets the 2 digit
*                           year from CURR-DATE decremented by 1

     Variable Name=(BEGIN-DATE In LOAN)
         Coverage Length=8,
              NAME               // read BEGIN-DATE of LOAN
*                         structure by NAME

         Coverage Length=8,
              FULL               // read BEGIN-DATE of LOAN structure
*                         by FULL (when LOAN referenced)

     Variable Name=JULIAN-DATE
         Coverage Length=7,
               Stmts=(143) // read initialization of INC-DATE

     Variable Name=J-DAY IN J-DATE
         Coverage Length=3,ReadEvery=100,
              MaxSave=5,Stmts=(153) // read J-DAY in loop
*                         every 100 times maximum of 5 times
```

Figure 56. Control File for COB02x

Create Setup JCL

Before test cases can be executed on the COB02x test program, UTA must insert breakpoints into the test program. UTA does this using the Setup JCL.

When you execute the Setup JCL, the UTA Setup program analyzes the assembler listings and creates a table containing breakpoint data (address, op code, and so on). Invalid instruction op codes are inserted for the instructions at the breakpoints in the object modules. You then link these modified object modules into a modified COB02x load module for UTA to use.

To create the Setup JCL:

1. Select option `1` from the `ATC Primary Option Menu`.

   The `Coverage, Distillation and Unit Test Assistant` panel is displayed.

2. Select option `2`.

   The `Create JCL for Setup` panel is displayed. You create the JCL for the setup of COB02x from this panel. All of the default values on the panel are correct for the COB02x example except for possibly the `Enable UTA` and the `Program Name` fields.

3. Ensure that `Enable UTA` is set to `YES`, that `Enable DA` is set to `NO`, and that the entry in the `Program Name` field is correct.

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 identified on the panel using member name SCOB02x.

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

For more detailed information, see “CA, DA, and UTA Setup” on page 231.

Create JCL to Start a Monitor Session

To create the JCL to start a monitor session:

1. Select option `1` from the `ATC Primary Option Menu`.

   The `Coverage, Distillation and Unit Test Assistant` panel is displayed.

2. Select option `3`.

   The `Create JCL to Start the Monitor` panel is displayed. You create the JCL for the UTA execution of COB02M from this panel.

3. Ensure that `Enable UTA` is set to `YES`, that `Enable DA` is set to `NO`, and that the entry in the `Program Name` field is correct.

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 identified on the panel using member name XCOB02x.

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

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

For more detailed information, see “Monitor Execution” on page 245.

Create JCL for a Variable Report

JCL is required to create a variable report. To create the variable report JCL:

1. Select option `1` from the `ATC Primary Option Menu`.
2. Select option `6`.

   The `Coverage, Distillation and Unit Test Assistant` panel is displayed.

   The `Unit Test Report` panel is displayed.
3. Change the Program Name to `COB02x`, and then select option `1`.

   Informational messages are written to your screen as the JCL is created. The created JCL is put into the JCL library identified on the panel using member name VCOB02x.
4. Press the End key (PF3) to exit the panel.

Create JCL to Link the Modified Object Modules

You must link the modified object modules (modified by the Setup step) into an executable program for testing. You can use the normal JCL that links your program, but be sure you use the object module library that contains the modified object modules. Sample JCL to link the COB02x example is provided in *hi_lev_qual*.V1R5M0.SAMPLE.JCL(LCOB02x).

Create JCL to Run the GO Step

You can use the normal JCL that executes your program, but be sure to specify the load module library that contains the link-edited modified object modules. Sample JCL to execute the GO step for the COB02x example is provided in *hi_lev_qual*.V1R5M0.SAMPLE.JCL(GCOB02x).

Execute the JCL

When you have created all of the COB02x JCL, you can run the COB02x summary example by executing the following functions in the order listed.

To see a flow diagram of these steps, see Figure 53 on page 168.

1. SCOB02x[33]

   Performs the setup step. All JCL steps should complete with condition code 0.
2. LCOB02x[34]

   Links the object modules that have been modified with breakpoints in the Setup step into the COB02*x* load module.
3. XCOB02x[33]

   Starts a monitor session. For UTA, a program continuously runs to write variable data to disk. The JCL does not complete until your session is stopped by step 6 on page 176.

[33] JCL created from the panels and put into the JCL library.

[34] JCL supplied with the installation materials in *hi_lev_qual*.V1R5M0.SAMPLE.JCL. (Sample JCL for all of the steps can be found in this partitioned data set [PDS].)

4. GCOB02x[34]

   Runs sample program COB02x. COB02x runs to completion with condition code 0.

5. CASTATS[35]

   Displays statistics using the CASTATS command. You should see a nonzero EVNTS count on the TOTALS line. (This is an optional step for illustrative purposes.)

6. CASTOP[35]

   Stops the monitor session. Unit Test Assistant writes the variable data to disk.

7. VCOB02x[33]

   Creates the variable reports for COB02x. The reports are in data sets proj_qual.COB02x.VARID and proj_qual.COB02x.VARDATA.

Multiple Compile Unit Test Case (COB01x)

The COB01M, COB012, and COB01O test cases are examples of reading variables from multiple compile units. The control file is shown in Figure 59 on page 180, the monitored variable report (MVR) is shown in Figure 57 on page 178, and the variable data report (VDR) is shown in Figure 58 on page 178.

In the COB01x test cases, variables are monitored in two of three compile units.

In COB01AM:

- The variable TAPARM1 is monitored on all statements in which it is referenced.
- Variable 2 is CITY In LOC-ID In TASTRUCT. It is monitored on all statements in which it is referenced by the level name CITY.
- Variable 3 is STATE In LOC-ID In TASTRUCT. It is monitored on all statements in which it is referenced by the level name or any of the qualifier names, since the FULL option was used in the read line in the control file.

In the program COB01BM, which is nested in the program COB01AM, the next variable monitored is TBPARM2. It is read only once, the first time statement #113 is executed. In the program COB01CM, variable 5 is TCPARM1. It is read once at each statement in which it is referenced. In COB01DM, no variables are monitored.

The MVR and VDR are the same regardless of the compiler that was used for the programs under test.

COB01M is a COBOL for MVS & VM test case, COB012 is a VS COBOL II test case, and COB01O is an OS/VS test case.

---

[35] Monitor commands issued from either the `Control the CA/DA/UTA Monitor` panel or the TSO command processor (ISPF option 6) by entering:

```
EX 'hi_lev_qual.V1R5M0.REXX(cacmd)'
```

where `cacmd` is the command issued (such as, CASTATS, CASTOP, and so on).

For COBOL programs, breakpoints are inserted into the object modules during Setup. These breakpoints are positioned so that the requested variable data can be read when the breakpoint interrupt occurs. When you are ready to test your program, you link the object modules that have been modified with breakpoints.

To produce the reports for COB01M, perform the following steps. **Steps 2a through 6 are described in more detail in topics that follow in this chapter.**

1. Compile the COBOL source you want to test. This produces listings that include the assembler statements needed by UTA. (This has already been done for the examples. For COBOL, the listings are in *hi_lev_qual*.V1R5M0.SAMPLE.COBOLST for the COBOL for MVS & VM and VS COBOL II compilers, and in *hi_lev_qual*.V1R5M0.SAMPLE.COSVSLST for the OS/VS compiler.)

   Make sure to use the compiler options specified in “Setup” on page 231.

2. Start the ATC ISPF dialog by entering the following from ISPF option 6:

   ```
   EX 'hi_lev_qual.V1R5M0.REXX(ATSTART)'
   ```

   The `ATC Primary Option Menu` is displayed.

   a. Edit the UTA control file.

      Verify that the control file includes the listings of the object modules you want to test.

   b. Create the setup JCL.

      Create the JCL that enables the Setup job to produce files containing breakpoint data, and to modify copies of your object modules by inserting breakpoints and information on variables you want to monitor.

   c. Create the start monitor session JCL.

      Create the JCL to start a monitor session.

   d. Create the variable report JCL.

      Create the JCL to produce the MVR and VDR reports after the COB01x test case has executed.

3. End the ATC ISPF dialog by pressing the End key (PF3) on the `ATC Primary Option Menu`.

4. Create the JCL to link the modified object modules.

   After the Setup step and before starting the monitor session, you must link the modified object modules into an executable program 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. Create the JCL to run the GO step.

   Create the JCL to run your program. Specify the same modified load module as in step 4.

6. Execute the JCL.

   Execute the created JCL files for COB01M in the correct order. (This order is shown in “Execute the JCL” on page 182.)

Figure 57 and Figure 58 show sample variable reports. For details about the information included in the reports, see “Unit Test Assistant Reports” on page 189.

```
* DATE: 09/11/1997
* TIME: 08:15:42
*
*
*CU Name:             ATC.V1R5M0.SAMPLE.COBOLST(COB01AM)
*External Program Id: COB01AM
*
*    CU    var    var   data   read   read   read   read   read   prog      var
* numbr     ID   mode   type  offst  lngth   freq    max  stmts     id      name
*-------------------------------------------------------------------------------------------------
      1      1   I      C         0      2      1            60   COB01AM   01 TAPARM1
                                                              61
      1      2   I      C         0      3      1            54   COB01AM   01 TASTRUCT 02 LOC-ID 03 CITY
      1      3   I      C         0      2      1            46   COB01AM   01 TASTRUCT 02 LOC-ID 03 STATE
                                                              49
                                                              53
      1      4   I      C         0      2      1      1    113   COB01BM   01 TBPARM2
*
*
*CU Name:             ATC.V1R5M0.SAMPLE.COBOLST(COB01CM)
*External Program Id: COB01CM
*
*    CU    var    var   data   read   read   read   read   read   prog      var
* numbr     ID   mode   type  offst  lngth   freq    max  stmts     id      name
*-------------------------------------------------------------------------------------------------
      2      5   I      C         0      2      1      1     37   COB01CM   01 TCPARM1
                                                              39
*
*
*CU Name:             ATC.V1R5M0.SAMPLE.COBOLST(COB01DM)
```

Figure 57. MVR for COB01M

```
* DATE: 12/12/1998
* TIME: 05:56:28
*
*
*    seq      CU     var      at         data
*  numbr   numbr      ID    stmt  stat   read
*-----------------------------------------
       1       1       3      46
       2       1       3      49         IL
       3       1       3      53         IL
       4       1       2      54         SPR
       5       1       1      60         05
       6       1       1      61         05
       7       2       5      37         05
       8       1       4     113         00
       9       1       1      60         04
      10       1       1      61         04
      11       1       1      60         03
      12       1       1      61         03
      13       1       1      60         02
      14       1       1      61         02
      15       1       1      60         01
      16       1       1      61         01
      17       1       1      60         00
```

Figure 58. VDR for COB01M

Edit the UTA Control File

UTA uses assembler listings to determine where to insert breakpoints to monitor variables. You supply the names of the listing files and data on the variables to read in each listing in the UTA control file (CACTL).

UTA uses the assembler listings produced by the COBOL compiler. Make sure to use the compiler options specified in "Setup" on page 231.

The CACTL control file for the COBOL one compile unit example is named COB01M (see Figure 59 on page 180).

To edit the CACTL for the COB01M one compile unit example:

1. Select option `1` from the `ATC Primary Option Menu`.

   The `Coverage, Distillation and Unit Test Assistant` panel is displayed.

2. Select option `1`.

   The `Work with the CA/DA/UTA Control File` panel is displayed.

3. Enter option `1` and specify the following:

Use Program Name for File Name	`YES`
Program Name	`COB01M`
Listing Type	`COBOL`

4. Press Enter.

   An ISPF edit session for the UTA control file you requested is displayed.

   The data in the control file consists of the following types of statements:

   DEFAULTS Specifies default values for operands not specified on subsequent COBOL statements.

   COBOL Specifies the data sets containing the compiler listing, object module, instrumented object module, and load module for the program that is to be tested.

   SCOPE Specifies the scope of the program in which the variables to be tested are defined.

   VARIABLE Specifies the name of the variables to be tested.

   COVERAGE Specifies detailed information about the type of coverage testing to be performed on a particular variable.

   If the control file you requested did not previously exist, it is created with comments in it to help you enter the appropriate information in the fields.

   If you want to use the shipped sample CACTL member as a template for your control file, delete the existing lines in the member and copy in *hi_lev_qual*.V1R5M0.SAMPLE.CACTL(COB01M). Change ATC. to *hi_lev_qual*. for all occurrences. Change the value of the ToObjDsn operand to proj_qual.ZAPOBJ. If you have recompiled the samples into proj_qual data sets, you will also have to edit the ListDsn and FromObjDsn keyword operands to point to your proj_qual listing and object data set names.

   The control file shown in Figure 59 on page 180 is the control file for COB01M. For a description of each of these lines, see "Editing the Unit Test Assistant Control File" on page 187.

5. Verify that the data set names are correct for your installation.
6. Press the End key (PF3) to exit the edit session.

For more detailed information about the control file, see “Editing the Unit Test Assistant Control File” on page 187.

```
*
* Cobol Example
*
* Statements required for coverage and unit test
*
               Defaults ListDsn=ATC.V1R5M0.SAMPLE.COBOLST(*),
                        LoadMod=COB01M,
                        FromObjDsn=ATC.V1R5M0.SAMPLE.OBJ,
                        ToObjDsn=ATC.V1R5M0.SAMPLE.ZAPOBJ

COB01AM:       COBOL ListMember=COB01AM
COB01CM:       COBOL ListMember=COB01CM
COB01DM:       COBOL ListMember=COB01DM

*
* Statements required for unit test
*

COB01AM_S:     Scope COBOL=COB01AM,ExtProgram-Id=COB01AM
COB01AM_S_B:   Scope COBOL=COB01AM,ExtProgram-Id=COB01AM,
                           NestedProgram-Id=COB01BM
COB01CM_S:     Scope COBOL=COB01CM,ExtProgram-Id=COB01CM

TAPARM1:       Variable Scope=COB01AM_S,Name=(TAPARM1)
CITY:          Variable Scope=COB01AM_S,Name=(CITY In LOC-ID In TASTRUCT)
STATE:         Variable Scope=COB01AM_S,Name=(STATE In LOC-ID In TASTRUCT)
TBPARM2:       Variable Scope=COB01AM_S_B,Name=(TBPARM2)
TCPARM1:       Variable Scope=COB01CM_S,Name=(TCPARM1)

               Coverage Variable=TAPARM1,Length=2,NAME // read TAPARM1
*                                       wherever it occurs (by NAME)

               Coverage Variable=CITY,Length=3,NAME   // read CITY In
*                                                           TASTRUCT

               Coverage Variable=STATE,Length=2,FULL  // read STATE In
*                                  STRUCT whether directly referenced or
*                                  via structure (FULL)

               Coverage Variable=TBPARM2,Length=2,MaxSave=1,Stmts=(113)
*                                  in COB01BM read TBPARM2 on line 113 only,
*                                  only once

               Coverage Variable=TCPARM1,Length=2,MaxSave=1,NAME // in
*                                       COB01CM read TCPARM1 only once
```

Figure 59. Control File for COB01M

Create Setup JCL

Before test cases can be executed on the COB01M test program, UTA must insert breakpoints into the test program. UTA does this using the Setup JCL.

When you execute the Setup JCL, the UTA Setup program analyzes the assembler listings and creates a table containing breakpoint data (address, op code, and so on). Invalid instruction op codes are inserted for the instructions at the breakpoints in the object modules. You then link these modified object modules into an executable COB01M load module for UTA to use.

To create the Setup JCL:

1. Select option 1 from the ATC Primary Option Menu.

   The Coverage, Distillation and Unit Test Assistant panel is displayed.

2. Select option 2.

   The Create JCL for Setup panel is displayed. You create the JCL for the setup of COB01M from this panel. All the default values on the Create JCL for Setup panel are correct for the COB01M example, except for possibly the Enable UTA and the Program Name fields.

3. Ensure that Enable UTA is set to YES, that Enable DA is set to NO, and that the Program Name field is correct.

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 identified on the panel using member name SCOB01M.

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

For more detailed information, see “CA, DA, and UTA Setup” on page 231.

Create JCL to Start a Monitor Session

To create the JCL to start a monitor session:

1. Select option 1 from the ATC Primary Option Menu.

   The Coverage, Distillation and Unit Test Assistant panel is displayed.

2. Select option 3.

   The Create JCL to Start the Monitor panel is displayed. You create the JCL for the UTA execution of COB01M from this panel.

3. Ensure that Enable UTA is set to YES, that Enable DA is set to NO, and that the entry in the Program Name field is correct.

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 identified on the panel using member name XCOB01M.

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

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

For more detailed information, see “Monitor Execution” on page 245.

Create Report JCL

To create the variable report JCL:

1. Select option 1 from the ATC Primary Option Menu.

   The Coverage, Distillation and Unit Test Assistant panel is displayed.

2. Select option 6.

   The Unit Test Report panel is displayed.

3. Change the Program Name to `COB01M`, and then select option `1`.

   Informational messages are written to your screen as the JCL is created. The created JCL is put into the JCL library identified on the panel using member name VCOB01M.

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

Create JCL to Link the Modified Object Modules

You must link the modified object modules (modified by the Setup step) into an executable program for testing. You can use the normal JCL that links your program, but be sure you use the object module library that contains the modified object modules. Sample JCL to link the COB01M example is provided in *hi_lev_qual*.V1R5M0.SAMPLE.JCL(LCOB01M).

Create JCL to Run the GO Step

You can use the normal JCL that executes your program, but be sure to specify the load module library that contains the link-edited modified object modules. Sample JCL to execute the GO step for the COB01M example is provided in *hi_lev_qual*.V1R5M0.SAMPLE.JCL(GCOB01M).

Execute the JCL

When you have created all of the COB01M JCL, you can run the COB01M summary example by executing the following functions in the order listed.

To see a flow diagram of these steps, see Figure 53 on page 168.

1. SCOB01M[33]

   Performs the Setup step. All JCL steps should complete with condition code 0.

2. LCOB01M[34]

   Links the object modules that have been modified with breakpoints in the Setup step into the COB01M load module.

3. XCOB01M[33]

   Starts a monitor session. For UTA, a program continuously runs to write variable data to disk. The JCL does not complete until your session is stopped by step 6.

4. GCOB01M[34]

   Runs sample program COB01M. COB01M runs to completion with condition code 0.

5. CASTATS[35]

   Displays statistics using the CASTATS command. You should see a nonzero EVNTS count on the TOTALS line. (This is an optional step for illustrative purposes.)

6. CASTOP[35]

   Stops the monitor session. UTA writes the variable data to disk.

7. VCOB01M[33]

   Creates the reports for COB01M. The reports are in data sets proj_qual.COB01M.VARID and proj_qual.COB01M.VARDATA.

Unit Test Assistant Read and Warp Descriptions

This chapter describes what variables can be read or warped by Unit Test Assistant (UTA) and when reading and warping takes place.

Description of the Variable Read Operation

The control file identifies the variables to read, the statements on which to read them, the number of times to read them, where variables are to be warped, and so on. See “Editing the Unit Test Assistant Control File” on page 187 for more information.

Where a Variable Is Read

A variable is read when the statement in which it is located is executed, and the maximum reads for that statement are not exceeded. A variable is read before the statement is executed. If the statement is a looping or iterative statement (such as a PERFORM), the variable will only be read before the first iteration.

Which COBOL Storage Areas Can Be Read

UTA does not support CICS® routines compiled with the OS/VS COBOL compiler. Reentrant COBOL routines are only supported for compilers that support the RENT compiler option.

UTA uses the compiler listings that contain the assembler code and cross-reference to determine where variables reside in storage. The variables to be read must appear in the cross-reference and reside in one of the following areas:

- Working storage
- Variably located data
- Linkage
- File

Consider the following when identifying COBOL storage areas that can and cannot be read:

- UTA can only read the first occurrence of a variable that is defined multiple times by way of the OCCURS clause.
- UTA cannot read variables defined in a LOCAL storage area. The LOCAL storage area is an IBM extension available in the COBOL for MVS & VM compiler. UTA also cannot read linkage variables in a program in which local variables are defined.
- If a variable is the object of a REDEFINE, all names should be monitored to ensure full coverage (that is, each name should have an entry in the control file).
- If a variable is declared as EXTERNAL, and is used in more than one level of nested programs, the control file must include entries for the variable for each internal program name in which it is used for full coverage.

© Copyright IBM Corp. 1997, 1999

- If a variable is the object of a USING statement (passed to another program as a parameter), entries must be included in the control file for the variable in the calling program and for the linkage variable in all called programs that use it in order to ensure full coverage.
- UTA cannot monitor COBOL variables accessed through pointer references.

How Much Data Can Be Read

One read of one variable can save up to 126 bytes of data. If the data item you want to save is longer than 126 bytes, you can do multiple reads with appropriate offsets and lengths defined.

Variable data is kept in storage buffers while your program is running, and periodically the data is written to disk. Two buffers are used: one is written to disk while the other is being used. It is possible that the buffers may be full when new data is to be written to the buffer. The size of each buffer is 65536 bytes. Each data entry contains the variable data plus eight bytes of other data.

If a buffer overrun occurs and causes loss of new variable data, you will see a statement similar to the following in the VARDATA file of data:

```
11915        1        1        38     KYLEX  3722AIX.
11916        1        1        38     KYLEX  3723AIX.
******** !!! BUFFER OVERFLOW !!! ********
11917        1        1        38     KYLEX  0000MVS.
```

If a buffer overflow occurs, the program that creates the VDR report of variable data (stepname EXVAR) completes with a return code of 4.

To reduce buffer overflows, try the following:

1. If you are reading a variable that is in a loop, only read it at intervals (for example, once every 10th time or 100th time).
2. Reduce the size of each read (for example, the variable may be 100 bytes long, but you may just need to look at the first 10 bytes).
3. Reduce the number of reads (for example, a variable may occur in hundreds of lines in your program, but you may only be interested in its value at a few locations).

Description of the Variable Warp Operation

The control file identifies the variable to warp, the statement that contains the variable to warp, the warp value, and so on. For more information about the contents and function of the control file, see “Editing the Unit Test Assistant Control File” on page 187.

What COBOL Variables Can Be Warped

Any COBOL variable that can be read by UTA can be warped. For more information, see “Which COBOL Storage Areas Can Be Read” on page 183.

When a COBOL Variable Is Warped

A COBOL variable is warped before the execution of the statement. For example, suppose a file is read into a buffer via this COBOL READ statement:

```
000226            READ  QSAMIN  INTO  WS-INPUT-RECORD
```

A warp of WS-INPUT-RECORD on line 226 would be performed before the file read was complete. Therefore, to warp the data in WS-INPUT-RECORD, the warp should be done on the statement following WS-INPUT-RECORD.

What PL/I Variables Can Be Warped

Only PL/I file input buffers can be warped. In the following example, any field in IN_RECORD can be warped on statement 38. Any other variables (including IN_RECORD located on other statements) cannot be warped.

```
38    2  0        READ  FILE(QSAMIN)  INTO(IN_RECORD);
```

PL/I warps can only be performed on the statement where the file read occurs.

When a PL/I Variable Is Warped

The warp is performed after the file read is complete, and the new data is in the buffer.

Reading and Warping on the Same Statement

If a variable or buffer is read and warped on the same statement, the ordering of the read and warp is indeterminate. If you want to read a variable or buffer that has been warped, do the read on the following statement.

Editing the Unit Test Assistant Control File

This chapter describes the function of the control file (CACTL) used by UTA. The CACTL contains information that tells UTA what compile units to analyze and the values of which variables to record. CA, DA, and UTA share the CACTL file. See “CA, DA, and UTA Control File” on page 209 for a complete description of this control file. This chapter only explains how to use the control file with Unit Test Assistant.

Contents of the Control File

The control file defines the variable to be recorded. The following describes the contents of the control file as it is used by Unit Test Assistant. See “Contents of the Control File” on page 211 for a description of the syntax of the control file.

- The COBOL statement specifies the following:
  - The source language (COBOL)
  - The data containing the compiler listing file
  - The name of the load module containing the program
  - The data set containing 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
- The SCOPE statement specifies the following:
  - The PROGRAM-ID of the external program in which variables are defined. The external program ID is the first program ID in the listing file.
  - The PROGRAM-ID of the nested (internal) program in which the variables are defined. This operand is required only if variables are defined within a nested program.
- The VARIABLE statement specifies the following:
  - The scope in which the variable is defined
  - The name of the variable
- The COVERAGE statement specifies attributes and coverage details for a specific variable.
- The WARP statement specifies details about how warping of a specific variable or buffer on a specific statement (or statements) is to be performed.

© Copyright IBM Corp. 1997, 1999

Examples

The examples show how you might construct a UTA control file (1) for capturing and logging COBOL variables and (2) for warping a PL/I input buffer.

1. The following example shows how variables in structures are read:

```
COBOL    ListDsn=Atc.V1R5M0.Sample.CoboLst(COB01AM),LoadMod=COB01M,
         FromObjDsn=Atc.V1R5M0.Sample.Obj,ToObjDsn=Atc.V1R5M0.Sample.ZapObj,ObjMember=COB01AM
Scope    ExtProgram-Id=COB01AM
Variable Name=(CITY In LOC-ID In TASTRUCT)
Coverage MaxSave=10,Name
Variable Name=(STATE In LOC-ID In TASTRUCT)
Coverage ReadEvery=2,Full
```

Figure 60. Sample UTA Control File. The variable CITY In LOC-ID In TASTRUCT is read up to a maximum of 10 times, for each statement where CITY is accessed by name. The variable STATE In LOC-ID In TASTRUCT is read every second time a statement is executed on which it is accessed through any level of the structure TASTRUCT. There is no maximum number of reads for this variable.

The example in Figure 60 is based on the following COBOL declarations:

```
 In COB01AM:
     WORKING-STORAGE SECTION.
     01 TAPARM1      PIC 99 VALUE 5.
     01 TAPARM2      PIC 99 VALUE 0.

     01 TASTRUCT.
       05 LOC-ID.
         10 STATE    PIC X(2).
         10 CITY     PIC X(3).

 In COB01BM:

     WORKING-STORAGE SECTION.
     01 TBPARM1      PIC 99 VALUE 5.
     01 TBPARM2      PIC 99 VALUE 0.
```

Figure 61. COBOL File Definitions

For an example of a control file for warping COBOL variables, see Figure 56 on page 173.

2. Figure 62 is an example of a control file for warping a PL/I input buffer:

```
PL/I ListDsn=ATC.V1R5M0.TEST.PLILST(PLI12M),
     LoadMod=PLI12M,
     FromObjDsn=ATC.V1R5M0.TEST.OBJ,
     ToObjDsn=ATC.V1R5M0.TEST.ZAPOBJ
     Scope PROCEDURE=PLI12M
       Variable Name=sfile

Warp Position=1,Length=4,Stmts=(33),Action=INCREMENT,VALUE=10,
     DataType=zoned,signed
```

Figure 62. Control File for Warping a PL/I Input Buffer

In the following statement, the field in buffer `si` (position 1 for a length of 4) is warped by incrementing it by 10 after each file read. This position must hold a zoned, signed integer.

```
33   1  0    read  file(sfile)  into(si);
```

Unit Test Assistant Reports

This chapter describes the following reports, which UTA creates for the monitored variables:

- Monitored variables report (MVR). A table of the monitored variables defined in the control file and their read specifications.
- Variable data report (VDR). A table of data for each time a monitored variable is read during the program execution.
- Combined variable data report (CVDR). An alternative version of the VDR, which includes the fully-qualified variable names for each read along with the read data.

Monitored Variables Report (MVR)

The monitored variables report lists the data for all monitored variables as defined in the control file. For an example of an MVR, see Figure 63 on page 191. For a description of the variable information in the control file, see "Editing the Coverage Assistant Control File" on page 81. All data in the MVR comes from the control file.

Each monitored compile unit (CU) is listed in the report. The `CU NAME` is the listing file name that was used in the Setup step. The `External Program ID` is the external program ID of your COBOL program.

Each monitored variable has one entry in this report. Each entry contains the following data:

CU numbr (compile unit number)
: The sequential number of the CU containing this monitored variable. The VDR lists the CU Number of the variable.

Var ID (variable ID)
: The sequential number of this monitored variable. The VDR identifies the variable by this variable ID.

Var mode (variable mode)
: The mode for this variable, which can be either of the following:

  I Internal (UTA variable read)
  F File (DA file read)

Data type
: Format in which data is displayed in the report:

  C Character representation of data
  X Hexadecimal representation of data

Read offst (read offset)
: The offset within the variable data area to read. For example, a variable might be defined as 132 characters in length, but you only want to read a serial number field that is at offset 40 within the field. This entry is 0-based (that is, 0 is the offset of the first byte).

© Copyright IBM Corp. 1997, 1999

Read lngth (read length)

The length of the variable data item to read. This may be the length of the entire data item, or some subset of it. A maximum of 126 bytes can be read from one variable. If more than 126 bytes is desired, you can have two (or more) reads of the same variable. All 256 bytes of CUST-REC can be read by the control file entries:

```
Definition of CUST-REC:
01 CUST-REC   PIC X(256)

Control File entries:
  VARIABLE NAME=(CUST-REC)
  COVERAGE Offset=0,Length=126,Name
  VARIABLE NAME=(CUST-REC)
  COVERAGE Offset=126,Length=126,Name
  VARIABLE NAME=(CUST-REC)
  COVERAGE Offset=252,Length=4,Name
```

The first variable/coverage pair reads the first 126 bytes of CUST-REC, the second reads the next 126 bytes, and the third reads the last 4 bytes.

Read freq (read frequency)

How often to read the variable when the statement in which the variable is located is executed. For example, if read frequency is one, the variable is read and the data is saved on each execution of the monitored statement. If the read frequency is five, it is read every fifth time the statement is executed.

Read max (Read maximum)

The maximum number of times to read the variable. If blank, it is read on each execution of monitored statements. You may only be interested in the value of the variable the first time a statement is executed, of the first *n* times the statement is executed; therefore, this is useful to reduce the number of reads (and consequently the amount of variable data saved).

Read stmts (read statements)

All statement numbers where the UTA control file requested (either explicitly or implicitly) that the specified variable be monitored.

Prog ID (program ID)

The program ID (external or internal) that contains the variable. The same variable name may be used in different program IDs.

Var name (variable name)

The variable name contains all qualifying level names if this is a read from a structure. A two-digit level number (starting with `01`) precedes each level name:

```
01 Level1-name  02 level2-name  ...
```

The following example shows an MVR.

```
* DATE: 09/11/1997
* TIME: 08:15:42
*
*
*CU Name:             ATC.V1R5M0.SAMPLE.COBOLST(COB01AM)
*External Program Id: COB01AM
*
*    CU    var    var   data   read   read   read   read   read   prog      var
* numbr     ID   mode   type  offst  lngth   freq    max  stmts     id      name
*-------------------------------------------------------------------------------------------------------
      1      1   I      C         0      2      1            60   COB01AM   01 TAPARM1
                                                              61
      1      2   I      C         0      3      1            54   COB01AM   01 TASTRUCT 02 LOC-ID 03 CITY
      1      3   I      C         0      2      1            46   COB01AM   01 TASTRUCT 02 LOC-ID 03 STATE
                                                              49
                                                              53
      1      4   I      C         0      2      1      1    113   COB01BM   01 TBPARM2
*
*
*CU Name:             ATC.V1R5M0.SAMPLE.COBOLST(COB01CM)
*External Program Id: COB01CM
*
*    CU    var    var   data   read   read   read   read   read   prog      var
* numbr     ID   mode   type  offst  lngth   freq    max  stmts     id      name
*-------------------------------------------------------------------------------------------------------
      2      5   I      C         0      2      1      1     37   COB01CM   01 TCPARM1
                                                              39
*
*
*CU Name:             ATC.V1R5M0.SAMPLE.COBOLST(COB01DM)
```

Figure 63. MVR for COB01M

Variable Data Report (VDR)

The VDR is a table of variable data. Each row shows the data of one variable read at one statement number. Columns of the row (for example, CU numbr and var ID) are used to reference the characteristics and names of the variables in the MVR.

The columns in the VDR are:

Seq numbr (sequence number)
: A sequential number identifying the variable data row.

CU numbr (CU number)
: The CU number containing the variable in the MVR.

Var ID (variable ID)
: The variable ID number in the MVR.

At stmt (at statement)
: The statement number where the variable was read.

Stat (status)
This field is only used for data warp errors. For information about data warp errors, see “Errors During Data Warping.”

Data read
The variable data read at this statement number. The length of the data is specified by the `read lngth` column in the MVR. A maximum of 126 bytes can be read. Hexadecimal numbers are flagged with an *X* to the left of the number and are displayed as two hexadecimal digits for each byte represented.

The following example shows a VDR.

```
* DATE: 09/11/1997
* TIME: 08:15:43
*
*
*    seq      CU     var      at          data
*  numbr   numbr      ID    stmt  stat   read
*-----------------------------------------------
       1       1       3      46          IL
       2       1       3      49          IL
       3       1       3      53          KY
       4       1       2      54          LEX
       5       1       1      60          05
       6       1       1      61          04
       7       2       5      37          05
       8       1       4     113          00
       9       1       1      60          04
      10       1       1      61          03
      11       1       1      60          03
      12       1       1      61          02
      13       1       1      60          02
      14       1       1      61          01
      15       1       1      60          01
      16       1       1      61          00
      17       1       1      60          00
```

Figure 64. VDR for COB01M

Errors During Data Warping

A sign check is done if you increment or decrement a packed or zoned variable. The sign you specify for the variable must match the sign of the variable in storage before the warp is performed. If there is a sign mismatch, a read of the variable in storage is done. You can view the error information by running the Unit Test Assistant reports described in this chapter.

The reads of variables with sign mismatches have an error number in the status field:

4 Packed signed error. The variable was specified as a signed packed number, but the variable read in the user program was unsigned.

5 Packed unsigned error. The variable was specified as an unsigned packed number, but the variable read in the user program was signed.

6 Zoned signed error. The variable was specified as a signed zoned number, but the variable read in the user program was unsigned.

7 Zoned unsigned error. The variable was specified as an unsigned zoned number, but the variable read in the user program was signed.

Combined Variable Data Report (CVDR)

The CVDR, an alternative to the VDR, combines elements of the MVR and the VDR. Each row shows the data from one variable read at one statement number and enough information to uniquely identify the variable. The var-ID column is used to reference the variable read characteristics in the MVR.

The columns in the CVDR are:

var-ID (variable ID)
The variable ID number in the MVR.

CU-Name (compile unit name)
The name of the CU containing the variable.

prog-ID (program ID)
The name of the program containing the variable.

var-name (variable name)
The fully-qualified variable name.

stmt-num (statement number)
The statement number at which the variable data was read.

data
The variable data read at this statement number.

```
* DATE: 09/11/1997
* TIME: 10:55:54
*
*
* var-ID  CU-Name    prog-ID           var-name                          stmt-num    data
*-------------------------------------------------------------------------------------------
       3  COB01AM    COB01AM           STATE of LOC-ID of TASTRUCT             46    IL
       3  COB01AM    COB01AM           STATE of LOC-ID of TASTRUCT             49    IL
       3  COB01AM    COB01AM           STATE of LOC-ID of TASTRUCT             53    KY
       2  COB01AM    COB01AM           CITY of LOC-ID of TASTRUCT              54    LEX
       1  COB01AM    COB01AM           TAPARM1                                 60    05
       1  COB01AM    COB01AM           TAPARM1                                 61    04
       5  COB01CM    COB01CM           TCPARM1                                 37    05
       4  COB01AM    COB01BM           TBPARM2                                113    00
       1  COB01AM    COB01AM           TAPARM1                                 60    04
       1  COB01AM    COB01AM           TAPARM1                                 61    03
       1  COB01AM    COB01AM           TAPARM1                                 60    03
       1  COB01AM    COB01AM           TAPARM1                                 61    02
       1  COB01AM    COB01AM           TAPARM1                                 60    02
       1  COB01AM    COB01AM           TAPARM1                                 61    01
       1  COB01AM    COB01AM           TAPARM1                                 60    01
       1  COB01AM    COB01AM           TAPARM1                                 61    00
       1  COB01AM    COB01AM           TAPARM1                                 60    00
```

Figure 65. CVDR for COB01M

Creating Unit Test Report JCL Using the Panels

To display the Unit Test Report panel:

1. Select option 1 from the ATC Primary Option Menu.

   The Coverage, Distillation and Unit Test Assistant panel is displayed.
2. Select option 6.

   The Unit Test Report panel, shown in Figure 66, is displayed.

```
------------------------------ Unit Test 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  COB01M

  Variable Report Type            FULL   (Full/Combine)

  Input Files:
    Debug Table Dsn . . . 'YOUNG.TEST.COB01M.DBGTAB'
    Variable Table Dsn. . 'YOUNG.TEST.COB01M.VARTAB'

  JCL Library and Member:
    JCL Dsn . . . . . . . 'YOUNG.TEST.JCL.CNTL(Vxxxxxxx)'

  Output Files:
    Variable ID Dsn . . . 'YOUNG.TEST.COB01M.VARID'
      (* for default sysout class)
    Variable Data Dsn . . 'YOUNG.TEST.COB01M.VARDATA'
      (* for default sysout class)
```

Figure 66. Unit Test Report Panel

The panel's options and fields are as follows:

Generate
: Generate JCL from the parameters you have specified on the panel.

Edit
: Make changes to existing JCL.

Submit
: Submit for execution the JCL specified in the JCL Dsn field on this panel.

Use Program Name for File Name
: 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, enter YES.

  When you press Enter, the file names on the panel are changed automatically. Using the program name to construct the data set names is the normal UTA procedure.

Program Name

Name to use for UTA files if you enter `YES` in the `Use Program Name for File Name` field. Note that this 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.TEST.COB01M.BRKTAB'

- Partitioned data sets:

  `'proj_qual.file_type(program_name)'`

  For example: 'YOUNG.TEST.BRKTAB(COB01M)'

Variable Report Type

Specifies the parameter to be used for the report program (PRINTVAR). FULL produces the VDR and COMBINE produces the CVDR.

Debug Table Dsn

Specifies the name of the file containing the variables that UTA is to monitor.

Variable Table Dsn

Specifies a UTA work file containing intermediate results of information gathered when variables were monitored.

JCL Dsn

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

Note: 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 V*xxxxxxx*, where *xxxxxxx* is the first seven characters of the program name.

Variable ID Dsn

Specifies the name of the file to contain various information about the variables being monitored (MVR). This includes the location of the variables in the listing, the compilation unit, and so on. (Enter * for the default SYSOUT class.)

Variable Data Dsn

Specifies the name of the file to contain the final monitoring results (VDR). This is formatted output based on the variable table work file. (Enter * for the default SYSOUT class.)

Examples of Reports

The ATC installation package contains the following examples:

- COB01M (COBOL for MVS & VM example)
- COB012 (VS COBOL II example)
- COB01O (OS/VS COBOL example)
- COB02M (COBOL for MVS & VM example)
- COB022 (VS COBOL II example)
- COB02O (OS/VS COBOL example)

The sample reports are contained in the following data sets. The data set names are in the form *hi_lev_qual*.V1R5M0.SAMPLE.*Y(X)*, where *X* is the test case name (COB02M, COB01M, and so on) and *Y* is one of the following:

- VARID. The MVR report.
- VARDATA. The VDR report.

These reports were created from the JCL in *hi_lev_qual*.V1R5M0.SAMPLE.JCL. The report JCL member is named VCOB02M, VCOB01M, and so on.

Parameters for the PRINTVAR Program

One parameter is passed to the report program (PRINTVAR). The values are:

- FULL. Produces the standard VDR report as described in "Variable Data Report (VDR)" on page 191.
- COMBINE. Produces the CVDR report as described in "Combined Variable Data Report (CVDR)" on page 193.
- KEYS_ONLY. Produces a report containing some informational comments followed by lines containing only the `Data read` information (starting in column 2).
- BOTH. Produces a report similar to FULL, but with the data lines prefixed by `*` (indicating a comment) followed by lines containing only the `Data read` information (starting in column 2).

The MVR report is always produced.

Unit Test Assistant File Warping

File warping is the modification of variables (typically date variables) in program input files to simulate input conditions for testing. In the context of the Year 2000 problem, file warping could be used to modify dates in input files to post-Year 2000 values to perform Year 2000 testing of remediated programs.

The UTA file warp feature enables you to statically warp dates in input files. File warp can copy any QSAM or VSAM file and warp fields in the copied file for testing. Any zoned or packed numeric field can be incremented, decremented, or set. Any zoned, packed, or character field can be set to a common value. A useful implementation of this file warping capability might be clearing fields in test copies of production input files for privacy or security reasons.

File Warp Operation

The following figure illustrates the UTA file warping process:

Figure 67. UTA—Flow Diagram

The UTA file warping feature can perform the following types of warps:

- Warps on all records
- Warps on matching record types

The warp control file controls the warping of file fields.

© Copyright IBM Corp. 1997, 1999

For record type warps, a field is identified as the record type. A warp record is then defined to do the warp if the record type matches the record type field value supplied. Note how a file record is defined in the following example copy book:

```
01  EMPLOYEE-RECORD.
    03  EMPLOYEE_NAME                  PIC 20(X).
    03 SOCIAL_SECURITY_NUMBER          PIC 9(9).
    03  SPACE1                         PIC 1(X).
    03  HIRE_DATE
                05   YEAR              PIC 9(2).
                05   MONTH             PIC 9(2).
                05   DAY               PIC 9(2).
    03  SPACE2                         PIC 1(X).
    03  LAST_PROMOTION_DATE
                05   YEAR              PIC 9(2) PACKED-DECIMAL.
                05   MONTH             PIC 9(2) PACKED-DECIMAL.
                05   DAY               PIC 9(2) PACKED-DECIMAL.
    03  SPACE3                         PIC 1(X).
    03  CURRENT_LEVEL                  PIC 9(1).
    03  SPACE4                         PIC 1(X).
    03  CURRENT_SALARY                 PIC 9(7).
```

Figure 68. Copy Book Defining a File Record

The following warp control file could be used for the copy book in Figure 68:

```
01  EMPLOYEE-RECORD.
          03  EMPLOYEE_NAME          20 CHARACTER
          03  SOCIAL_SECURITY_NUMBER  9 ZONED = 0
          03  SPACE1                  1 CHARACTER
          03  HIRE_DATE
                    05   YEAR         2 ZONED = 01
                    05   MONTH        2 ZONED
                    05   DAY          2 ZONED
          03  SPACE2                  1 CHARACTER
          03  LAST_PROMOTION_DATE
                    05 YEAR           2 PACKED SIGNED
R1:           1                       2 PACKED SIGNED + 1
R1:           2                       2 PACKED SIGNED + 2
R1:          DEFAULT                  2 PACKED SIGNED - 1
                    05 MONTH          2 PACKED SIGNED
                    05 DAY            2 PACKED SIGNED
           03  SPACE3                 1 CHARACTER
1:         03  CURRENT_LEVEL             1 ZONED
           03  SPACE4                 1 CHARACTER
           03  CURRENT_SALARY         7 ZONED = 0
```

Figure 69. Example of a Warp Control File

For all records:

- The SOCIAL_SECURITY_NUMBER and CURRENT_SALARY are set to 0.
- The 2-digit zoned field YEAR in HIRE_DATE is set to 01.
- For records with CURRENT_LEVEL=1, 1 is added to the packed field YEAR in LAST_PROMOTION_DATE.
- For records with CURRENT_LEVEL=2, 2 is added to the packed field YEAR in LAST_PROMOTION_DATE.
- For records with any other CURRENT_LEVEL, 1 is subtracted from the packed field YEAR in LAST_PROMOTION_DATE.

File Warp Samples

The samples shipped with ATC include a file warp sample. This sample has the following files:

File warp control file:
hi_lev_qual.V1R5M0.SAMPLE.FWCTL(FWARP)
QSAM input file:
hi_lev_qual.V1R5M0.SAMPLE.FWARPIN
QSAM output file created as the result of the file warp:
hi_lev_qual.V1R5M0.SAMPLE.FWARPOUT

To create file warp JCL and execute this sample, complete the following steps:

1. Select option 1 from the `ATC Primary Option Menu.`

   The `Coverage, Distillation and Unit Test Assistant` panel is displayed.

2. Select option 6.

   The `Unit Test Assistant` panel is displayed.

3. Select option 2.

   The `Generate JCL for File Warping` panel is displayed.

```
----------------------- Generate JCL for File Warping ---------- Enter option
 Command ===>

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

 Enter END to Terminate

 Input Dsn . . . . . .  'ATC.V1R5M0.SAMPLE.FWARPIN'
   Volume Serial . . .            Unit . .          (If not cataloged)

 Output Dsn. . . . . .  'YOUNG.TEST.FWARPOUT'
   Volume Serial . . .            Unit . .          (If not cataloged)

 Use Identifier for File Name    YES (Yes|No)   Identifier  FWARP

 File Warp Control Dsn  'ATC.V1R5M0.SAMPLE.FWCTL(FWARP)'
   (If the File Warp Control Dsn is blank, the input file is copied unchanged.)

 JCL Library and Member:
   JCL Dsn . . . . . .  'YOUNG.TEST.JCL.CNTL(FFWARP)'



 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
 F10=Actions  F12=Cancel
```

Figure 70. Generate JCL for File Warping Panel

4. Specify the following on the `Generate JCL for File Warping` panel:
   - The '*hi_lev_qual*.V1R5M0.SAMPLE.FWARPIN' in the `Input Dsn` field.
   - The name you would like to use for the new warped copy in the `Output Dsn` field.
   - `YES` in the `Use Identifier for File Name` field.
   - `FWARP` in the `Identifier` field.

- '*hi_lev_qual*.V1R5M0.SAMPLE.FWCTL(FWARP)' in the `File Warp Control Dsn` field.
- The data set and member name you want to use for the JCL name in the `JCL Dsn` field.

5. Press Enter to return to the command line.
6. Select option 1.

   Informational messages are written to your screen as the JCL is created. The JCL is created in the JCL library member you specified.
7. If you want to edit the JCL you just generated, select option 2.

   An edit session is opened for the JCL data set.
8. Select option 3.

   Your JCL will be submitted to run file warping.

The output file is created with fields warped as defined in the file warp control file. The record structure and file warp control file contents are the same as those used in the example in "File Warp Operation" on page 197.

File Warp Control File Syntax

The file warp control file contains the following types of lines:

- Field definition
- Record type definition

The warp control file structure allows copy books to be the basis for the definitions. For structures, the lowest level variables have to have a length and data type. Higher level variables can have length/data type information, or this information can be left blank. If provided, the keyword LEVEL must be present. The entire copy book does not need to be used for the control file. You can just select the fields to be warped, and use filler field names to define the spaces between warped variables.

You might use a definition similar to the following if you were only interested in warping 2 fields and you calculated the offsets to them. The offsets are supplied in the fill fields.

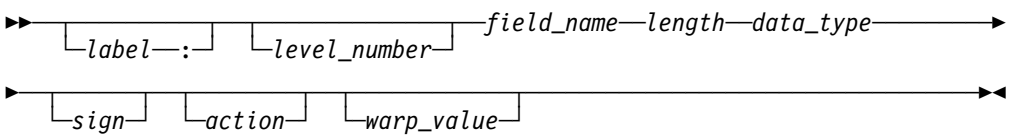
```
01      master_file
        03              fill       20 character
        03       warp_field1        2 zoned + 28
        03              fill       40 character
        03       warp_field2        4 packed signed - 28
```

Field Definition

A field definition is the lowest level of a structure (copy book). A field can be warped or it can just be present as a filler. The following is an example of a field definition:

```
          03           BEGIN-DATE             8 Packed Signed  + 2
          03           SKIP_UNTIL_WARP      100 Character
           1:   05     BIRTH-DATE             4 Binary + 100
                       LAST_NAME             16 Character = ABCDEFGHIJLMNOPS
                       FIRST_NAME             6 Character = "      "
                       birthday               4 Binary   = '0000'x
```

The syntax for field definition statements is as follows:

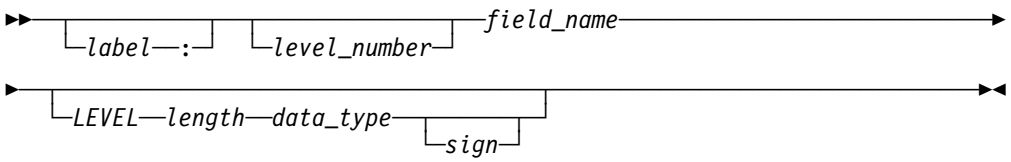


Group Level Definition

A group level definition is a higher level structure that contains field definitions. The following is an example of a group level definition that is not warped:

```
      // length data_type optional:
   03      EMP-HIRE-DATE
      // If length and data_type are present,
      // keyword LEVEL must be supplied
   03      EMP-HIRE-DATE     LEVEL 6  Zoned
      //   followed by structure definition:
         05     YY          2 Zoned
         05     MM          2 Zoned
         05     DD          2 Zoned
```

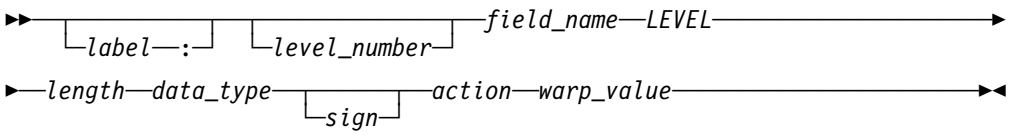
The syntax for group-level field definition statements that are not warped is as follows:



The following is an example of a group level definition that is warped:

```
 * length, data_type and keyword LEVEL must be present if warped
    03     EMP-HIRE-DATE    LEVEL 6  character     =  991231
       05    YY       2 Zoned
       05    MM       2 Zoned
       05    DD       2 Zoned
```

The syntax for group-level field definition statements that are warped is as follows:



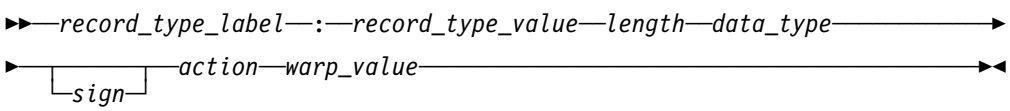
Record Type Definition

Record type definitions define record type fields used to selectively perform warping based on the contents of the record type field:

```
  r1:      1      2 packed + 1
  r1:      2      2 packed - 1
  r6:  '00001'x   2 Zoned   signed = 99
  r99:  DEFAULT   4 character  = ABCD
```

Record type definitions follow the field definition that they warp. The record type label must end with a number that is defined in the warp file as a field definition label.

The syntax for record type definition statements is as follows:



Positional Parameter Definitions

All parameters are positional and separated by at least one blank. Parameters are not case sensitive.

label

A label is a positive integer followed by a colon (:). It is used on field definition statements to define them as record types. It is recommended (but not required) that label numbers start with one and be sequential. The following example shows a label parameter definition:

```
 1:
99:
-5:     // illegal: negative integer
4.5:    // illegal: not an integer
```

level_number

The level_number is an optional positive integer typically used in structures. It is ignored by file warping and can be omitted. The following example shows a level number parameter definition:

```
03      EMP-DATE          2 packed
05      BIRTHDAY          6 zoned level
-9      EMP-DATE          // illegal: negative
```

field_name

The name of the field. It can be any characters (it need not be a legal compiler name) but must be present. The following example shows a field name parameter definition:

```
05     LAST_NAME   20 character
 1:    @%$&#(       4 zoned
03     fill        50       // Fill to define record length
                             // until field to be warped
03     4 packed + 1         // error: no field name
```

length

The length of the field:

Packed Number of digits. File warp translates the number to bytes needed for the field: (digits + 2)/2

Binary Number of digits. File warp translates the number to bytes needed for the field:

1 to 4	2 bytes
5 to 9	4 bytes
10 to 18	8 bytes

Zoned Number of digits. One byte per digit.

Character Number of characters. One byte per character.

Any other data type cannot be warped. It must be defined as a character field of the proper length. The length must be a positive integer.

data_type One of the following types of fields:

- Packed
- Zoned
- Binary
- Character

The above types follow COBOL definitions. Only the first character of the data_type is examined. For example, P may be used for packed.

sign For packed and zoned numbers, the sign (Signed or Unsigned) of the field. For binary and character fields, the sign (if supplied) is ignored. If supplied, the sign must follow the data_type. If omitted, the default is Unsigned. Signed can be abbreviated to S; Unsigned can be abbreviated U. To add or subtract from zoned and packed fields, the sign in the control file must match the sign of the field in the record or an error will occur.

action One of the following warp actions to take:

+ Add the warp_value to the field.
– Subtract the warp_value from the field.
= Replace the field with the warp_value.

For character fields, only = is valid.

warp_value The warp value used in conjunction with the action. For + or –, the warp_value must be a positive integer. For =, the field can be a:

- Hexadecimal value enclosed in quotes and followed by *X*:
  '0001'X.
- Character string. If the string contains spaces, enclose in quotes. For example: 'AB CD'
  Either single quotes (') or double quotes (") can be used.

record_type_label The definition of a record type for record type warping. It must be of the format: the character *r* followed by a positive integer, followed by a colon (:). For example:

```
r1:
r99:
```

record_type_value The value to be matched with the record type in order to do the warp. If DEFAULT, the warp is done if none of the defined record type values matches the record type. For example:

```
r4:      1        2 z + 1
r4:      2        2 z + 2
r4:      default  2 z = 3  //  if record type 4
                           //  is not 1 or 2, do
                           //  this warp
```

The record_type_value is treated as a character (byte) field for comparison to the record type. In the previous examples, the values character 1 ('F1'x) or 2 ('F2'x) are compared to the record type.

comments Comments can be as follows:

Entire line The line must start with *.
End of line Anything following // is commented

The following example shows an end of line comment:

```
fieldName   4 zoned + 1  // end of line comment on
                         // a field definition
```

Control File Example

The following is an example of the control file with the file warping definitions:

```
    01       hi_level_name1
1:      03          what_record1 1 char     // record type field used for warping
        03          fill        39 char     // user can leave out
                                            // fields as long
                                            // as they put in a field with
                                            // cumulative offset
        03          some_date    8 packed signed = 2001
     some_number                 2 z   + 1  // level numbers are
                                            // optional
        05      rec_warp1        2 packed signed
r1:      '01'x       2 p s = 99             // if '01'x records set rec_warp
r1:      '02'x       2 p s = 01             // if '02'x records set rec_warp
r1:      '03'x       2 p s + 1              // if '03'x records add 1 to rec_warp
r1:      default     2 p s = 98             // otherwise set to 98
        05      rec_warp2        4 z
r2:      CUST1       4 z + 1                // if CUST1 record add 1 to rec_warp2
r2:      CUST2       4 z + 2                // if CUST2 record add 2 to rec_warp2
r2:      ' C 3'      4 z + 3                // if ' C 3 ' record add 3 to
                                            //  rec_warp2
2:       what_record2 5 c                   // record type field for rec_warp2 field
           fill               100 c         // space to next warp
         some_name              10 c = 1234567890
```

Figure 71. Example Control File with File Warping Definitions

In the previous example:

- The some_date field is set to the packed signed number 2001 for all records.
- The some_number field is incremented with the zoned unsigned value 1 for all records.

- The what_record1 field defines a record type. The rec_warp1 field is warped as follows:
  - If '01'x, records (hex) are set to the packed, signed number 99.
  - If '02'x, records (hex) are set to the packed, signed number 01.
  - If '03'x, records (hex) add 1.
  - Any records that do not have one of the previous values will be set to 98.
- The what_record2 field is a record type used to warp rec_warp2 as follows:
  - CUST1 records (character) are incremented by 1.
  - CUST2 records are incremented by 2.
  - ' C 3 ' records (character) are incremented by 3.
- The some_name field is set to the character string '1234567890' for all records.

File Warp Return Codes

For file warping, there are two job steps:

1. Check the file warp control file for syntax (job step CVTCTL).
2. Perform the file warp (job step FWARP).

If the file warp JCL returns a nonzero return code, there was a failure in the warp process.

When the CVTCTL step fails, an error message is printed in the SYSTSPRT DD statement of the CVTCTL step. When the FWARP step fails, an error message is printed in the SYSPRINT DD statement of the FWARP step. For an explanation of file warping error messages, see Appendix A, “Problem Determination” on page 307.

Common CA, DA, and UTA Information

© Copyright IBM Corp. 1997, 1999

CA, DA, and UTA Control File

This chapter describes the function of the CA/DA/UTA control file (CACTL). The CACTL file contains information that describes the compile units to be analyzed, the file that is to be monitored, and the values of the variables to be recorded.

Note: CA, DA, and UTA share the CACTL file.

To edit the control file:

1. Select option `1` from the `ATC Primary Option Menu`. The `Coverage, Distillation and Unit Test Assistant` panel is displayed.

```
------------ Coverage, Distillation and Unit Test Assistant--------------------
Option ===>

1  Cnt1File      Work with the CA/DA/UTA Control File
2  Setup         Create JCL for Setup
3  StartMon      Create JCL to Start the Monitor
4  CA            Coverage Reports
5  DA            Distillation
6  UTA           Unit Test Report
7  Monitor       Control the CA/DA/UTA Monitor

Enter END to Terminate
```

Figure 72. Coverage, Distillation and Unit Test Assistant Panel

2. Select option `1` to display the `Work with the CA/DA/UTA Control File` panel, shown in Figure 73.

```
--------------------- Work with the CA/DA/UTA Control File --------------------
Option ===>

1  Edit          Edit CA/DA/UTA Control File
2  Reset         Reset CA/DA/UTA Control File from System Master

Enter END to Terminate

Use Program Name for File Name  YES (Yes|No)   Program Name  COB01M

CA/DA/UTA Control File:
  Control File Dsn. . . 'YOUNG.TEST.COB01M.CACTL'
  Listing Type. . . . . COBOL    (COBOL|PL/I|ASM)
```

Figure 73. Work with the CA/DA/UTA Control File Panel

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

© Copyright IBM Corp. 1997, 1999

The panel's options and fields are as follows:

Edit

Starts an edit session for the control file specified in the `Control File Dsn` field.

Reset

Replaces the information in the control file specified in the `Control File Dsn` field with information from the system master control file. The system master control file is a sample control file with comments describing all of the lines, fields, and options. (See Listing Type below.)

Use Program Name for File Name

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, enter `YES`.

When you press Enter, the file names on the panel are changed automatically. Using the program name is the normal CA, DA, and UTA procedure.

Program Name

Specifies the next-to-last qualifier to be used when the `Use Program Name for File Name` field is set to `Yes`.

Control File Dsn

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

Listing Type

Type of CACTL template that is retrieved from *hi_lev_qual*.V1R5M0.FORMS if you select Edit (*and* the data set or member in the `Control File Dsn` field does not exist) or Reset.

4. Press Enter.

   An ISPF edit session for the data set 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.

5. Edit the control file to include all programs and variables to be monitored in this session.
6. Press the End key (PF3) key 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, unit testing, or distillation.

The following describes the contents of the control file:

- The **INCLUDE** statement. This statement allows 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 of the following:
  - The data set name of the file to be included
  - The DDNAME of a previously allocated file that is to be included
- The **DEFAULTS** statement. This statement allows defaults to be set for certain keywords on subsequent COBOL, PL/I, or ASM statements.
- The compilation unit (**COBOL**, **PL/I**, or **ASM**) statement. This statement identifies the following:
  - The type of listing file (COBOL, PL/I, or ASM)
  - The name of the data set containing 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 containing 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

  Note: If breakpoints are placed in object modules, then a specific compilation unit should be specified only once in a control file. If the compilation unit is link-edited into more than one load module, it should be listed for just one of the load modules.
- The scope definition (**SCOPE**) statement.[36] This statement does either of the following:
  - Defines the scope for subsequent COBOL variable or file definitions and identifies:
    - The PROGRAM-ID of the external program containing the variable
    - The PROGRAM-ID of the internal (nested) program containing the variable (if the variable is defined within a nested program)
  - Defines the scope for subsequent PL/I file definitions and identifies the name of the procedure or begin-block containing the variable
- The variable definition (**VARIABLE**) statement. This statement identifies the *fully-qualified* name of the COBOL variable for which unit test coverage is requested or the *partially-qualified* name of a variable when unit test coverage is requested for all variables containing this qualifier.
- The coverage definition (**COVERAGE**) statement. This statement specifies attributes and coverage details for a particular variable.

[36] **SCOPE** statements are used only by Distillation Assistant and Unit Test Assistant. They are not used by Coverage Assistant.

- The warp definition (**WARP**) statement. This statement specifies attributes and warping details for a particular variable or buffer.
- The file definition (**FILE**) statement. This statement specifies that all reads of the specified COBOL or PL/I file are to be monitored for distillation. The VARIABLE and FILE statements should never be used in the same control file.

DBCS support for the control file statements is explained in Appendix C, “DBCS Support” on page 391.

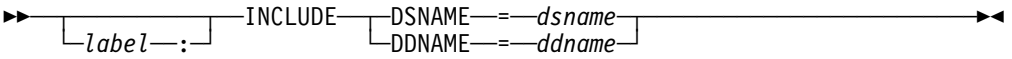
Control File Statement Syntax

This section describes the syntax of the control statements described previously. The syntax of all control statements 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 following the two slashes is a comment.
- Lines containing nothing but blanks are ignored.
- Keywords and operands may be coded in any combination of uppercase and lowercase characters.
- Operands may 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 may be continued, if desired, 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 specified on COBOL, PL/I, and ASM statements cannot be repeated on any of those statements. Likewise, labels on SCOPE statements cannot be repeated on other SCOPE statements and labels on VARIABLE statements cannot be repeated on other VARIABLE statements.
- Operands 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.

INCLUDE Statement

The INCLUDE statement can be used to include information from another CA/UTA/DA control file. When such a control file is included, all statements are processed as if they were in the original control file.



where:

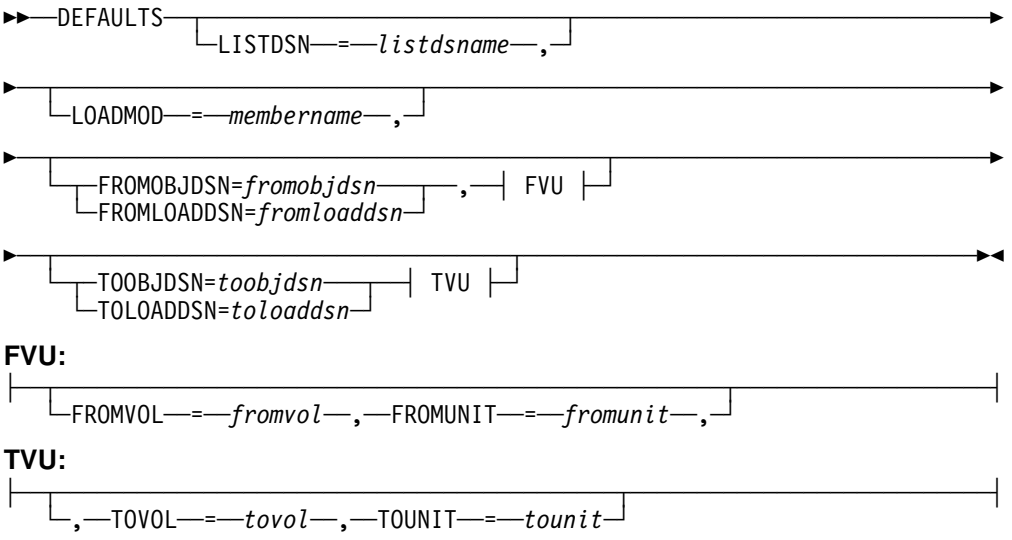
dsname specifies the data set name of the CA/DA/UTA control file to be included.

ddname specifies a ddname which has been previously allocated to a CA/DA/UTA control file.

DEFAULTS Statement

The DEFAULTS statement specifies defaults to be used for certain keywords on subsequent COBOL, PL/I, and ASM statements.

If a COBOL, a PL/I, or an ASM statement that does not specify a keyword is encountered, and the statement was preceded by a DEFAULTS statement that specified the keyword, the value specified on the DEFAULTS statement is used. If more than one DEFAULTS statement is found, the last DEFAULTS statement that specified the keyword in question is used.



where:

listdsname the name of the data set that contains the compiler listing for this compilation unit. If *listdsname* is specified on the DEFAULTS statement, an asterisk must be specified as the member name if the data set is partitioned or as one of the qualifiers if the data set is sequential. If LISTDSN= is not specified on a subsequent COBOL, PL/I, or ASM statement, the LISTMEMBER= operand must be specified. In this case the LISTMEMBER= operand will be used to replace the asterisk to create the name to be used for that statement.

membername the name of the load module containing this compilation unit.

fromobjdsn the data set name of the partitioned data set containing the object generated by the compiler for this compilation unit.

fromloaddsn the data set name of the partitioned data set containing the load module generated by the linker/binder.

fromvol the volume containing 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 the 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.

tovol the volume containing the *toobjdsn* or *toloaddsn* data set if it is not cataloged.

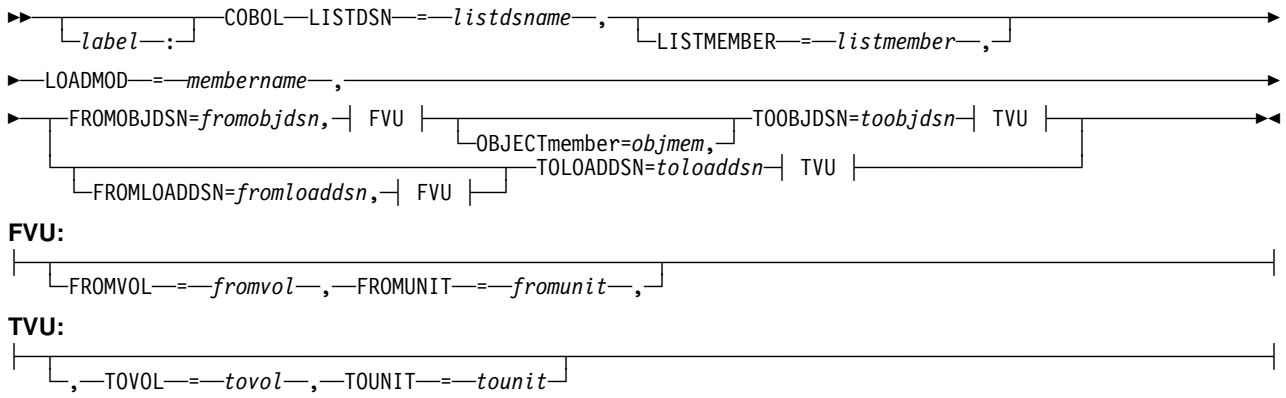
tounit the unit specification for the *toobjdsn* or *toloaddsn* data set if it is not cataloged.

COBOL Statement (COBOL Compilation Unit Definition)

The COBOL statement identifies a COBOL compilation unit.

Note: If breakpoints are placed in object modules, then a specific compilation unit should be specified only once in a control file. If the compilation unit is link-edited into more than one load module, it should be listed for just one of the load modules.

The syntax of the COBOL statement is:



where:

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) containing 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 containing 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,** *listdsname* **must be specified** and must contain a member name specification (which may be obtained from the LISTMEMBER= operand). In this case, the member name specified as a part of *listdsname* will be 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 containing 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. Coding FROMOBJDSN indicates that the load module *membername* is to be copied from *fromloaddsn* to *toloaddsn* before the breakpoints are applied via AMASPZAP to *membername* in *toloaddsn*.

The *fromloaddsn* load module may be coded as an * to prevent copying of the load modules before AMASPZAP is called (typically, this is only used if FROMLOADDSN has a default value).

AMASPZAP 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.

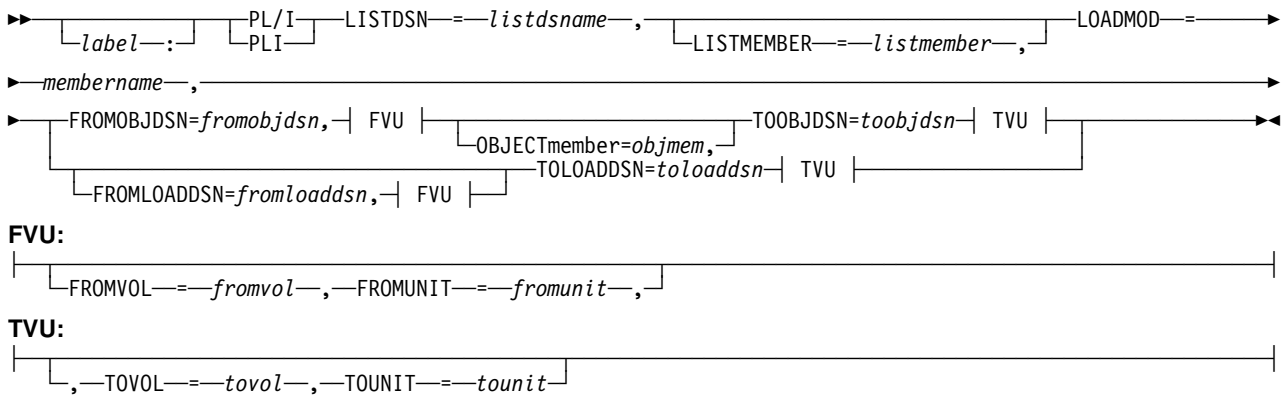
If the load module being instrumented has aliases, you should not use the FROMLOADDSN keyword, but should do the copy yourself using a method that preserves the aliases. Note that applying breakpoints to load modules is slower than applying them to object modules and requires access to AMASPZAP. However, it may be preferable if your normal build process does not keep a copy of the object modules for later processing or you do not want to maintain a separate set of link/bind JCL.

PL/I Statement (PL/I Compilation Unit Definition)

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

Note: If breakpoints are placed in object modules, then a specific compilation unit should be specified only once in a control file. If the compilation unit is link-edited into more than one load module, it should be listed for just one of the load modules.

The syntax of the PL/I statement is:



where:

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) containing 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 containing 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,** *listdsname* **must be specified** and must contain a member name specification (which may be obtained from the LISTMEMBER= operand). In this case, the member name specified as a part of *listdsname* will be 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 containing 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. Coding FROMOBJDSN indicates that the load module *membername* is to be copied from *fromloaddsn* to *toloaddsn* before the breakpoints are applied via AMASPZAP to *membername* in *toloaddsn*.

The *fromloaddsn* load module may be coded as an * to prevent copying of the load modules before AMASPZAP is called (typically, this is only used if FROMLOADDSN has a default value).

AMASPZAP 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.

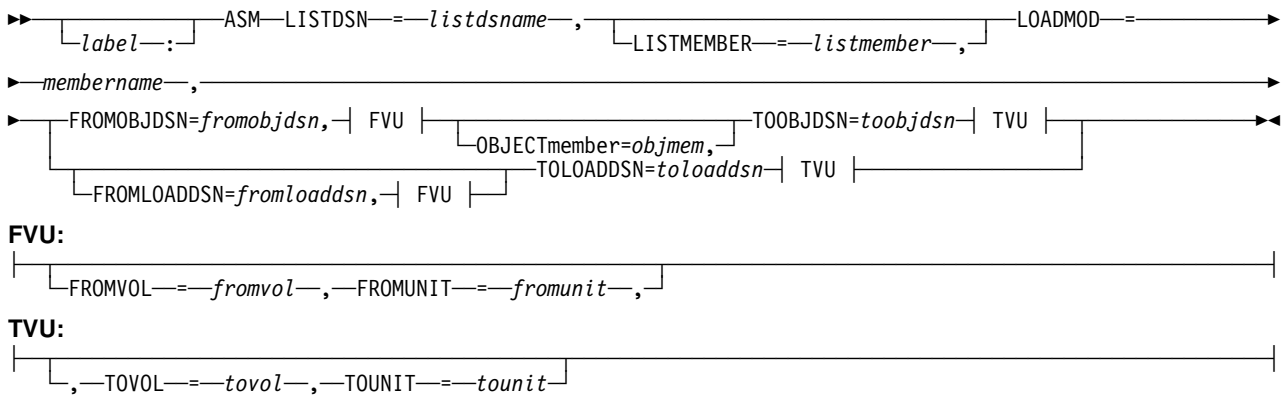
If the load module being instrumented has aliases, you should not use the FROMLOADDSN keyword, but should do the copy yourself using a method that preserves the aliases. Note that applying breakpoints to load modules is slower than applying them to object modules and requires access to AMASPZAP. However, it may be preferable if your normal build process does not keep a copy of the object modules for later processing or you do not want to maintain a separate set of link/bind JCL.

ASM Statement (Assembler Compilation Unit Definition)

The ASM statement identifies an assembler program compilation unit.

Note: If breakpoints are placed in object modules, then a specific compilation unit should be specified only once in a control file. If the compilation unit is link-edited into more than one load module, it should be listed for just one of the load modules.

The syntax of the ASM statement is:



where:

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

listdsname the name of the data set which contains the assembler 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) containing 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 containing 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,** *listdsname* **must be specified** and must contain a member name specification (which may be obtained from the LISTMEMBER= operand). In this case, the member name specified as a part of *listdsname* will be 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 containing 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. Coding FROMOBJDSN indicates that the load module *membername* is to be copied from *fromloaddsn* to *toloaddsn* before the breakpoints are applied via AMASPZAP to *membername* in *toloaddsn*.

The *fromloaddsn* load module may be coded as an * to prevent copying of the load modules before AMASPZAP is called (typically, this is only used if FROMLOADDSN has a default value).

AMASPZAP 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.

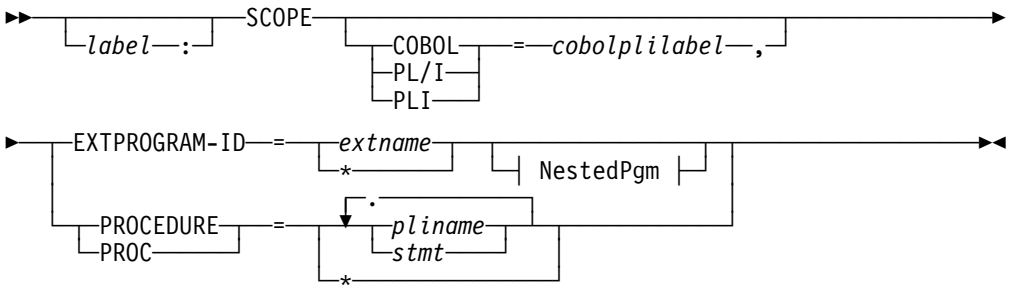
If the load module being instrumented has aliases, you should not use the FROMLOADDSN keyword, but should do the copy yourself using a method that preserves the aliases. Note that applying breakpoints to load modules is slower than applying them to object modules and requires access to AMASPZAP. However, it may be preferable if your normal build process does not keep a copy of the object modules for later processing or you do not want to maintain a separate set of link/bind JCL.

SCOPE Statement

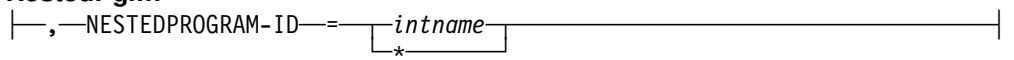
The SCOPE statement identifies a program scope in a COBOL or PL/I compilation unit that contains variables or files of interest.

Each SCOPE statement must be preceded by (and applies to) a COBOL or PL/I statement that specifies the listing containing the scope defined by the SCOPE statement.

The syntax of the SCOPE statement is:



NestedPgm:



where:

cobolplilabel a label on the COBOL or PL/I statement that defines the compilation unit containing this scope. If this operand is not specified, the default is the previous COBOL or PL/I statement.

extname the COBOL program-ID of the external COBOL program that contains items to be referenced.

***** indicates that the search for the referenced items is to be done through all external COBOL programs in the specified listing.

intname the COBOL program-ID of the internal (nested) COBOL program that contains items to be referenced. This operand should be specified only if the variables of interest are defined in a nested COBOL program. If this keyword is not specified, the variable is assumed to be defined in the external COBOL program.

***** indicates that the search for the referenced items is to be done through all internal program IDs in the specified external COBOL program-ID or through all procedures in the specified external PL/I procedure in the specified listing.

pliname the PL/I procedure or BEGIN-block that contains items to be referenced. For PL/I internal procedures, the form PROC1.PROC2 must be used, where PROC1 is the external procedure and PROC2 is an internal procedure contained in PROC1. Also, in the case of PL/I, named BEGIN blocks are considered to be equivalent to named procedures and are specified in exactly the same way. For unnamed BEGIN-blocks, the statement number (*stmt*) where the BEGIN-block is defined is used in place of the procedure name.

For example, **PROCEDURE=P0.P1.B1.2451.P3** specifies an external procedure named P0 that contains an internal procedure, or named BEGIN-block, called P1 that contains an internal procedure, or named BEGIN-block, called B1 that contains an unnamed BEGIN-block defined in statement 2451 that contains an internal procedure, or named BEGIN-block, called P3.

stmt see the previous description of *pliname*.

* indicates that the search for the referenced items is to be done through all PL/I procedures and BEGIN-blocks in the specified listing.

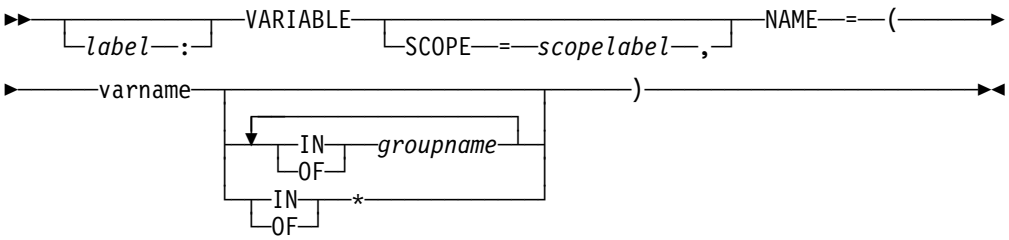
VARIABLE Statement

The VARIABLE statement identifies a COBOL variable for which coverage or a warp action is to be requested. The VARIABLE statement can also identify a PL/I file constant whose associated input file buffer is to be warped. VARIABLE statement can be used to specify either of the following:

1. The name of a fully-qualified variable.
2. The name of a single qualifier. In this case any fully-qualified names containing this qualifier are considered to be a match.

The SCOPE specified for a PL/I variable must be the procedure or BEGIN-block in which the variable is defined (either explicitly or implicitly), although it can be referenced in other, contained blocks. This means that the scope for implicitly declared variables (other than parameters) must be defined as the external procedure.

The syntax of the VARIABLE statement is:



where:

scopelabel the label specified on a SCOPE statement (which can refer to either a COBOL or a PL/I statement) that defines the scope in which this variable was defined. If this operand is not specified, the default is the previous SCOPE statement.

varname the name of the variable to be referenced.

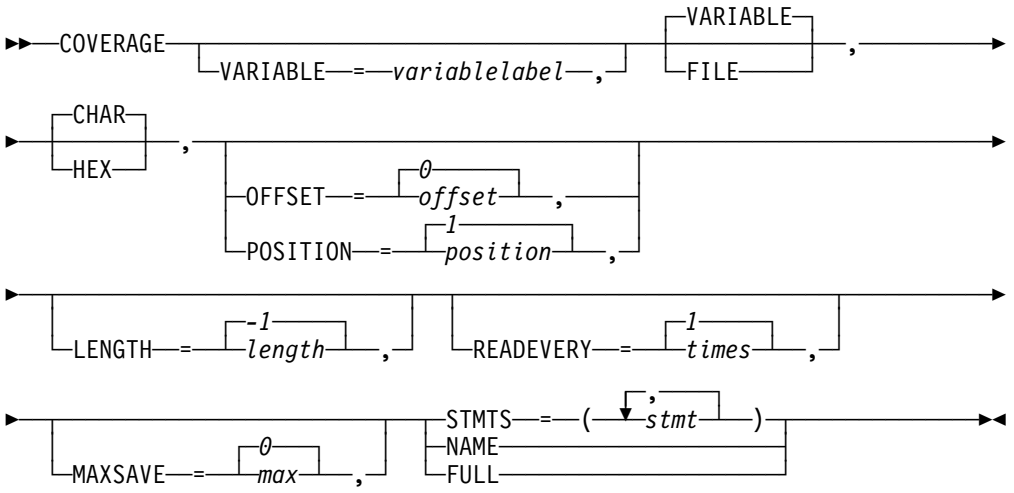
groupname the name of a group containing the referenced variable.

* indicates that a single qualifier is specified and that *any* fully-qualified variable that contains this qualifier is considered to be a match.

COVERAGE Statement

The COVERAGE statement specifies the coverage desired for a specific COBOL variable. COVERAGE with VARIABLE (coded or implied) causes the logging of the variable value at the beginning of the statement (for additional details, see "Where a Variable Is Read" on page 183). COVERAGE with FILE causes logging of the variable after a READ but before any other type of COBOL statement.

The syntax of the COVERAGE statement is:



where:

variablelabel
: the label on the VARIABLE statement to which this coverage applies. If this operand is not specified, the default is the previous VARIABLE statement.

VARIABLE indicates an internal (program) variable.

FILE indicates a File read variable *to be processed for distillation*.

CHAR specifies that variable data will be displayed as characters.

HEX specifies that variable data will be displayed as hexadecimal numbers, two hexadecimal digits per byte.

offset specifies the offset from the start of the data item where the read of the value is to begin. (An offset of 0 indicates the beginning of the item.)

position specifies the position of the first byte of the data item where the read of the value is to begin. (A position of 1 indicates the beginning of the item.)

length specifies the number of bytes of the data item to be read. A value of -1 indicates that the length is to be taken from the listing. The maximum length supported is 126 bytes.

times specifies the number of times to skip reading this data item. A value of 1 indicates that the value of the item is to be saved each time the item is referenced.

max specifies the maximum number of values that are to be saved for the item. A value of 0 indicates that all values are to be saved.

STMTS specifies that the variable is to be monitored only at specific statements.

stmt specifies the specific statements at which the variable is to be monitored.

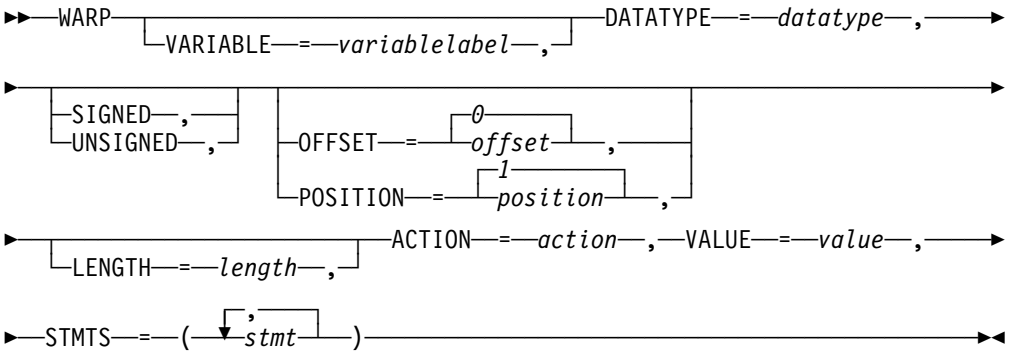
NAME specifies that the contents of the variable are to be read at all statements that reference it by the specified, fully-qualified name.

FULL specifies that the contents of the variable are to be read at all statements that reference it by the specified, fully-qualified name or by the name of any containing or contained group or element.

WARP Statement

The WARP statement specifies that a COBOL variable or PL/I file buffer is to be modified at a particular statement or statements (taken from the listing). This modification takes place *before* the statement has executed for COBOL, and *after* the statement has executed for PL/I.

The syntax of the WARP statement is:



where:

variablelabel

the label on the VARIABLE statement to which this warp action applies. If this operand is not specified, the default is the previous VARIABLE statement. For COBOL, this label can refer to any variable that represents a number. For PL/I, this label must refer to a file constant, and warping can occur only at file reads.

datatype specifies the type of data that is to be warped. Valid types are:

ZONED Indicates that the variable or file segment contains zoned decimal data (for example, 99='F9F9'x).

PACKED Indicates that the variable or file segment contains packed decimal data (for example, 99='099F'x).

BINARY Indicates that the variable or file segment contains binary data (for example, 99='63'x).

SIGNED specifies that the data is signed. This is the default for PL/I PACKED and PL/I BINARY datatypes.

UNSIGNED specifies that the data is unsigned. This is the default for all COBOL datatypes and also PL/I ZONED data types.

offset specifies the offset of the first byte of the data item where the read of the value is to begin. (An offset of 0 indicates the beginning of the item.)

position specifies the position of the first byte of the data item where the read of the value is to begin. (A position of 1 indicates the beginning of the item.)

length specifies the length of the data item to be warped.

This parameter indicates the number of:

- Digits (1-15) for ZONED and PACKED data.
- Bytes (2 or 4) for BINARY data.

The default values are as follows:

- For COBOL, the default of -1 indicates that the length is to be taken from the listing.
- For PL/I, the defaults are 2, if the data type is BINARY, and 1, if the datatype is ZONED or PACKED.

action specifies the warp action to be taken on the data. Valid actions are:

INCREMENT	Add the VALUE parameter to the data.
DECREMENT	Subtract the VALUE parameter from the data.
SET	Set the data equal to the VALUE parameter.

value indicates the numeric value to be used by the specified ACTION. The value must be a nonnegative integer.

stmt specifies the specific statements at which the variable or file segment is to be warped.

COBOL and PL/I Syntax Conversion Tables: The following tables may be helpful in converting COBOL and PL/I syntax to control card syntax for WARP statements:

Table 2. COBOL Syntax Conversion Table

COBOL Data Division Entry	Datatype	Sign	Length
Pic 9(P) Usage Display	Zoned	Unsigned	P
Pic S9(P) Usage Display	Zoned	Signed	P
Pic 9(P) Usage Comp-3	Packed	Unsigned	P
Pic S9(P) Usage Comp-3	Packed	Signed	P
Pic 9(1...4) Usage Comp	Binary	Unsigned	2
Pic 9(5...9) Usage Comp	Binary	Unsigned	4
Pic S9(1...4) Usage Comp	Binary	Signed	2
Pic S9(5...9) Usage Comp	Binary	Signed	4
Note: P=any integer 1-15 inclusive			

Table 3. PL/I Syntax Conversion Table

PL/I Declare	Datatype	Sign	Length
Character (P)[1]	Zoned	Unsigned	P
Picture '(P)9'[2]	Zoned	Unsigned	P
Picture '(P-1)9T'[2]	Zoned	Signed	P
Fixed Decimal (P,0)	Packed	Signed	P
Fixed Binary (15,0)	Binary	Signed/Unsigned	2
Fixed Binary (31,0)	Binary	Signed/Unsigned	4

Notes:

P=any integer 1-15 inclusive

For Fixed Decimal (P,Q) and Fixed Binary (P,Q), 0 is the only valid value for Q.

[1] Data must be right-justified within the field and padded on the left with zeros.

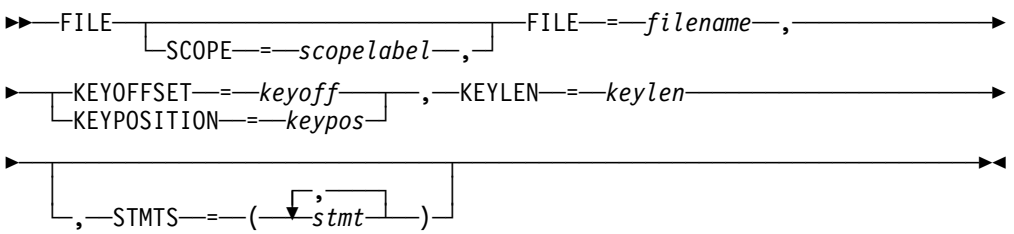
[2] Only characters *9* and *T* are supported (if used, *T* must be the rightmost character).

FILE Read Statement

The FILE statement identifies a COBOL or PL/I file to be processed for distillation. The file buffer is processed after the execution of the READ statement.

Note: Never use the FILE statement with VARIABLE statements in the same control file. If you use these statements together, the variable read data will be mixed with the key list, preventing distillation.

The syntax of the File statement is:



where:

scopelabel the label specified on the SCOPE statement that defines the scope in which this file was defined. If this operand is not specified, the default is the previous SCOPE statement.

filename the COBOL filename as specified on the FD statement or the name of a PL/I file *constant* with the RECORD attribute. PL/I file *variables* cannot be specified.

keyoff specifies the offset from the start of the record to the start of the logical key. (An offset of 0 indicates the beginning of the record.)

keypos specifies the position of the start of the logical key within the record. (A position of 1 indicates the beginning of the record.)

keylen the length of the logical key.

stmt a list of COBOL or PL/I statement numbers at which reads of the file are to be monitored.

For COBOL, if no *stmt* values are specified, all reads of the specified file are monitored.

For PL/I, *stmt* values are required. Only reads of the specified file at the specified statements are monitored.

Control File Examples

The following examples show how these statements might be used.

Coverage Assistant Control File Examples

These examples show two control files that might be used for Coverage Assistant only.

Coverage Assistant Example 1: This example shows a typical control file where coverage is requested for a single compilation unit.

```
 Cobol ListDsn=ATC.V1R5M0.Sample.Cobolst(Cob01AM),
       LoadMod=Cob01M,
       FromObjDsn=ATC.V1R5M0.Sample.Obj,
       ToObjDsn=ATC.V1R5M0.Sample.ZapObj
```

Coverage Assistant Example 2: This example shows a typical control file where coverage is requested for multiple compilation units.

```
         Defaults ListDsn=ATC.V1R5M0.Sample.Cobolst(*),
                  LoadMod=Cob01M,
                  FromObjDsn=ATC.V1R5M0.Sample.Obj,
                  ToObjDsn=ATC.V1R5M0.Sample.ZapObj
*
         Cobol    ListMember=Cob01AM
         Cobol    ListMember=Cob01CM
         Cobol    ListMember=Cob01DM
```

Distillation Assistant Control File Example

This example shows a control file that might be used for Distillation Assistant. In this example, all reads of the specified file are monitored.

```
         Cobol    ListDsn=ATC.V1R5M0.Sample.Cobolst(Cob092),
                  LoadMod=Cob092,
                  FromObjDsn=ATC.V1R5M0.Sample.Obj,
                  ToObjDsn=ATC.V1R5M0.Sample.ZapObj
         Scope    ExtProgram-Id=Cob092
         File     File=VSAMIN,KeyPosition=1,KeyLen=11
```

Unit Test Assistant Control File Example

This example shows a control file that might be used for Unit Test Assistant (including data warping).

```
Defaults ListDsn=ATC.V1R5M0.SAMPLE.COBOLST(*),
         LoadMod=COB02M,
         FromObjDsn=ATC.V1R5M0.SAMPLE.OBJ,
         ToObjDsn=ATC.V1R5M0.SAMPLE.ZAPOBJ

COBOL ListMember=COB02M

Scope ExtProgram-Id=COB02M

Variable Name=JULIAN-DATE
    Warp Action=Set, Value=0099365,
         Datatype=Zoned,Unsigned,Stmts=(83)

    Coverage Stmts=(87) // read JULIAN-DATE after it is warped

Variable Name=CURR-DATE
    Warp Action=Set, Value=00991231,
         Datatype=Zoned,Unsigned,Stmts=(94)

    Coverage Stmts=(97) // read CURR-DATE after it is warped

Variable Name=YEAR4
    Warp Action=Increment,Value=3,
         Datatype=Zoned,Unsigned,Stmts=(103)

    Coverage Length=4,
         Stmts=(106) // read YEAR4 after it is warped

Variable Name=YEAR IN YEAR-BY-FIELD IN DATE-BY-FIELD
    Warp Action=Decrement, Value=1,
         Datatype=Zoned,Unsigned,Stmts=(103)

Variable Name=YEAR2
    Coverage Length=2,
         Stmts=(114) // read YEAR2 after it gets the 2 digit

Variable Name=(BEGIN-DATE In LOAN)
    Coverage Length=8,
         NAME               // read BEGIN-DATE of LOAN

    Coverage Length=8,
         FULL               // read BEGIN-DATE of LOAN structure

Variable Name=INC-DATE
    Coverage Length=7,
          Stmts=(143) // read initialization of INC-DATE

Variable Name=J-DAY IN J-DATE
    Coverage Length=3,ReadEvery=100,
         MaxSave=5,Stmts=(153) // read J-DAY in loop
```

CA, DA, and UTA Setup

This chapter describes the CA, DA, and UTA Setup steps. You create the JCL to start the CA, DA, and UTA Setup program from the `Create JCL for Setup` panel, shown in Figure 76 on page 239.

Setup

The Setup process uses assembler statements from the listings produced by the compiler to determine where to place breakpoints. For DA and UTA, Setup also uses data map and data cross-reference (XREF) to extract variable storage and access information.

You must supply the following items for Setup:

1. Compiler listings with assembler statements included (for DA and UTA, also include data map and XREF)
2. The original object modules created during the source compile step
3. An object library (allocated like the original) to receive the new object modules, (after they have been modified with breakpoints)
4. Information on what variables (UTA only) or files (DA only) to monitor.

Modify this information as necessary in the control file (CACTL), as described at "Editing the Coverage Assistant Control File" on page 81, "Editing the Distillation Assistant Control File" on page 145, "Editing the Unit Test Assistant Control File" on page 187, and "CA, DA, and UTA Control File" on page 209.

Note: When using CA Targeted Summary, you must use the instructions, which follow, under **DA and UTA Setup**.

Compiler Options

CA Setup

COBOL For COBOL for OS/390 & VM and COBOL for MVS & VM, specify the following compiler options:[37]

- SOURCE
- LIST
- OBJECT
- NOOPTIMIZE
- NONUMBER
- NOTEST or TEST(NONE)
- LIB[38]

[37] Specifying the *CBL (*CONTROL) NOSOURCE Compiler-Directing Statement to suppress printing of COBOL executable statements prevents these statements from being included in the annotated listing report. Specifying the *CBL (*CONTROL) NOLIST Compiler-Directing Statement to suppress printing of the assembler code prevents setup from inserting any breakpoints into the suppressed assembler code.

[38] The LIB compiler option is only required by ATC if you have multiple source programs separated by CBL (Process) Compiler-Directing Statements.

© Copyright IBM Corp. 1997, 1999

For VS COBOL II, specify the following compiler options:[37]

- SOURCE
- LIST
- OBJECT
- NOOPTIMIZE
- NONUMBER
- NOTEST
- LIB[38]

For OS/VS COBOL, specify the following compiler options:

- SOURCE
- PMAP
- OBJECT
- NOLST
- NOOPTIMIZE
- NONUM
- NOBATCH
- NOTEST
- NOFLOW
- NOCOUNT
- NOSYMDMP

For COBOL, each listing may 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.

All COBOL listings must have the following DCB attributes:

- RECFM=FBA
- LRECL=133 for COBOL for OS/390 & VM, COBOL for MVS & VM, and VS COBOL II
- LRECL=121 for OS/VS COBOL

PL/I For PL/I for MVS & VM, specify the following compiler options:[40]

- SOURCE
- LIST
- OBJECT
- NOOPTIMIZE
- NOTEST or TEST(NONE)

For PL/I 2.3.0, specify the following compiler options:[39] [40]

- SOURCE
- LIST
- OBJECT
- NOOPTIMIZE

---

[39] ATC 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.

[40] Specifying the %NOPRINT statement to suppress printing of PL/I executable statements prevents these statements from being included in the annotated listing report produced by CA.
Multiple PL/I external source programs separated by *PROCESS statements are not supported.
The use of the REORDER Procedure or BEGIN block option is not supported.

- NOTEST
- NOCOUNT
- NOFLOW

For PL/I 1.5.1, specify the following compiler options:[39] [40]

- SOURCE
- LIST
- OBJECT
- NOOPTIMIZE
- NOCOUNT
- NOFLOW

For PL/I, each listing may contain (1) one external procedure, and (2) 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 CA-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:

- RECFM=VBA
- LRECL=125

ASM For the High Level Assembler, specify the following assembler options:

- NOBATCH
- ESD
- NOGOFF (Release 3 only)
- LIST
- OBJECT or DECK
- NOXOBJECT
- PCONTROL(ON,GEN)

For Assembler H, specify the following assembler options:[41]

- NOBATCH
- ESD
- LIST
- OBJECT or DECK

All ASM listings must have the following DCB attributes:

- RECFM=FBM
- LRECL=133 for the High Level Assembler
- LRECL=121 for Assembler H

[41] Specifying the PRINT assembler directive to suppress printing of executable instructions or data prevents Setup from inserting any breakpoints into the suppressed assembler code.

Each listing can contain one assembly. If you only want a summary report of code coverage, you can discard the compiler listings after the Setup step.

ATC does not support breakpointing of:

- Self-modifying code
- Code that uses the ESA/390 branch relative instructions
- An assembly that contains more than one executable CSECT

DA and UTA Setup and CA Targeted Summary Setup

COBOL For COBOL for OS/390 & VM and COBOL for MVS & VM, specify the following compiler options:[42]

- SOURCE
- LIST
- OBJECT
- XREF
- MAP
- NOOPTIMIZE
- NONUMBER
- NOTEST or TEST(NONE)
- LIB[38]

For VS COBOL II, use the following compiler options:[42]

- SOURCE
- LIST
- OBJECT
- XREF
- MAP
- NOOPTIMIZE
- NONUMBER
- NOTEST
- LIB[38]

For OS/VS COBOL, use the following compiler options:[43]

- SOURCE
- DMAP
- PMAP
- OBJECT
- XREF
- NOLST
- NOCLIST
- NOOPTIMIZE
- NONUM
- NOBATCH
- NOSXREF

---

[42] Specifying the *CBL (*CONTROL) NOLIST Compiler-Directing Statement to suppress printing of the assembler code prevents CA from inserting any breakpoints into the suppressed assembler code.

[43] DA and UTA do not support CICS routines compiled with the OS/VS COBOL compiler.

- NOTEST
- NOFLOW
- NOCOUNT
- NOSYMDMP

For COBOL, each listing may 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.

All COBOL listings must have the following DCB attributes:

- RECFM=FBA
- LRECL=133 for COBOL for OS/390 & VM, COBOL for MVS & VM, and VS COBOL II
- LRECL=121 for OS/VS COBOL

PL/I

For PL/I for MVS & VM, specify the following compiler options:[40]

- ATTRIBUTES(FULL)
- NOGRAPHIC
- LIST
- MAP
- NEST
- OBJECT
- OFFSET
- NOOPTIMIZE
- OPTIONS
- SOURCE
- XREF(FULL)
- NOTEST or TEST(NONE)

For PL/I 2.3.0, specify the following compiler options:[39] [40]

- ATTRIBUTES(FULL)
- NOGRAPHIC
- LIST
- MAP
- NEST
- OBJECT
- OFFSET
- NOOPTIMIZE
- OPTIONS
- SOURCE
- XREF(FULL)
- NOTEST
- NOCOUNT
- NOFLOW

For PL/I 1.5.1, specify the following compiler options:[40]

- ATTRIBUTES(FULL)
- NOGRAPHIC
- LIST
- MAP
- NEST
- OBJECT
- OFFSET

- NOOPTIMIZE
- OPTIONS
- SOURCE
- XREF(FULL)
- NOCOUNT
- NOFLOW

For PL/I, each listing may contain (1) one external procedure, and (2) 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 CA-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:

- RECFM=VBA
- LRECL=125

Setup creates a breakpoint table (BRKTAB) and uses it along with the object module created at compile time to create a new object module containing the breakpoint data. Once 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.

For DA and UTA, Setup also creates a debug table (DBGTAB) of data on variables to read. DA and UTA use the DBGTAB table during the Execution and Report steps.

Also for DA and UTA, setup creates a VARCTL file that contains variable location information. DA and UTA use the VARCTL file during Execution.

The steps involved in the Setup procedure for CA are shown in Figure 74 on page 237. The DA and UTA Setup procedure is shown in Figure 75 on page 237. The names outside the boxes in the figures (for example, BRKTAB) correspond to the DDNAMEs in the created JCL.

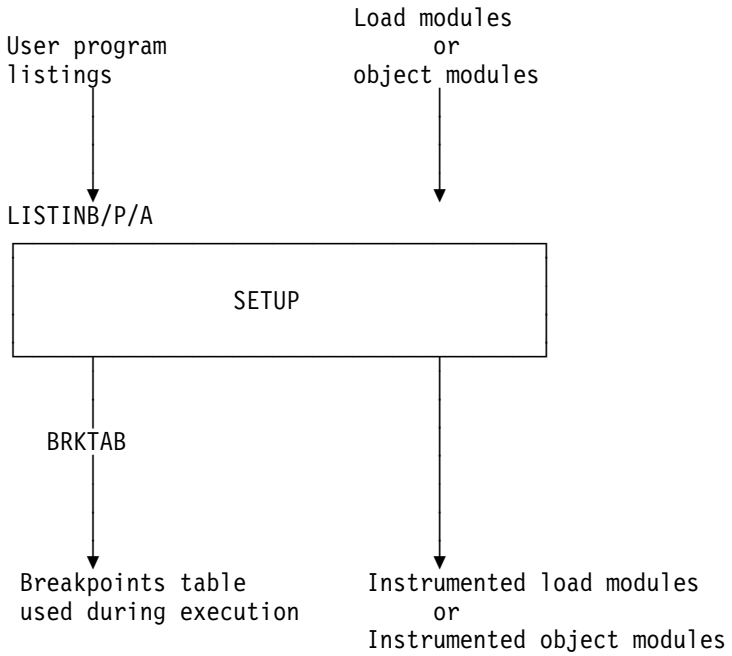


Figure 74. CA Setup—Flow Diagram

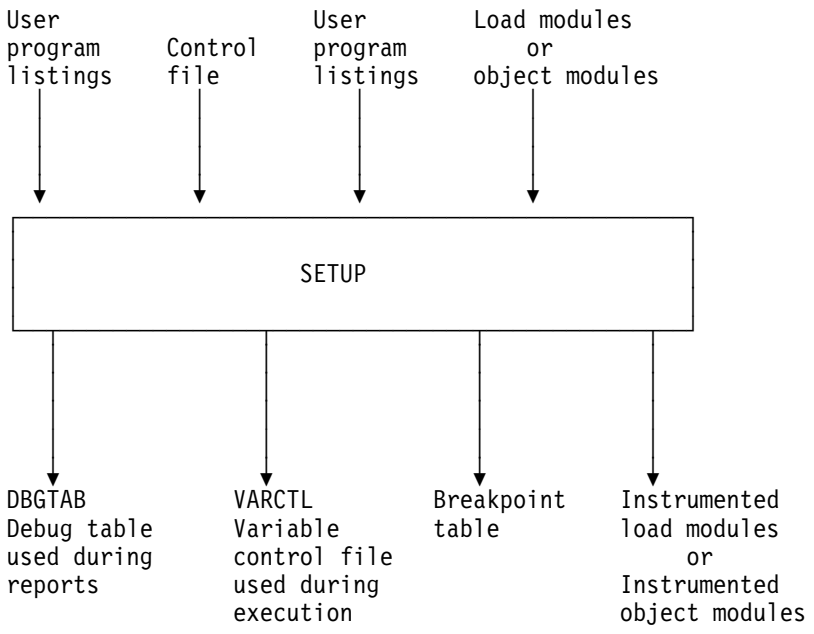


Figure 75. DA and UTA Setup—Flow Diagram

Instrumentation of Load Modules instead of Object 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 programs. For locations where instrumenting object modules is difficult because the link step is built into the standard location-wide build procedures, instrumentation of load modules can be used.

The samples shipped with ATC show how to instrument object modules.

To instrument object modules for the COB01M test case, the following control file would be used:

```
 Defaults ListDsn=ATC.V1R5M0.SAMPLE.COBOLST(*),
          LoadMod=COBO1M,
          FromObjDsn=ATC.V1R5M0.SAMPLE.OBJ,
          ToObjDsn=yourid.SAMPLE.ZAPOBJ

COB01AM:      COBOL ListMember=COB01AM
COB01CM:      COBOL ListMember=COB01CM
COB01DM:      COBOL ListMember=COB01DM
```

To instrument load modules, the control file is changed. All other steps remain the same, except 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 COB01M test case, the following COBOL control file would be used:

```
Defaults ListDsn=ATC.V1R5M0.SAMPLE.COBOLST(*),
         LoadMod=COBO1M,
         FromLoadDsn=ATC.V1R5M0.SAMPLE.LOADLIB,
         ToLoadDsn=yourid.SAMPLE.RUNLIB

COB01AM:      COBOL ListMember=COB01AM
COB01CM:      COBOL ListMember=COB01CM
COB01DM:      COBOL ListMember=COB01DM
```

Note that the FromObjDsn and ToObjDsn keywords that identify object module libraries are not used. Instead you use the FromLoadDsn and ToLoadDsn keywords. If both FromLoadDsn and ToLoadDsn are supplied, the load module is copied before being instrumented. If only the ToLoadDsn keyword is supplied, 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.

Note that the load module copy step (using IEBCOPY) will not work for load modules that have aliases.

The instrumentation of the load module is done using the IBM utility AMASPZAP (sometimes referred to as the Superzap utility). Your location may restrict the use of this utility. Contact your system support personnel if you cannot access it.

Instrumenting load modules using the AMASPZAP utility takes significantly longer than instrumentation of object modules for large programs.

Creating the Setup JCL Using the Panels

To create the setup JCL:

1. Select option 1 from the ATC Primary Option Menu.

   The Coverage, Distillation, and Unit Test Assistant panel is displayed.

2. Select option 2.

   The Create JCL for Setup panel, shown in Figure 76, is displayed. Each field on the panel is described following the figure.

3. Enter any information you want to change, select option 1, and 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

Process Options:
  Enable CA  YES (Yes|No)  Enable DA  NO (Yes|No)  Enable UTA  NO (Yes|No)

Use Program Name for File Name  YES (Yes|No) Program Name  COB01M

CA/DA/UTA Control File:
  Control File Dsn. . . 'YOUNG.TEST.COB01M.CACTL'

JCL Library and Member:
  JCL Dsn . . . . . 'YOUNG.TEST.JCL.CNTL(Sxxxxxxx)'

Output Breakpoint Table:
  Breakpoint Table Dsn. 'YOUNG.TEST.COB01M.BRKTAB'

DA/UTA Output Debug Table and Variable Control Files:
  Debug Table Dsn . . . 'YOUNG.TEST.COB01M.DBGTAB'
  Variable Control Dsn. 'YOUNG.TEST.COB01M.VARCTL'
```

Figure 76. Create JCL for Setup Panel

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

Generate
: Generates JCL using the parameters you have specified on the panel.

Edit
: Displays an ISPF edit session for you to make changes to existing JCL.

Submit
: Submits for execution the JCL specified in the JCL Dsn field on this panel.

Enable CA

Specifies whether Coverage Assistant's setup JCL is to be generated.

Enable DA

Specifies whether Distillation Assistant's setup JCL is to be generated. If you set this option to `YES`, set `Enable UTA` to `NO`.

Enable UTA

Specifies whether Unit Test Assistant's setup JCL is to be generated. If you set this option to `YES`, set `Enable DA` to `NO`.

Use Program Name for File Name

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, enter `YES`.

When you press Enter, the file names on the panel are changed automatically. Using the program name is the normal ATC procedure.

Program Name

Name to use for ATC files if you enter `YES` in the `Use Program Name for File Name` field. Note that this 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.TEST.COB01M.BRKTAB'

- Partitioned data sets:

  `'proj_qual.file_type(program_name)'`

  For example: 'YOUNG.TEST.BRKTAB(COB01M)'

Control File Dsn

Specifies the control file (CACTL) data set containing the names of the listing files being annotated and the variables being read. If you want to edit the control file, you can do so from the `Work with the CA/DA/UTA Control File` panel.

JCL Dsn

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

Note: 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 S*xxxxxxx*, where *xxxxxxx* is the first seven characters of the program name.

Breakpoint Table Dsn

Name of the BRKTAB data set created during setup and used by the monitor program.

Debug Table Dsn
Name of the data set containing the variables that UTA is to monitor and the files that DA is to monitor.

Variable Control Dsn
Name of the data set containing the variable information.

When to Create or Submit Setup JCL

Run the setup JCL if:

- A user program (and consequently, the listing) changes.
- You change the variables to read.

You only need to recreate the setup JCL if the test environment changes.

For example, if you:

- Add or delete listings in the CACTL, recreate the setup JCL from the panel, and run the new setup JCL.
- Change a listing, then submit the old setup JCL without changes.
- Change a variable in the CACTL, submit the old setup JCL without changes.

Setup JCL for the Compile Job Stream

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

You can create this JCL by modifying the sample JCL in the ATC JCL library or by generating the setup JCL from the `Create JCL for Setup` panel.

Parameters for SETUP and ZAPTXT Programs

This section describes the input parameters you can provide to the SETUP and ZAPTXT programs.

SETUP

The SETUP program input parameters are created automatically from the defaults file. The parameters are as follows:

LoadMod Load module name. This is the name of the load module that contains the PA or CSECT name.

LPDFRP List_Type/PA_Type/Debug_mode/Frequency_mode/Read_Variable_data/Performance_mode.
This is a six-part parameter defined as follows:

List_Type The type of listing. This is controlled by the CACTL variable in the site defaults file. Enter any of the following:
`B` For COBOL
`P` For PL/I
`A` For Assembler

PA_Type Program Area Type. Enter the following:
`0` For DASD resident program

Debug_mode

Use Debug Mode? Enter either of the following:

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

Note: Use debug mode only at the direction of ATC support personnel or as directed by this manual.

N Do not use debug mode: only execute breakpoints once.

Note: For CA, DA, and UTA, set to N. See also "Displaying Execution Counts in an Annotated Listing" on page 102 for more information about the use of this flag.

Frequency_count_mode

Use Frequency Count Mode? Enter either of the following:

Y Save frequency and branch to information (for branches) in the coverage file (BRKOUT). This adds 16 bytes per breakpoint to the BRKOUT size. This can be set to N, unless you want the Events count information in the annotated listing.

N Do not save the Events count and branch to information in the BRKOUT file.

Note: For CA, set to N unless analyzing assembler code with relative branches (see "ASTERISKS BY STATEMENT TOTALS" on page 89), displaying execution counts (see "Displaying Execution Counts in an Annotated Listing" on page 102). For DA, set to N unless recording first execution of DA keys (see"Recording Which Keys Execute a Statement" on page 142). For UTA, set to N.

Read_Variable_data

Read Variable Data? Enter either of the following:

Y DA and UTA can read file keys and variable data.

N Do not set read points for DA and UTA.

Note: For CA, set to N. For UTA and DA, set to Y.

Performance_mode

Use Performance Mode? Enter either of the following:

Y Conditional branch coverage is disabled.

N Conditional branch coverage is enabled.

For more information, see "Using the Performance Mode to Reduce Monitor Overhead" on page 253.

Listing file name
Listing file name used to determine breakpoint placement.

2-byte SVC #
SVC number (in hexadecimal notation) for breakpointing 2-byte instructions.

4-byte SVC #
SVC number (in hexadecimal notation) for breakpointing 4- and 6-byte instructions.

ZAPTXT

The ZAPTXT program inserts breakpoints into object modules. It has one input parameter, which is built automatically by the ISPF dialog during the creation of the setup job.

```
//OBJZAP EXEC PGM=ZAPTXT,PARM='x'
```

where *x* is a number that indicates which BRKTAB in the BRKTAB data set to use.

Monitor Execution

This chapter describes the procedures for running a monitor session for MVS. The monitor handles the user supervisor call instructions (SVCs) that are used as breakpoints (BPs).

You install the monitor by running the monitor installation program during ATC installation. For details about installing the monitor, see “System Installation” on page 13.

You create the JCL to start a monitor session from the `Create JCL to Start the Monitor` panel (shown in Figure 80 on page 247). Multiple user sessions can be active at the same time.

The monitor program examines the breakpoint table information (and for UTA and DA, the variable control information) produced by the Setup step and accumulates data as the application program test case executes. When you execute the CASTOP command, the monitor writes the results to disk. The CA execution flow is shown in Figure 77, the DA execution flow is shown in Figure 78 on page 246, and the UTA execution flow is shown in Figure 79 on page 246.

When the monitor session is started, the tables needed for handling the session are created in ESQA. For the amount of storage used for a user session, see Appendix B, “ATC Requirements and Resources” on page 383. For more details on running multiple sessions, see page 249.

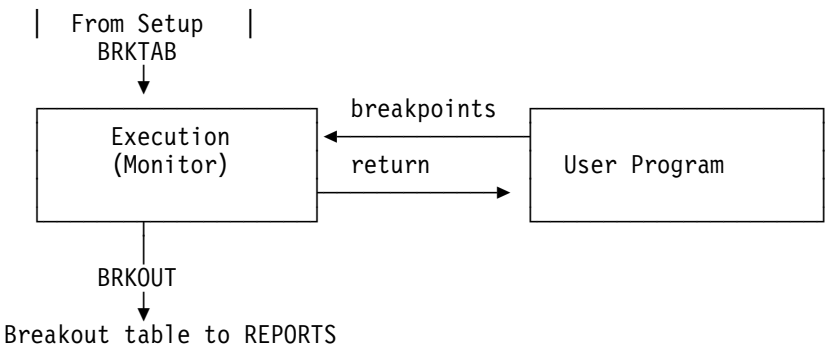


Figure 77. CA Test Case Execution—Flow Diagram

© Copyright IBM Corp. 1997, 1999

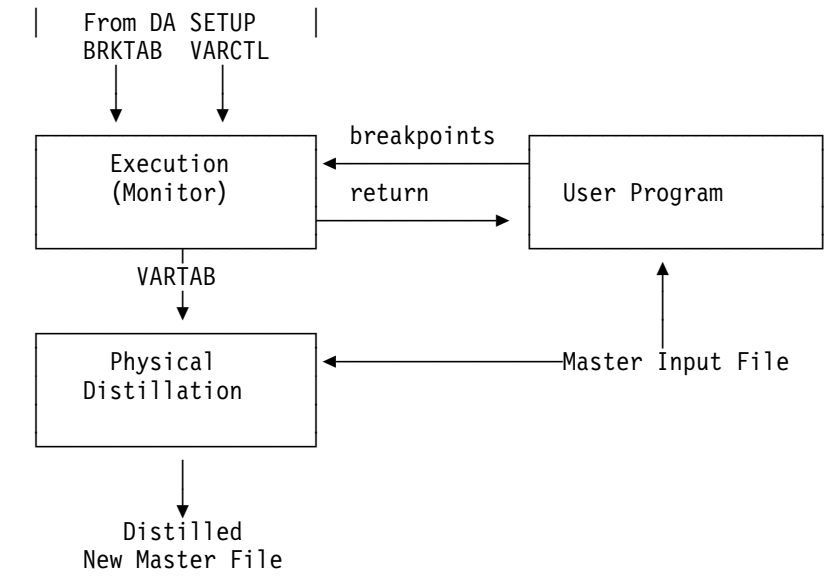


Figure 78. DA Test Case Execution—Flow Diagram

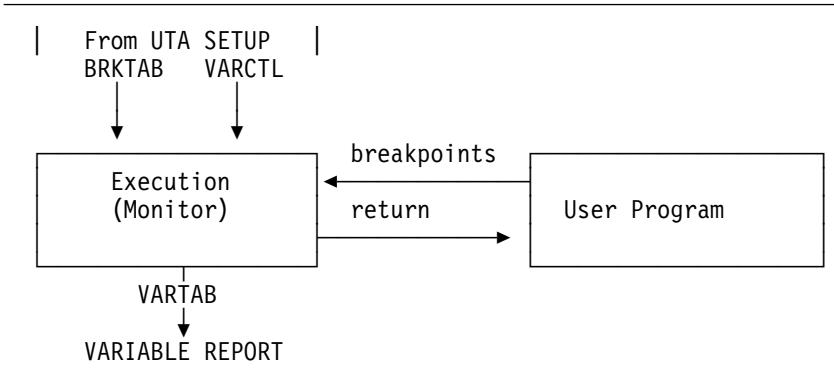


Figure 79. UTA Test Case Execution—Flow Diagram

Creating the Monitor JCL Using the Panels

To create the monitor JCL:

1. Select option `1` from the `ATC Primary Option Menu`.

   The `Coverage, Distillation, and Unit Test Assistant` panel is displayed.

2. Select option `3`.

   The `Create JCL to Start the Monitor` panel, shown in Figure 80 on page 247 is displayed.

3. Enter any information you want to change, select option `1`, and 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

Process Options:
  Enable CA YES (Yes|No)  Enable DA NO  (Yes|No)  Enable UTA NO  (Yes|No)

Use Program Name for File Name  YES (Yes|No) Program Name  COB01M

Session ID  . . . . . . YOUNG

Input File:
  Breakpoint Table Dsn. 'YOUNG.TEST.COB01M.BRKTAB'

DA/UTA Input File:
  Variable Control Dsn. 'YOUNG.TEST.COB01M.VARCTL'

JCL Library and Member:
  JCL Dsn . . . . . . . 'YOUNG.TEST.JCL.CNTL(Xxxxxxxx)'

Output File:
  Breakout Dsn. . . . . 'YOUNG.TEST.COB01M.BRKOUT'

DA/UTA Output File:
  Variable Table Dsn. . 'YOUNG.TEST.COB01M.VARTAB'
```

Figure 80. Create JCL to Start the Monitor Panel

The panel's options and fields are as follows:

Generate
: Generates JCL from the parameters you have specified on the panel.

Edit
: Displays an ISPF edit session for you to make changes to existing JCL.

Submit
: Submits for execution the JCL specified in the `JCL Dsn` field on this panel.

Enable CA
: Specifies whether Coverage Assistant's execution JCL is to be generated.

Enable DA
: Specifies whether Distillation Assistant's execution JCL is to be generated. If you set this option to `YES`, set `Enable UTA` to `NO`.

Enable UTA
: Specifies whether Unit Test Assistant's execution JCL is to be generated. If you set this option to `YES`, set `Enable DA` to `NO`.

Use Program Name for File Name

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, enter `YES`.

When you press Enter, the file names on the panel are changed automatically. Using the program name to construct the data set names is the normal ATC procedure.

Program Name

Name to use for ATC files if you enter `YES` in the `Use Program Name for File Name` field. Note that this 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.TEST.COB01M.BRKTAB'`

- Partitioned data sets:

  `'proj_qual.file_type(program_name)'`

  For example: `'YOUNG.TEST.BRKTAB(COB01M)'`

Session ID

Session ID for your session. This defaults to your TSO user ID. Multiple testers can use the monitor simultaneously. For more information, see “Multiple User Sessions” on page 249 for details.

Breakpoint Table Dsn

Name of the BRKTAB data set created during Setup and used by the monitor program.

Variable Control Dsn

Name of the data set containing the variable control information (created during Setup).

JCL Dsn

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

Note: 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 X*xxxxxxx*, where *xxxxxxx* is the first seven characters of the program name.

Breakout Dsn

Name of the BRKOUT data set created during Execution and used by the Report program.

Variable Table Dsn

Name of the variable table data set, which is a work file containing intermediate results of information gathered when variables were monitored.

Parameters for Start Monitor (CMDUSVC)

You can provide the following input parameters to the start monitor routine (CMDUSVC):

Session type

START The user is trying to start a new session (with the ID passed in the Session ID parameter). Up to 32 sessions can be active on the same MVS system.

Session ID An eight-character string that identifies the session.

Parameters for Variable Monitor (VARMON3)

You can provide the following input parameters to the variable monitor routine.

Session ID An eight-character string that identifies the session. This **must** match the Session ID passed to CMDUSVC.

MinWait The minimum wait time to write buffers (in 1/100 seconds)

MaxWait The maximum wait time to write buffers (in 1/100 seconds)

Multiple User Sessions

More than one tester can execute simultaneously on an MVS system. Each separate invocation of the monitor is called a session. The monitor identifies sessions 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.

Change the session ID to a user-defined ID by either 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 you choose:

- The second qualifier in the following data set names:
  - prefix.sessid.EXTEMP.EXEC
  - prefix.sessid.TEMP.VMTEMP
- The second parm to the CMDUSVC program
- The first parm to the VARMON3 program (optional; only used for DA and UTA)

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 CASESSN command). This may 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 execute shared modules, coverage 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 the following rules apply. When multiple testers execute a monitored module, and the module is monitored by:

1. Only one session (its BRKTAB appears in only one session), all coverage from all testers appears in that session's coverage data.
2. 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 topics in this chapter describe various scenarios in which the monitor may be used:

1. “Multiple Testers Executing Code in a Module Monitored by One Session” on page 251
2. “Multiple Testers Executing Code in a Module Monitored by Multiple Sessions” on page 252
3. “Multiple Testers Executing Code in a Module, but Each Tester with a Unique Copy of the Module” on page 253

Multiple Testers Executing Code in a Module Monitored by One Session:
This example describes multiple testers executing the same module, but only one session is monitoring the module:

Table 4. Multiple Testers Executing Code in a Module Monitored by One Session

	Setup	Start Session	Execute Code
Tester 1	• Listings for module A, module C, and module D • Instrumented objects linked into executable modules A through F	• Session 1 • BRKTABs from module A, module C, and module D	In modules A through F
	Note: Tester 1 is monitoring the coverage of modules A, C, and D, but running test cases that execute modules A through F.		
Tester 2	• Listings for module B and module E • Instrumented objects linked into executable modules A through F	• Session 2 • BRKTABs from module B and module E	In modules A through F
	Note: Tester 2 is monitoring modules B and E, but executing test cases that execute modules A through F.		
Tester 3	• Listings for module F • Instrumented objects linked into executable modules A through F	• Session 3 • BRKTABs from module F	In modules A through F
	Note: Tester 3 is monitoring module F, but executing test cases that execute modules A through F.		

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.

Multiple Testers Executing Code in a Module Monitored by Multiple Sessions: This example describes multiple testers executing the same module, with multiple sessions monitoring identical module.

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

Table 5. Multiple Testers Executing Code in a Module Monitored by Multiple Sessions

	Setup	Start Session	Execute Code
Tester 1	• Listings for modules A through F • Instrumented objects linked into executable modules A through F	• Session 1 • BRKTABs from module A, module C, and module D	In modules A through F
	Note: In session 1, Tester 1 is monitoring the coverage of modules A, C, and D, but running test cases that execute Modules A through F.		
Tester 2	• Listings for modules A through F • Instrumented objects linked into executable modules A through F	• Session 2 • BRKTABs from module B, module C, and module E	In modules A through F
	Note: In session 2, Tester 2 is monitoring modules B, C and E, but executing test cases that execute modules A through F.		
Tester 3	• Listings for modules A through F • Instrumented objects linked into executable modules A through F	• Session 3 • BRKTABs from module B and module F	In modules A through F
	Note: In session 3, Tester 3 is monitoring modules B and F, but executing test cases that execute modules A through F.		

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

Any coverage in 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 will now appear 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.

Multiple Testers Executing Code in a Module, but Each Tester with a Unique Copy of the Module: Each tester can ensure unique coverage data for test cases run by that tester, by using the following procedure:

Table 6. Multiple Testers Executing Code in a Module, but Each Tester with a Unique Copy of the Module

	Setup	Start Session	Execute Code
Tester 1	• Listings for modules A through F • Link into modules A1 through F1	• Session 1 • BRKTABs from modules A through F from Tester 1 Setup	In modules A1 through F1
Tester 2	• Listings for modules A through F • Link into modules A2 through F2	• Session 2 • BRKTABs from modules A through F from Tester 2 Setup	In modules A2 through F2
Tester 3	• Listings for modules A through F • Link into modules A3 through F3	• 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 A*x* through F*x*. The coverage statistics represent testing done by an individual tester and no one else.

Using the Performance Mode to Reduce Monitor Overhead

Measuring when a conditional branch branches takes more overhead because the breakpoint at the conditional branch must be left in storage. If the increased overhead is unacceptable for your testing, conditional branch coverage can be turned off. You can do this by setting the Performance Mode. Set the Performance Mode in either of the following ways:

1. Change a default setting. Change the Performance Mode default (in the Setup Defaults section of ATC Defaults) to Yes. For more information, see “Modifying Your ATC Defaults” on page 27. Any BRKTAB files created in the Setup step will have a flag set to indicate that conditional branches should **not** be left in storage.
2. Issue a command to turn Performance Mode on temporarily during testing. After a session is started, turn Performance Mode on or off by entering the following commands:
   a. `EX '`*`hi_lev_qual`*`.V1R5M0.REXX(CAPRFON)'` turns the Performance Mode on
   b. `EX '`*`hi_lev_qual`*`.V1R5M0.REXX(CAPRFOFF)'` turns the Performance Mode off

When the Performance Mode is set on via SETUP, the summary report for the test run will not show conditional branch coverage. For more information, see “Suppression of Conditional Branch Coverage with Performance Mode” on page 87. The annotation for conditional statements in an annotated listing is also modified. For more information, see “Changes in Annotation Symbols with Performance Mode” on page 104.

When the Performance Mode is set off via SETUP, and then later enabled via CAPRFON, then summary reports and annotated listings will still include conditional breakpoint information, although the data may be incomplete.

Buffer Monitor

A buffer monitor program is available for UTA and DA. The buffer monitor stays resident while your session is active in order to write full buffers of data to disk. Two buffers are used: one is written to disk and the other saves data.

The size of each buffer is 65536 bytes. These buffers reside in ESQA storage.

You can specify the minimum and maximum times that the buffer monitor program checks for full buffers. As shipped, the minimum time is 20 (.2 seconds) and the maximum time is 800 (8 seconds). These times are in hundredths of a second. Valid settings range from 20 to 1600. If you try to set times outside of this range, a warning message is displayed and the time is set within this range.

Your program may access variables to be saved faster than they can be written to disk, which causes a buffer overflow and a loss of data occurs. The following statement appears in the VARDATA report if a buffer overflow and a loss of data occurs:

```
11915        1        1        38     KYLEX  3722AIX.
11916        1        1        38     KYLEX  3723AIX.
******** !!! BUFFER OVERFLOW !!! ********
11917        1        1        38     KYLEX  0000MVS.
```

The buffer monitor program is resident as long as your session is active. The job that started your monitor session stays active until you end your session with the CASTOP or CAQUIT command. This is not true for CA monitor sessions. For CA sessions, there is no buffer monitor program, and your start monitor job ends once your monitor session has been established.

Monitor Commands

Several commands are available to control the execution monitor and to display statistics. These commands, which are described in this chapter, can only be executed while the monitor is running.

Note: A common monitor program executes for the CA, DA, and UTA tools and the commands described in this chapter are valid for all three tools.

The command execs are shipped in a partitioned data set (PDS) named *hi_lev_qual*.V1R5M0.REXX. Any status messages resulting 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.

Issuing Commands

You can issue commands in the following ways:

- From the `Control the CA/DA/UTA Monitor` panel, shown in Figure 81 on page 256. The commands are listed on the panel. If you select a command, a panel is displayed allowing you to enter any command parameters.
- From the TSO command line. Some commands have optional parameters that can be passed. The explicit invocation shows the method for passing parameters to the command. The following is an example of issuing a command from the TSO command line:

```
EX 'hi_lev_qual.V1R5M0.REXX(cmdname)' 'parm1 parm2'
```

- From MVS BATCH using JCL. This method is useful in automating test case runs. The JCL can be embedded in your batch stream. When the commands are executed in batch mode, the MSGS.FILE is appended with messages from each command executed in the job. You can view this file for problem determination. The following is an example of issuing commands using JCL:

```
//YOUNGC    JOB (12345678),
//            YOUNG,NOTIFY=YOUNG,USER=YOUNG,
//            TIME=1,MSGCLASS=H,CLASS=A,REGION=2M
//* SAMPLE JCL TO EXECUTE CA COMMANDS:
//* FIRST A CASTATS COMMAND IS EXECUTED, WITH RESULTS IN
//*   YOUNG.MSGS.FILE
//* NEXT A CASTOP COMMAND IS EXECUTED
//TSOTMP EXEC PGM=IKJEFT01,DYNAMNBR=30,REGION=4096K
//SYSEXEC  DD DSN=hi_lev_qual.V1R5M0.REXX,DISP=SHR
//SYSTSPRT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//SYSTSIN   DD *
     CASTATS YOUNG 1 1 9999
     CASTOP
/*
```

To see a list of the ATC operator commands common to CA, DA, and UTA, select option `1` from the `ATC Primary Option Menu` panel and option `7` from the `Coverage, Distillation, and Unit Test Assistant` panel. This displays the `Control the CA/DA/UTA Monitor` panel, shown in Figure 81 on page 256, which contains a list of commands you can select.

© Copyright IBM Corp. 1997, 1999

Each command on the `Control the CA/DA/UTA Monitor` panel is described in more detail in the topics that follow in this chapter.

```
------------------------ Control the CA/DA/UTA Monitor -----------------------
Option ===>

1  Start         Create JCL to Start the Monitor

2  Stop          Stop monitor execution normally                  (CASTOP)

3  SessDisplay   Display all active sessions                      (CASESSN)

4  Listings      Display listings                                 (CALIST)
5  Statistics    Display statistics                               (CASTATS)
6  BPDisplay     Display Breakpoint status                        (CABPDSP)
7  VADisplay     Display Variable status                          (CAVADSP)

8  AddId         Specify a unique testcase id                     (CAIDADD)
9  Snapshot      Take snapshot of data                            (CADATA)
10 Reset         Reset all data in monitor                        (CARESET)

11 Quit          Terminate monitor without saving breakpoint data (CAQUIT)

Enter END to Terminate
```

Figure 81. Control the CA/DA/UTA Monitor Panel

CABPDSP

The CABPDSP command, issued from the panel shown in Figure 82, 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
```

Figure 82. Monitor: Display Breakpoint Status Panel

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

You can enter this command on the command line as follows:

```
EX 'hi_lev_qual.V1R5M0.REXX(CABPDSP)' 'session_id list pa_num bp_start bp_end'
```

where:

session_id Session ID to be displayed.

list Listing number (the default is 1).

pa_num PA number (the default is 1) in this listing. The PA number is *1 origined* for the specified listing.

bp_start BP number of the first breakpoint in the PA you want displayed (the default is 1).

bp_end BP number of the last breakpoint in the PA you want displayed (the default is 9999).

Example 1

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

```
EX 'hi_lev_qual.V1R5M0.REXX(CABPDSP)'
```

Example 2

To display all breakpoints for session YOUNG in LIST 1, PA 2 with a bp≥10 and ≤50, enter this command:

```
EX 'hi_lev_qual.V1R5M0.REXX(CABPDSP)' 'YOUNG 1 2 10 50'
```

A status panel, such as that shown in Figure 83, is displayed.

```
 ------------------------------------------------------------------------------
 BROWSE    YOUNG.MSGS.FILE                          Line 00000000 Col 001 080
 Command ===>                                          Scroll ===> CSR
******************************** Top of Data ********************************
Num Listing                                  Date   Time     PAs   BPs    VAs
------------------------------------------------------------------------------
001 ATC.V1R5M0.SAMPLE.COBOLST(COB01AM)       98.121 09:35.59 00005 000044 00007
PA    ADR      BPS    EVNTS      ACTVE
------------------------------------------------------------------------------
00001 00000000 000006 0000000000 000006
RNUM OFFSET OPCD     BRNCH TO EVENTS    1ST KY VRIX LB NB AB DE CT CF BV B> AC
------------------------------------------------------------------------------
9BC8 0002A4 D2078020 00000000 000000000 000000       X  X                   X
A353 0002AA D2048020 00000000 000000000 000000 0000  X  X                   X
BDEF 0002B0 D2028025 00000000 000000000 000000 0001  X  X                   X
2E7C 0002B6 D2018020 00000000 000000000 000000       X  X                   X
59AF 0002BC D2028022 00000000 000000000 000000 0002  X  X                   X
199A 0002C2 D2028025 00000000 000000000 000000 0003  X  X                   X
******************************** Bottom of Data ********************************
```

Figure 83. Breakpoint Status Panel

The following fields are displayed for the selected listing and PA:

Num

Sequential listing number.

Listing

Listing data set name.

Date
Date of compile

Time
Time of compile

PAs
Number of PAs in listing

BPs
Number of BPs in listing

VAs
Number of variable reads in listing

PA
Sequential PA number.

ADR
If the program has executed, this is the storage address of the PA.

BPS
Number of BPs in this PA.

EVNTS
Number of events (BP executions) for this PA.

ACTVE
Number of active BPs in the PA.

The following fields are displayed for each breakpoint:

RNUM
Random number for this long SVC BP (0 if short SVC BP).

OFFSET
The hexadecimal offset of the breakpoint in the PA.

OPCD (op code)
The op code of the instruction at the breakpoint.

BRNCH TO (branch to)
If this is a branch instruction that has branched, this is the target address. If it starts with FF, the address is an offset within this PA.

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

1ST KY
For Distillation Assistant, the key number that caused the BP to execute for the first time.

VRIX
The index into the VARCTL file for a UTA/DA read at this BP.

LB
For UTA/DA, this BP left in storage.

NB (not a branch)
If not a branch instruction, this is an X.

AB (always branch)
If an unconditional branch instruction, this is an X.

DE (dummy entry)
If a dummy entry, this is an X. A dummy entry is the instruction after a conditional branch instruction that is breakpointed to tell when the branch falls through.

CT (condition true)
A conditional branch instruction has branched.

CF (condition false)
A conditional branch instruction 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 following this instruction is executed, this breakpoint is updated as fallen through.

AC (active)
The breakpoint is active (invalid instruction still present).

CADATA

The CADATA command, issued from the panel shown in Figure 84, writes the coverage statistics (BRKOUT) to the file name specified on the panel.

```
------------------------ 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)
```

Figure 84. Monitor: Take Snapshot of Data—Panel 1

When you press Enter on this panel, the monitor displays the panel shown in Figure 85, allowing 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. . . . . .
```

Figure 85. Monitor: Take Snapshot of Data—Panel 2

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

You can enter this command on the command line as follows:

```
EX 'hi_lev_qual.V1R5M0.REXX(CADATA)' 'test_id, session_id'
```

where:

test_id Assigned test ID.

session_id Session ID for the snapshot.

Example 1

To allow the test ID to default to a time stamp and the BRKOUT name to be built using that same time stamp, enter this command:

```
EX 'hi_lev_qual.V1R5M0.REXX(CADATA)'
```

Example 2

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

```
EX 'hi_lev_qual.V1R5M0.REXX(CADATA)' 'TCASE1'
```

A status panel, such as that shown in Figure 86, 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 *******************************
```

Figure 86. Snapshot Status Panel

All parameters are optional. If you invoke the command with no parameters, an exec named prefix.sessionid.EXTEMP.EXEC, which is built during execution, runs.

The session ID defaults to the TSO user ID.

The test ID, if entered, is used to build the BRKOUT file name. If you do not provide the test ID parameter, the default time stamp (date and time) is used. The BRKOUT file name is created using the day of the year and the number of seconds that have elapsed since the start of the day.

Note that the parameters are separated by commas; therefore, to code a null parameter, enter a comma in the parameter's position.

CAIDADD

The CAIDADD command, issued from the panel shown in Figure 87, allows you to add a unique test case ID.

```
------------------------------ Monitor: Add ID -------------------------------
Option ===>_

To assign a test case ID, complete the menu below and press ENTER:

  Test Case ID . . . . .

  Session ID . . . . . .   YOUNG
```

Figure 87. Monitor: Add ID Panel

The test case ID is printed in the summary report.

You can enter this command on the command line as follows:

```
EX 'hi_lev_qual.V1R5M0.REXX(CAIDADD)' 'test_id, session_id'
EX 'hi_lev_qual.V1R5M0.REXX(CAIDADD)' 'TCASE1'
```

where:

test_id Assigned test ID.

session_id Session ID to be added.

A status panel, such as that shown in Figure 88, 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 *******************************
```

Figure 88. Add ID Status Panel

Test ID is an optional parameter and defaults to a time stamp comprised of the date and time the command was invoked. Session ID is also optional and defaults to the TSO user ID. The parameters are delimited with commas; therefore, if you want to code the session ID and set the test ID to the default value, code a single comma for test ID. The test case ID is printed in the summary report.

CALIST

The CALIST command, issued from the panel shown in Figure 89, allows you 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
```

Figure 89. Monitor: Display Listings

You can enter this command on the command line as follows:

```
EX 'hi_lev_qual.V1R5M0.REXX(CALIST)' 'session_id start_list end_list'
```

where:

session_id The session for which to display statistics.

start_list The first listing number to be displayed (the default is 1).

end_list The last listing number to be displayed (the default is 9999).

The session ID is an optional parameter used to indicate which session table you want to display. It defaults to the TSO user ID. If start_list and end_list are passed, then session_id must also be passed, even if the default user ID is used.

Example 1:

To display statistics on all listings, enter this command:

```
EX 'hi_lev_qual.V1R5M0.REXX(CALIST)'
```

Example 2:

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

```
EX 'hi_lev_qual.V1R5M0.REXX(CALIST)' 'YOUNG 2'
```

Example 3:

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

```
EX 'hi_lev_qual.V1R5M0.REXX(CALIST)' 'YOUNG 2 4'
```

A statistics panel, such as that shown in Figure 90 on page 263 is displayed.

```
 ------------------------------------------------------------------------------
 BROWSE    YOUNG.MSGS.FILE                          Line 00000000 Col 001 080
 Command ===>                                          Scroll ===> CSR
******************************** Top of Data ********************************
Num Listing                                  Date   Time     PAs   BPs    VAs
------------------------------------------------------------------------------
001 ATC.V1R5M0.SAMPLE.COBOLST(COB01AM)       98.121 09:35.59 00005 000044 00007
002 ATC.V1R5M0.SAMPLE.COBOLST(COB01CM)       98.121 09:36.06 00002 000022 00002
003 ATC.V1R5M0.SAMPLE.COBOLST(COB01DM)       98.121 09:36.13 00002 000017 00000
******************************* Bottom of Data ******************************
```

Figure 90. Listings Statistics Panel

For each listing, the following fields are displayed:

Num Sequential listing number
Listing Listing data set name
Date Date of compile
Time Time of compile
PAs Number of PAs in listing
BPs Number of BPs in listing
VAs Number of variable reads in listing

CAPRFOFF

The CAPRFOFF command turns the monitor Performance Mode off. This enables conditional branch coverage. The BPs needed for conditional coverage are left in storage. Note that overhead will be higher when these BPs are left in storage.

If you are not interested in conditional coverage, turn the Performance Mode on, either by changing the default parameter that controls Performance Mode before you generate the JCL for the Setup step, or by using the CAPRFON command. Performance mode can be turned on and off for all PAs or for a selected PA.

You enter this command on the command line as follows:

```
EX 'hi_lev_qual.V1R5M0.REXX(CAPRFOFF)' 'session_id, PA_number'
```

where:

session_id The session for which to turn Performance Mode off.

PA_number The PA number.

The session ID defaults to the TSO user ID. It identifies the session you want to modify. The PA number defaults to ALL for the specific session. The parameters are delimited with commas; therefore, if you want to code the PA number and set the session ID to the default value, code a single command for session ID.

Use the CASTATS command to get the PA number for selectively resetting breakpoints.

Example 1:

To turn Performance Mode off for all PAs, enter this command:

```
EX 'hi_lev_qual.V1R5M0.REXX(CAPRFOFF)'
```

Example 2:

To turn Performance Mode off for session YOUNG, PA 2, enter this command:

```
EX 'hi_lev_qual.V1R5M0.REXX(CAPRFOFF)' 'YOUNG, 2'
```

CAPRFON

The CAPRFON command turns the monitor Performance Mode on. This disables conditional branch coverage. The BPs needed for conditional coverage are not left in storage. Note that overhead will be higher when these BPs are left in storage. If you are not interested in conditional coverage, turn the Performance Mode on, either by changing the default parameter that controls Performance Mode before you generate the JCL for the Setup step or by using the CAPRFON command. Performance Mode can be turned on and off for all PAs or for a selected PA.

You enter this command on the command line as follows:

```
EX 'hi_lev_qual.V1R5M0.REXX(CAPRFON)' 'session_id, PA_number'
```

where:

session_id The session for which to turn Performance Mode on.

PA_number The PA number.

The session ID defaults to the TSO user ID. It identifies the session you want to modify. The PA number defaults to ALL for the specific session. The parameters are delimited with commas; therefore, if you want to code the PA number and set the session ID to the default value, code a single command for session ID.

Use the CASTATS command to get the PA number for selectively resetting breakpoints.

Note: If the BRKTAB for this session was built with the Performance Mode flag set to N (off), then the Summary and Annotated listings will still include (possibly incomplete) conditional breakpoint information after this command is issued.

Example 1:

To turn Performance Mode on for all PAs, enter this command:

```
EX 'hi_lev_qual.V1R5M0.REXX(CAPRFON)'
```

Example 2:

To turn Performance Mode on for session YOUNG, PA 2, enter this command:

```
EX 'hi_lev_qual.V1R5M0.REXX(CAPRFON)' 'YOUNG, 2'
```

CAQUIT

The CAQUIT command, issued from the panel shown in Figure 91, is the same as CASTOP, except that no output file is written (BRKOUT or VARTAB).

```
---------------------------- Monitor: Quit Monitor -----------------------------
Option ===>_

Quit Monitor Without Saving Breakpoint Data:
  Override default session id NO  (Yes|No)
```

Figure 91. Monitor: Quit Monitor—Panel 1

When you press Enter from this panel, the monitor displays the panel shown in Figure 92, allowing you to enter additional parameters.

```
---------------------------- Monitor: Quit Monitor -----------------------------
Option ===>_

Press ENTER to Quit Monitor Without Saving Breakpoint Data:
  Override default session id NO  (Yes|No)
  Session Id . . . .
```

Figure 92. Monitor: Quit Monitor—Panel 2

You can enter this command on the command line as follows:

```
EX 'hi_lev_qual.V1R5M0.REXX(CAQUIT)' 'session_id'
```

where:

session_id Session ID to cancel.

When the quit command completes, a panel such as the one shown in Figure 93 is displayed.

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

******************************** Top of Data ********************************
Session YOUNG ended no data written.
****************************** Bottom of Data ******************************
```

Figure 93. Quit Monitor Status Panel

Session ID is an optional parameter used to indicate which session to cancel. If not specified, the session ID defaults to the *TSO user ID*.

CARESET

The CARESET command, issued from the panel shown in Figure 94, resets all statistics in the current monitor session to zero.

```
---------------------- Monitor: Reset All Data in Monitor ----------------------
Option ===>_

To reset data to zero, complete the menu below and press ENTER:

  Session ID . . . . . .   YOUNG

  PA Number. . . . . .   ALL
```

Figure 94. Monitor: Reset All Data in Monitor Panel

The monitor resumes updating statistics immediately.

You can enter this command on the command line as follows:

```
EX 'hi_lev_qual.V1R5M0.REXX(CARESET)' 'session_id, PA_number'
```

where:

session_id The session ID for which to reset statistics.

PA_number The PA number.

The session ID defaults to the TSO user ID. It identifies the session you want to reset. The PA number defaults to ALL for the specific session. The parameters are delimited with commas; therefore, if you want to code the PA number and set the session ID to the default value, code a single comma for session ID.

Use the CASTATS command to get the PA number for selectively resetting breakpoints.

CASESSN

The CASESSN command displays a list of the current active sessions. Figure 95 shows a typical display in response to selecting option 3 (`Display all active sessions`) on the `Control the CA/DA/UTA Monitor` panel.

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

Figure 95. Active Session Display Panel

Use this command to identify session names or currently active users who need to stop or cancel their sessions before the monitor is terminated.

You can enter this command on the command line as follows:

```
EX 'hi_lev_qual.V1R5M0.REXX(CASESSN)'
```

To display the current release of the ATC monitor, use the *level* parameter. For example:

```
EX 'hi_lev_qual.V1R5M0.REXX(CASESSN)' 'level'
```

The CASESSN command with this parameter displays the release level and table address data used by ATC support. For example:

```
 ------------------------------------------------------------------------------
 BROWSE    YOUNG.MSGS.FILE                              Line 00000000 Col 001 080
 Command ===>                                                 Scroll ===> CSR
******************************** Top of Data ********************************
Monitor Release: V1R5M0 Date: 1998.346
MAST: 00F9B608 PSA: 00F87000 CPU:  00F87000 SEST: 00F9B2F0 UNID: 00000000
SESSION ID: YOUNG    PA: 01A4B990 BP: 01A36F10 VA: 019F7CA0
****************************** Bottom of Data ******************************
```

Figure 96. Release Level and Table Address Data

CASTATS

The CASTATS command, issued from the panel shown in Figure 97, allows you to select a session ID and PAs 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
```

Figure 97. Monitor: Display Statistics Panel

You can enter this command on the command line as follows:

```
EX 'hi_lev_qual.V1R5M0.REXX(CASTATS)' 'session_id start_PA end_PA'
```

where:

session_id The session ID for which to display statistics.

list The listing number to display (the default is 1).

start_PA The first PA number to be displayed (the default is 1) in this listing. The start_PA number is *1 origined* for the specified listing.

end_PA The last PA number to be displayed (the default is 9999). The end_PA number is *1 origined* for the specified listing.

The session ID is an optional parameter used to indicate which session result table you want to display. It defaults to the TSO user ID. If start_PA and end_PA are passed, then session_id must also be passed, even if the default user ID is used.

Example 1:

To display statistics on all PAs, enter this command:

```
EX 'hi_lev_qual.V1R5M0.REXX(CASTATS)'
```

Example 2:

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

```
EX 'hi_lev_qual.V1R5M0.REXX(CASTATS) 'YOUNG 1 1'
```

Example 3:

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

```
EX 'hi_lev_qual.V1R5M0.REXX(CASTATS) 'YOUNG 1 2 4'
```

A statistics panel, such as that shown in Figure 98 on page 269, is displayed.

```
 ------------------------------------------------------------------------------
 BROWSE    YOUNG.MSGS.FILE                          Line 00000000 Col 001 080
 Command ===>                                          Scroll ===> CSR
******************************** Top of Data ********************************
Num Listing                                  Date   Time     PAs   BPs    VAs
------------------------------------------------------------------------------
001 ATC.V1R5M0.SAMPLE.COBOLST(COB01AM)       98.121 09:35.59 00005 000044 00007
PA    ADR      BPS    EVNTS      ACTVE
------------------------------------------------------------------------------
00001 00000000 000006 0000000000 000006
00002 00000000 000018 0000000000 000018
00003 00000000 000001 0000000000 000001
00004 00000000 000017 0000000000 000017
00005 00000000 000002 0000000000 000002
------------------------------------------------------------------------------
TOTAL 00000000 000044 0000000000 000044
****************************** Bottom of Data ******************************
```

Figure 98. PA Statistics Panel

For each PA, the following fields are displayed:

Num
: Sequential listing number.

Listing
: Listing data set name.

Date
: Date of compile.

Time
: Time of compile.

PAs
: Number of PAs in listing.

BPs
: Number of BPs in listing.

VAs
: Number of variable reads in listing.

PA
: Sequential PA number of PAs in listing.

ADR
: When PA has executed, the storage address of the PA.

BPS
: Number of BPs in PA.

EVNTS (events)
: Number of BPs that have executed in the PA.

ACTVE (active)
: Number of BPs still in storage in the PA.

CASTOP

The CASTOP command, issued from the panel shown in Figure 99, writes current statistics to disk, removes all remaining breakpoints for all PAs 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)
```

Figure 99. Monitor: Stop Monitor—Panel 1

When you press Enter for this panel, the monitor displays the panel shown in Figure 100, allowing 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 . . . .
```

Figure 100. Monitor: Stop Monitor—Panel 2

The statistics (BRKOUT file) are written to the BRKOUT data set name as supplied in the JCL that started the monitor session. You can supply a different file name with the CASTOP command. The variable data for UTA and DA is written to the VARTAB file as supplied in the JCL that started the monitor session.

You can enter this command on the command line as follows:

```
EX 'hi_lev_qual.V1R5M0.REXX(CASTOP)' 'session_id brkout'
```

where:

session_id The session ID to stop.

brkout Name of the breakout table data set.

Warning: CASTOP does not currently support the BRKOUT parameters if the monitor session has DA or UTA enabled.

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.V1R5M0.REXX(CASTOP)'
```

Example 2

To write data from MySessId to proj_qual.TEST1.BRKOUT, enter this command:

```
EX 'hi_lev_qual.V1R5M0.REXX(CASTOP)' 'MySessID TEST1.BRKOUT'
```

When the stop command completes, a status panel, such as that shown in Figure 101, 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 *********************************
```

Figure 101. Stop Completed Panel

All parameters are optional. If you invoke the command with no parameters, an exec named prefix.userid.EXTEMP.EXEC, which is built during execution, runs. It allocates the BRKOUT and VARTAB files based on the data set names used when the monitor JCL was created.

If you supply a session ID parameter, but not a BRKOUT file parameter, an exec name prefix.sessionid.EXTEMP.EXEC, which is built during execution, runs. It allocates the BRKOUT and VARTAB files based on the data set names used when the monitor JCL was created.

If you supply a session ID and a BRKOUT file, the statistics are written to the BRKOUT file you specified.

CAVADSP

The CAVADSP command, issued from the panel shown in Figure 102, allows you to display various information on variables being monitored.

```
---------------------- Monitor: Display Variable Status ---------------------
Command ===>

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

  Session ID . . . . . .   YOUNG
```

Figure 102. Monitor: Display Variable Status Panel

You can enter this command on the command line as follows:

```
EX 'hi_lev_qual.V1R5M0.REXX(CAVADSP)' 'session_id'
```

where:

session_id The session for which to display statistics.

The session ID is an optional parameter used to indicate which session information you want to display. It defaults to the TSO user ID.

Example 1:

To display variable information, enter this command:

```
EX 'hi_lev_qual.V1R5M0.REXX(CAVADSP)'
```

Example 2:

To display variable information for session YOUNG, enter this command:

```
EX 'hi_lev_qual.V1R5M0.REXX(CAVADSP)' 'YOUNG'
```

An information panel, such as that shown in Figure 103, is displayed.

```
 ------------------------------------------------------------------------------
 BROWSE    YOUNG.MSGS.FILE                          Line 00000000 Col 001 080
 Command ===>                                          Scroll ===> CSR
******************************** Top of Data ********************************
DBGI LNTH TIMES  MAX    BLOC     BOFF EX TOT EX CNT FI PL
------------------------------------------------------------------------------
0000 0002 000001 999999 01280009 0020 000000 000000 N  N
0000 0002 000001 999999 01280009 0020 000000 000000 N  N
0000 0002 000001 999999 01280009 0020 000000 000000 N  N
0000 0003 000001 999999 01280009 0022 000000 000000 N  N
0000 0002 000001 999999 01280009 0000 000000 000000 N  N
0000 0002 000001 999999 01280009 0000 000000 000000 N  N
0000 0002 000001 000001 01280009 0030 000000 000000 N  N
0001 0002 000001 000001 01280009 0000 000000 000000 N  N
0001 0002 000001 000001 01280009 0000 000000 000000 N  N
******************************** Bottom of Data ********************************
```

Figure 103. Variable Information Panel

For each variable, the following fields are displayed:

DBGI	Index of DBGTAB section this variable read came from
LNGTH	Length of variable
TIMES	Times to read variable (1 means to read every time)
MAX	Maximum times to read variable
BLOC	Base offset and register that points to working storage
BOFF	Offset of variable in working storage
EX TOT	Total times variable read
EX CNT	Number of times BP executed since last variable read
FI	File read
PL	DA read for PL/I program

Diagnosing Monitor Problems

This chapter describes how to solve the following problems:

- 047 abend when trying to start a monitor session
- 0C1 abend on user program during test case run
- F*nn* system interruption code during test case run
- Insufficient SQA space
- Poor performance when measuring conditional branch coverage

047 Abend

If this abend occurs when submitting the monitor JCL or when using the commands, the command handler program was not in an authorized data set. See "Setting up the Authorized Data Sets" on page 16 to identify the ATC program that must meet this requirement. If you are not sure which data sets are authorized or how to copy programs into them, contact your MVS system support personnel.

Operation Exception (0C1) on User Program

If an instrumented user program is executed and a session has not been started to handle it, or the session is started with a BRKTAB file that does not match the executable program, the user program abends when the monitor cannot identify the user SVC installed as the breakpoint. When this occurs, the program terminates with an 0C1 operation exception (001 program-interruption code). The monitor forces the user program to terminate by replacing the unidentified user SVC with the invalid instruction C8, which causes the operation exception. The monitor may not recognize the user SVC as a breakpoint if:

- The listing used during Setup does not match the code in the module under test.
- The program has changed, but the Setup step has not been re-executed.
- You have not relinked your program with the modified object modules after running the Setup step.
- A monitor session that matches the code in the module under test is not running.
- You have not stopped and restarted the monitor session after the Setup step has completed.

System Interruption Code of Fnn on User Program

If the monitor is not installed and an instrumented program is executed, the user program terminates with a system interruption code of F*nn*, where *nn* is the SVC number used as the breakpoint.

© Copyright IBM Corp. 1997, 1999

Lack of SQA Space

The monitor is loaded in ECSA and SQA storage and allocates ESQA storage for the tables. For storage requirements, see “Monitor ECSA, SQA, and ESQA Usage” on page 386. If you encounter ESQA storage limitations, contact your systems programmer to see if ESQA storage can be increased.

Because of ESQA storage fragmentation, there may be sufficient free storage available, but not in a segment of sufficient size to load the tables. When a request for ESQA storage is made to the operating system, it fails if there is no free segment large enough to honor the request.

Poor Performance When Measuring Conditional Branch Coverage

Because of the change in breakpoint instrumentation (using user SVCs), measuring when a conditional branch branches takes more overhead (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. For instructions on turning off conditional branch coverage, see “Using the Performance Mode to Reduce Monitor Overhead” on page 253.

Using Source Audit Assistant

© Copyright IBM Corp. 1997, 1999

Introduction

This chapter contains the following topics:

- What Is Source Audit Assistant?
- What Does SAA Require?
- How Does SAA Work?

What Is Source Audit Assistant?

The Source Audit Assistant (SAA) is an application that compares two levels of source code, usually before and after modification. The results are placed in a comparison report, which allows you to see differences and helps you to identify items that need closer examination. SAA also supports compiler listing comparisons. Listings are converted to source code prior to comparison.

What Does SAA Require?

SAA requires the Interactive System Productivity Facility (ISPF).

Currently, SAA supports MVS and the following programming languages:

- Assembler
- C
- C++
- COBOL
- PL/I

Notes:

1. SAA does not support redefined character operators for PL/I (that is, use of the NOT or OR compiler options is not supported). The supported NOT symbol is X'5F'; the supported OR symbol is X'4F'.
2. SAA does not support templates in C++ programs.
3. If C/C++ keywords (such as, *if*, *do*, and so on) or block begin/end symbols (such as, { }) are redefined in the source, they will not be processed correctly.

SAA also supports the following types of listings:

- IBM Assembler H Version 2
- IBM High Level Assembler Version 1 Releases 1, 2, and 3
- IBM COBOL for OS/390 & VM 2.1 (plus Millennium Language Extensions [MLE])
- IBM COBOL for MVS & VM 1.2 (plus Millennium Language Extensions [MLE])
- IBM VS COBOL II Release 4.0
- IBM OS/VS COBOL Release 2.4
- IBM PL/I for MVS & VM 1.1.1 (plus Millennium Language Extensions [MLE])
- IBM OS PL/I Optimizing Compiler 2.3.0
- IBM PL/I Optimizing Compiler 1.5.1

You specify these listing types by entering `LPL/I`, `LCOB`, or `LASM` in the language field on the `Execute Source Audit Assistant` panel. When you generate this listing, specify the appropriate language processor option to produce a listing of the

© Copyright IBM Corp. 1997, 1999

source code. For the DCB requirements of each listing type, see “Compiler Options” on page 231 in “CA, DA, and UTA Setup.”

Note: Any preprocessor source will be ignored in a PL/I listing.

How Does SAA Work?

When comparing source code, SAA lets you select filters, which gives you control over the types of information placed in the comparison report. For instance, you can tell SAA to disregard changes to comments or declaration statements. You can also specify a range of columns that are to be compared.

If you are comparing listing files, SAA converts the listings to source code prior to performing the comparison.

After generating a comparison report, you can run the SAA postprocessor. The postprocessor generates a prototype CA targeted summary control file and a change validation report. The change validation report is based on matching the comparison report against a list of variables, the seed list, which are expected to be involved in the changes. The change validation report identifies:

- All specified variables that did not appear in any new line in the SAA comparison report
- All lines in the comparison report that did not contain at least one of the specified variables.

Source Audit Assistant Samples

This chapter describes the samples delivered with SAA and how to verify your installation.

The samples are divided into two groups. The first, or **standard group**, demonstrates running SAA against two partitioned data sets using each of the filtering options. The second, or **target group**, demonstrates running SAA against a member of two partitioned data sets and then running a change validation report against that output.

Sample Data Sets

The following table lists all SAA sample data sets for the **standard group**.

Data Set Name	Description
'ATC.V1R5M0.SAMPLE.SAA.NEW.COPY'	Modified PDS of COBOL copy members.
'ATC.V1R5M0.SAMPLE.SAA.OLD.COPY'	Original PDS of COBOL copy members.
'ATC.V1R5M0.SAMPLE.SAA.COPY.B.NNN.CMP'	Comparison results using No|No|No for the Comments, Declares, and Reformatted filtering options, respectively (background execution).
'ATC.V1R5M0.SAMPLE.SAA.COPY.B.NNY.CMP'	Comparison results using No|No|Yes for the Comments, Declares, and Reformatted filtering options, respectively (background execution).
'ATC.V1R5M0.SAMPLE.SAA.COPY.B.NYY.CMP'	Comparison results using No|Yes|Yes for the Comments, Declares, and Reformatted filtering options, respectively (background execution).
'ATC.V1R5M0.SAMPLE.SAA.COPY.B.YYY.CMP'	Comparison results using Yes|Yes|Yes for the Comments, Declares, and Reformatted filtering options, respectively (background execution).
'ATC.V1R5M0.SAMPLE.SAA.COPY.NNN.CMP'	Comparison results using No|No|No for the Comments, Declares, and Reformatted filtering options, respectively (foreground execution).
'ATC.V1R5M0.SAMPLE.SAA.COPY.NNY.CMP'	Comparison results using No|No|Yes for the Comments, Declares, and Reformatted filtering options, respectively (foreground execution).
'ATC.V1R5M0.SAMPLE.SAA.COPY.NYY.CMP'	Comparison results using No|Yes|Yes for the Comments, Declares, and Reformatted filtering options, respectively (foreground execution).
'ATC.V1R5M0.SAMPLE.SAA.COPY.YYY.CMP'	Comparison results using Yes|Yes|Yes for the Comments, Declares, and Reformatted filtering options, respectively (foreground execution).
'ATC.V1R5M0.SAMPLE.SAA.NEW.COBLST'	PDS containing a COBOL listing of a modified program.
'ATC.V1R5M0.SAMPLE.SAA.OLD.COBLST'	PDS containing a COBOL listing of an original program.
'ATC.V1R5M0.SAMPLE.SAA.COBLST.CMP'	Comparison results between the two COBOL listing data sets (foreground execution).

© Copyright IBM Corp. 1997, 1999

The following table lists all SAA sample data sets for the **target group**.

Data Set Name	Description
'ATC.V1R5M0.SAMPLE.SAA.NEW.COBOL(COBOLM)'	Modified PDS COBOL member for demonstrating SAA change validation reports.
'ATC.V1R5M0.SAMPLE.SAA.OLD.COBOL(COBOLM)'	Original PDS COBOL member for demonstrating SAA change validation reports.
'ATC.V1R5M0.SAMPLE.SAA.COBOLM.CMP'	Comparison results using No|No|No as filtering options (foreground execution).
'ATC.V1R5M0.SAMPLE.SAA.SEEDLIST(COBOLM)'	Input seed list for SAA change validation reports.
'ATC.V1R5M0.SAMPLE.SAA.COBOLM.REP'	Change validation report using 'ATC.V1R5M0.SAMPLE.SAA.SEEDLIST(COBOLM)' and 'ATC.V1R5M0.SAMPLE.SAA.COBOLM.CMP'.
'ATC.V1R5M0.SAMPLE.SAA.COBOLM.TARGCTL'	A skeleton CA targeted summary control file (listing DSName must be modified).

Execution and Verification

The information in this topic explains how to verify the SAA installation by running the sample programs and comparing your output with the output that is shipped with the package.

As a member of the ATC collection of tools, you start SAA from the ATC main menu. To invoke ATC, issue the following command from ISPF option 6:

```
EXEC 'hi_lev_qual.V1R5M0.REXX(ATSTART)'
```

The `ATC Primary Option Menu`, shown in Figure 104, is displayed.

```
------------------------ ATC Primary Option Menu V1R5M0 -----------------------
Option ===>_

0  Defaults      Manipulate ATC defaults
1  CA/DA/UTA     Coverage, Distillation and Unit Test Assistant
2  SAA           Source Audit Assistant
3  SINFO         SInfo Assistant

Enter X to Terminate
```

Figure 104. ATC Primary Option Menu

Select option 2 to start SAA. The `Execute Source Audit Assistant` panel, shown in Figure 105 on page 283, is displayed.

```
------------------------ Execute Source Audit Assistant ----------------------
Option ===>

1  Background    Execute as Batch Job(s) via generated JCL
2  Foreground    Execute in the Foreground under ISPF
3  Compare       View Compare Dsn
4  Log           View Log Dsn
5  Postprocessor Validate Changes & Create Prototype Target Control Dsn

Enter END to Terminate
```

Figure 105. Execute Source Audit Assistant Panel

Standard Group - Background Execution

To run the **standard group** of samples in the background, complete the following steps to generate a comparison report:

1. Select option `1` on the `Execute Source Audit Assistant` panel. The panel shown in Figure 106 is displayed.

```
---------------------- SAA Background Execution Parameters ---------------------
Option ===>

New Source Dsn: . . .  'ATC.V1R5M0.SAMPLE.SAA.NEW.COPY(*)'

Old Source Dsn: . . .  'ATC.V1R5M0.SAMPLE.SAA.OLD.COPY(*)'

Output Compare Dsn: .  'yourid.SAMPLE.SAA.COPY.B.xxx.CMP'

Output Log Dsn: . . .  'yourid.SAMPLE.SAA.COPY.B.xxx.LOG'

Programming Language:  COBOL   (ASM,C,C++,COBOL,PL/I,LASM,LCOB,or LPL/I)

Select Line Audit Filters (Y = apply filter, N = do not apply filter):
  Comments  Y (Y or N)   Declares  Y (Y or N)   Reformatted  Y (Y or N)

Select Comparison Columns:                    Select Execution Option:
  Start Col 1   (1-176)   End Col 72  (1-176)    Edit JCL Y (Y or N)

DBCS support:
  Enable  N (Y or N)

Press END to Terminate
```

Figure 106. SAA Background Execution Parameters Panel

2. Complete the `SAA Background Execution Parameters` panel as follows, and then press Enter:
   - Replace ATC with the high-level qualifier under which ATC was installed.
   - Replace *yourid* with your TSO user ID, your TSO prefix, or a high-level qualifier other than the one with which you replaced ATC.

- Replace *xxx* with the first letter of each value you enter for the filter options if you want to verify each sample data set shipped with SAA.

  For example, if you enter `Yes` for each filter, you would replace *xxx* with *YYY*. If you enter `Yes` for the Comments filter, `No` for the Declares filter, and `Yes` for the Reformatted filter, you would replace *xxx* with *YNY*.
- Change the value for any of the filter options that you choose. Any combination of `Yes` or `No` values is acceptable. SAA supplies samples for four of the possible combinations.
- Change the range of columns to be compared. For the samples, change the `Start Col` value to `1`, and change the `End Col` value to `72`.
- Change the value of the Edit JCL field. If you enter:
  - `Y` (Yes), SAA generates the JCL stream in a temporary file and displays an ISPF edit session for that file. From the edit session, you can:
    - SUBMIT the JCL to run the background compare. The message `IKJ56250I JOB XXXXXXX1(JOBXXXXX) SUBMITTED` is displayed upon successful submission of the job.
    - Close the session without submitting the job by pressing the END key (PF3) or typing CANCEL on the command line and pressing Enter.
  - `N` (No), SAA generates the JCL and submits the job to run.
- Change the value of the DBCS support `Enable` field. When running the samples, set the value to `N`.

Figure 106 on page 283 shows an example of the panel.

The background execution places the comparison report and log file into the data sets specified on the `SAA Background Execution Parameters` panel.

Use an output viewing utility such as SDSF to see the completion codes of the batch job.

Note: If syntax problems occur during the analysis, they are written to the analysis log, which you can view by selecting option `4` from the `Execute Source Audit Assistant` panel.

3. After the SAA background job has executed, press the End key (PF3) to return to the `Execute Source Audit Assistant` panel.

   Note: If you want to view the comparison report, select option `3` on the `Execute Source Audit Assistant` panel.
4. Verify the SAA installation by comparing the comparison report with the SAA-supplied data set that was created using the same filter option values that you used (that is, ATC.V1R5M0.SAMPLE.SAA.COPY.B.*xxx*.CMP).
5. After you finish viewing the comparison report, press the End key (PF3) to return to the `Execute Source Audit Assistant` panel.

Standard Group - Foreground Execution

To run the **standard group** of samples in the foreground, complete the following steps to generate a comparison report:

1. Select option 2 on the Execute Source Audit Assistant panel. The panel shown in Figure 107 is displayed.

```
---------------------- SAA Foreground Execution Parameters ----------------------
Option ===>

New Source Dsn: . . .  'ATC.V1R5M0.SAMPLE.SAA.NEW.COPY(*)'

Old Source Dsn: . . .  'ATC.V1R5M0.SAMPLE.SAA.OLD.COPY(*)'

Output Compare Dsn: .  'yourid.SAMPLE.SAA.COPY.xxx.CMP'

Output Log Dsn: . . .  'yourid.SAMPLE.SAA.COPY.xxx.LOG'

Programming Language:  COBOL   (ASM,C,C++,COBOL,PL/I,LASM,LCOB,or LPL/I)


Select Line Audit Filters (Y = apply filter, N = do not apply filter):
  Comments  Y (Y or N)   Declares  Y (Y or N)   Reformatted  Y (Y or N)

Select Comparison Columns:
  Start Col  1   (1-176)   End Col  72  (1-176)

DBCS support:
  Enable  N (Y or N)

Press END to terminate
```

Figure 107. SAA Foreground Execution Parameters Panel

2. Complete the SAA Foreground Execution Parameters panel as follows, and then press Enter:
   - Replace ATC with the high-level qualifier under which ATC was installed.
   - Replace *yourid* with your TSO user ID, your TSO prefix, or a high-level qualifier other than the one with which you replaced ATC.
   - Replace *xxx* with the first letter of each value you enter for the filter options if you want to verify each sample data set shipped with SAA.

     For example, if you enter Yes for each filter, you would replace *xxx* with *YYY*. If you enter Yes for the Comments filter, No for the Declares filter, and Yes for the Reformatted filter, you would replace *xxx* with *YNY*.
   - Change the value for any of the filter options that you choose. Any combination of Yes or No values is acceptable. SAA supplies samples for four of the possible combinations.
   - Change the range of columns to be compared. For the samples, change the Start Col value to 1, and change the End Col value to 72.
   - Change the value of the DBCS support Enable field. When running the samples, set the value to N.

   Figure 107 shows an example of the panel.

As SAA is executing, status messages are displayed for each data set or member that is processed. Figure 108 on page 286 shows an example of what you would see if you entered `Yes` for each filter option.

Note: Status messages **are not** displayed for options set to `No`.

```
SAAR008I Comparing Partitioned Datasets
SAAR006I Comparing member DADIV
SAAC003I New and old files are being compared
SAAC004I Comparison results are being filtered
SAAC005I Reformatted lines are being removed
SAAC006I New and old files are being analyzed
SAAC007I Comment lines are being filtered
SAAC008I Declare lines are being filtered
SAAC009I Output being formatted
SAAC010R SAA Comparison done RC = 0
SAAR006I Comparing member IDDIV
SAAC003I New and old files are being compared
SAAC004I Comparison results are being filtered
SAAC005I Reformatted lines are being removed
SAAC006I New and old files are being analyzed
SAAC007I Comment lines are being filtered
SAAC008I Declare lines are being filtered
SAAC009I Output being formatted
SAAC010R SAA Comparison done RC = 0
SAAR006I Comparing member MOVES
SAAC003I New and old files are being compared
SAAC004I Comparison results are being filtered
SAAC005I Reformatted lines are being removed
SAAC006I New and old files are being analyzed
SAAC007I Comment lines are being filtered
SAAC008I Declare lines are being filtered
SAAC009I Output being formatted
SAAC010R SAA Comparison done RC = 0
```

Figure 108. Status Messages When All Filter Options Are Yes

The foreground execution places the comparison report and log file into the data sets specified on the `SAA Foreground Execution Parameters` panel.

Note: If syntax problems occur during the analysis, they are written to the analysis log, which you can view by selecting option `4` from the `Execute Source Audit Assistant` panel.

3. After the SAA foreground job has executed, press the End key (PF3) to return to the `Execute Source Audit Assistant` panel.

   Note: If you want to view the comparison report, select option `3` on the `Execute Source Audit Assistant` panel. If you want to view the analysis log, select option `4`.

4. Verify the SAA installation by comparing the comparison report with the SAA-supplied data set (that is, ATC.V1R5M0.SAMPLE.SAA.COPY.*xxx*.CMP) that was created using the same filter option values that you used.

5. After you finish viewing the comparison report, press the End key (PF3) to return to the `Execute Source Audit Assistant` panel.

Target Group

To run the **target group** of samples, complete the following steps to generate an SAA comparison report, and then run the SAA postprocessor step:

1. Select option 2 on the `Execute Source Audit Assistant` panel. The `SAA Foreground Execution Parameters` panel is displayed.
2. Complete the panel displayed as follows and press Enter when you are finished:
   - Replace ATC with the high-level qualifier under which ATC was installed.
   - Replace *yourid* with your TSO user ID, your TSO prefix, or a high-level qualifier other than the one with which you replaced ATC.
   - Leave each filter option value as `No` for this sample.
   - Leave the DBCS support `Enable` field set to `N` for this sample.

   Figure 109 shows an example of the `SAA Foreground Execution Parameters` panel.

```
---------------------- SAA Foreground Execution Parameters ----------------------
Option ===>

New Source Dsn: . . .  'ATC.V1R5M0.SAMPLE.SAA.NEW.COBOL(COBOLM)'

Old Source Dsn: . . .  'ATC.V1R5M0.SAMPLE.SAA.OLD.COBOL(COBOLM)'

Output Compare Dsn: .  'yourid.SAMPLE.SAA.COBOLM.CMP'

Output Log Dsn: . . .  'yourid.SAMPLE.SAA.COBOLM.LOG'

Programming Language:  COBOL   (ASM,C,C++,COBOL,PL/I,LASM,LCOB,or LPL/I)


Select Line Audit Filters (Y = apply filter, N = do not apply filter):
  Comments  N (Y or N)   Declares  N (Y or N)   Reformatted  N (Y or N)

Select Comparison Columns:
  Start Col  1   (1-176)   End Col  72  (1-176)

DBCS support:
  Enable  N (Y or N)

Press END to terminate
```

Figure 109. SAA Foreground Execution Parameters Panel for Target Group Samples

As SAA is executing, the following status messages are displayed:

```
SAAC003I New and old files are being compared
SAAC004I Comparison results are being filtered
SAAC009I Output being formatted
SAAC010R SAA Comparison done RC = 0
```

Figure 110. Status Messages When All Filter Options Are No

3. Verify the SAA installation by comparing the comparison report with the SAA supplied data set (that is, ATC.V1R5M0.SAMPLE.SAA.COBOLM.CMP).
4. Press the End key (PF3) to return to the `Execute Source Audit Assistant` panel.

5. To run the SAA postprocessor step, select option `5` from the `Execute Source Audit Assistant` panel.
6. Complete the `SAA Postprocessor` panel as follows, and then press Enter:
   - Replace ATC with the high-level qualifier under which ATC was installed.
   - Replace *yourid* with your TSO user ID, your TSO prefix, or a high-level qualifier other than the one with which you replaced ATC.
   - Leave the DBCS support `Enable` field set to `N` for this sample.

   Figure 111 shows an example of the panel.

```
------------------------------ SAA Postprocessor ------------------------------
Command ===>


Input Data Sets:
SAA Compare Dsn  . . . 'yourid.SAMPLE.SAA.COBOLM.CMP'
Seed List Dsn  . . . . 'ATC.V1R5M0.SAMPLE.SAA.SEEDLIST(COBOLM)'

Output Data Sets:
Target Control Dsn . . 'yourid.SAMPLE.SAA.COBOLM.TARGCTL'
Chg Validation Rpt Dsn 'yourid.SAMPLE.SAA.COBOLM.REP'

DBCS support:
  Enable  N (Y or N)
```

Figure 111. SAA Postprocessor Panel

   After the SAA change validation report is generated, a session in which you can view the report is displayed.
7. When you are finished viewing the report, press the End key (PF3) to return to the `Execute Source Audit Assistant` panel.

Starting Source Audit Assistant

You start SAA by selecting option 2 from the ATC Primary Option Menu. From the Execute Source Audit Assistant panel, shown in Figure 112, you can select option 1 to execute SAA in the background (batch), option 2 to execute SAA in the foreground (TSO), option 3 to view the comparison report(s), or option 4 to view the log file generated from the last execution of SAA. (The log file is a collection of system messages from the most recent execution of SAA. These messages pertain to the execution of the tool rather than the content of the comparison.) You can also select option 5 to create a change validation report and a prototype CA targeted summary control file based on the changes reported.

```
------------------------ Execute Source Audit Assistant ----------------------
Option ===>

1  Background    Execute as Batch Job(s) via generated JCL
2  Foreground    Execute in the Foreground under ISPF
3  Compare       View Compare Dsn
4  Log           View Log Dsn
5  Postprocessor Validate Changes & Create Prototype Target Control Dsn

Enter END to Terminate
```

Figure 112. Execute Source Audit Assistant Panel

Executing SAA in the Background

When you select option 1 on the Execute Source Audit Assistant panel, the SAA Background Execution Parameters panel, shown in Figure 113, is displayed.

© Copyright IBM Corp. 1997, 1999

```
---------------------- SAA Background Execution Parameters ----------------------
Option ===>

New Source Dsn: . . .  'project.newgroup.cobol(*)'

Old Source Dsn: . . .  'project.oldgroup.cobol(*)'

Output Compare Dsn: .  'yourid.SAA.YYY.CMP'

Output Log Dsn: . . .  'yourid.SAA.YYY.LOG'

Programming Language:  COBOL   (ASM,C,C++,COBOL,PL/I,LASM,LCOB,or LPL/I)

Select Line Audit Filters (Y = apply filter, N = do not apply filter):
  Comments  Y (Y or N)   Declares  Y (Y or N)   Reformatted  Y (Y or N)

Select Comparison Columns:                     Select Execution Option:
  Start Col 1   (1-176)   End Col 72  (1-176)    Edit JCL Y (Y or N)

DBCS support:
  Enable  N (Y or N)

Press END to Terminate
```

Figure 113. SAA Background Execution Parameters Panel

The panel's options and fields are as follows:

New Source Dsn
Enter the name of the data set you want SAA to compare. If you are comparing a partitioned data set (PDS), include the member name, a matching pattern (for example, AB*), or an asterisk (*) to include all members.

You can compare an original *source* data set (Old Source File) with a modified *source* data set (New Source File), or you can compare *listing* data sets.

Old Source Dsn
Enter the name of the data set you want SAA to compare. If you are comparing a partitioned data set (PDS), include the member name, a matching pattern (for example, AB*), or an asterisk (*) to include all members.

Output Compare Dsn
Enter the name of the comparison report to be generated by SAA. This can be a physical sequential (PS) data set or a partitioned data set (PDS) file. You can preallocate this file or allow SAA to allocate it for you.

To preallocate the comparison file, use the following parameters:

- DSORG=PS or PO
- SPACE=(TRK, (5,5)) for PS
- SPACE=(TRK, (10,5,20)) for PO
- RECFM=FB
- LRECL=203
- BLKSIZE=27811

Background processing automatically creates a compare file if the specified output data set is not found at the time of execution. To allow for

this allocation, SAA uses the following rules to determine the data set organization (DSORG) of the output comparison file.

If the DSORG of the input New Source File and Old Source File is:

PO Output Compare File will be type PDS.

PS Output Compare File will be type sequential.

Background processing does not allow input of New Source Files and Old Source Files of mixed type DSORGs.

Notes:

1. When you use a sequential file, DISP=(MOD,KEEP) will be used so that all compare reports for a multistep job will be preserved. If you use a PDS, each member will contain a report for the corresponding source (or listing) member.
2. If the SAA postprocessor is going to be used to generate a change validation report and a prototype targeted summary control file, it is highly recommended that a sequential file be used as the coverage output comparison file.

Output Log Dsn

Enter the name of the analysis log to be generated by SAA. This should be a physical sequential (PS) data set.

Programming Language

Enter the programming language of your source or listing file. This allows SAA to identify comment lines, declaration statements, and reformatted lines, if you choose to filter these out of the comparison report.

The following values are accepted:

- ASM
- C
- C++
- COBOL
- PL/I
- LASM
- LCOB
- LPL/I

Note: When you compare listings, the comparison report will show the changes to the source code that was extracted from the listing files.

Select Line Audit Filters:

Select which filters you want use.

Comments

Enter `Y` to have the application disregard all comment lines. These lines will not be shown in the comparison report.
Enter `N` to have the application show all mismatched comment lines in the comparison report.

Note: One or more comments located on a line (record) that also contains code will not be filtered.

Declares

Enter `Y` to have the application disregard all declaration statements for the language specified. These lines will not be shown in the comparison report.
Enter `N` to have the application show all mismatched declaration statements in the comparison report.

Reformatted

Enter `Y` to have the application disregard all reformatted lines for the language specified. These lines will not be shown in the comparison report.
Enter `N` to have SAA show all reformatted lines in the comparison report.

Note: This filter only looks at one line (record) at a time (that is, reformatted language statements that span more than one line will not be filtered unless all reformatting is done within a line).

Select Comparison Columns

Enter the column positions in the source file where you want the comparison to begin and end.

Start Col Either enter a column position from 1–176 to indicate where you want the comparison to begin or leave the field empty.

Notes:

1. If you leave this field or the `End Col` field empty, SAA will compare all columns.
2. The `Start Col` value must be less than or equal to the `End Col`.
3. When comparing listings, enter the starting column position in the converted source code, not in the listing.

End Col Either enter a column position from 1–176 to indicate where you want the comparison to end or leave the field empty.

Notes:

1. If you leave this field or the `Start Col` field empty, SAA will compare all columns.
2. The `End Col` value must be greater than or equal to the `Start Col`.
3. When comparing listings, enter the ending column position in the converted source code, not in the listing.

Select Execution Option

Select whether to edit the JCL before submitting it.

Edit JCL Enter `Y` to generate the JCL and then edit the JCL in an ISPF edit session that starts after the JCL is generated. Enter `N` to generate the JCL and SUBMIT the job stream directly to the system for execution.

DBCS Support

Select whether DBCS support is enabled.

Enable Enter `N` if the source code does not contain DBCS characters. Enter `Y` if it does.

DBCS support for the control file statements is explained in Appendix C, “DBCS Support” on page 391.

Executing SAA in the Foreground

When you select option 2 on the `Execute Source Audit Assistant` panel, the `SAA Foreground Execution Parameters` panel, shown in Figure 114, is displayed.

```
---------------------- SAA Foreground Execution Parameters ----------------------
Option ===>

New Source Dsn: . . .  'project.newgroup.cobol(*)'

Old Source Dsn: . . .  'project.oldgroup.COBOL(*)'

Output Compare Dsn: .  'yourid.SAA.YYY.CMP'

Output Log Dsn: . . .  'yourid.SAA.YYY.LOG'

Programming Language:  COBOL   (ASM,C,C++,COBOL,PL/I,LASM,LCOB,or LPL/I)


Select Line Audit Filters (Y = apply filter, N = do not apply filter):
  Comments  Y (Y or N)   Declares  Y (Y or N)   Reformatted  Y (Y or N)

Select Comparison Columns:
  Start Col  1   (1-176)   End Col  72  (1-176)

DBCS support:
  Enable  N (Y or N)

Press END to terminate
```

Figure 114. SAA Foreground Execution Parameters Panel

The panel's fields are as follows:

New Source Dsn

Enter the name of the data set you want SAA to compare. If you are comparing a partitioned data set (PDS), include the member name or an asterisk (*) to include all members.

You can compare an original *source* data set (Old Source Dsn) with a modified *source* (New Source Dsn), or you can also compare *listing* data sets.

Old Source Dsn

Enter the name of the data set you want SAA to compare. If you are comparing a partitioned data set (PDS), include the member name or an asterisk (*) to include all members.

Output Compare Dsn

Enter the name of the comparison report to be generated by SAA. Foreground processing requires a physical sequential (PS) data set. You can preallocate this file or allow SAA to allocate it for you.

To preallocate the comparison file, use the following parameters:

- DSORG=PS
- SPACE=(TRK, (5,5))
- RECFM=FB
- LRECL=203
- BLKSIZE=27811

Note: Foreground processing automatically creates a sequential Output Compare File if the specified data set is not found at the time of execution.

Output Log Dsn

Enter the name of the analysis log to be generated by SAA. This should be a physical sequential (PS) data set. You can preallocate this file or allow SAA to allocate it for you.

To preallocate the log file, use the following parameters:

- DSORG=PS
- SPACE=(TRK, (5,5))
- RECFM=FBA
- LRECL=121
- BLKSIZE=27951

Note: Foreground processing automatically creates a sequential Output Log File if the specified data set is not found at the time of execution.

Programming Language

Enter the programming language of your source or listing data set. This allows SAA to identify comment lines, declaration statements, and reformatted lines, if you choose to filter these out of the comparison report.

The following values are accepted:

- ASM
- C
- C++
- COBOL
- PL/I
- LASM
- LCOB
- LPL/I

Note: When you compare listings, the comparison report will show the changes to the source code that was extracted from the listing data sets.

Select Line Audit Filters

Select which filters you want use.

Comments

Enter `Y` to have the application disregard all comment lines. These lines will not be shown in the comparison report.
Enter `N` to have the application show all mismatched comment lines in the comparison report.

Note: One or more comments located on a line (record) that also contains code will not be filtered.

Declares

Enter `Y` to have the application disregard all declaration statements for the language specified. These lines will not be shown in the comparison report.
Enter `N` to have the application show all mismatched declaration statements in the comparison report.

Reformatted

Enter `Y` to have the application disregard all reformatted lines for the language specified. These lines will not be shown in the comparison report.
Enter `N` to have SAA show all reformatted lines in the comparison report.

Note: This filter only looks at one line (record) at a time (that is, reformatted language statements that span more than one line will not be filtered unless all reformatting is done within a line).

Select Comparison Columns

Enter the column positions in the source file where you want the comparison to begin and end.

Start Col Either enter a column position from 1–176 to indicate where you want the comparison to begin or leave the field empty.

Notes:

1. If you leave this field or the `End Col` field empty, SAA will compare all columns.
2. The `Start Col` value must be less than or equal to the `End Col`.
3. When comparing listings, enter the starting column position in the converted source code, not in the listing.

End Col Either enter a column position from 1–176 to indicate where you want the comparison to end or leave the field empty.

Notes:

1. If you leave this field or the `Start Col` field empty, SAA will compare all columns.
2. The `End Col` value must be greater than or equal to the `Start Col`.
3. When comparing listings, enter the ending column position in the converted source code, not in the listing.

DBCS Support

Select whether DBCS support is enabled.

Enable Enter `N` if the source code does not contain DBCS characters. Enter `Y` if it does.

DBCS support for the control file statements is explained in Appendix C, “DBCS Support” on page 391.

Executing the SAA Postprocessor

If you select option `5` on the `Execute Source Audit Assistant` panel, the `SAA Postprocessor` panel, shown in Figure 115, is displayed. This panel allows you to invoke the SAA postprocessor, which can use a compare file generated by a previous SAA run and a list of *seed* variables that you are interested in monitoring to generate:

1. A change validation report, which shows changed lines that did not contain any seed variables and seed variables that did not appear in any changed lines. The change validation report allows you to ensure that:
   - Only planned changes to your source code were made
   - All seed variables were changed

   This report is optional.
2. A prototype CA target control file, which can be used with few or no changes as input to CA targeted summary. CA targeted summary can then produce a summary report on all variables found in the New Source Dsn's changed lines.

```
------------------------------ SAA Postprocessor -------------------------------
Command ===>


Input Data Sets:
SAA Compare Dsn  . . . 'yourid.SAMPLE.SAA.COBOLM.CMP'
Seed List Dsn  . . . . 'ATC.V1R4M0.SAMPLE.SAA.SEEDLIST(COBOLM)'

Output Data Sets:
Target Control Dsn . . 'yourid.SAMPLE.SAA.COBOLM.TARGCTL'
Chg Validation Rpt Dsn 'yourid.SAMPLE.SAA.COBOLM.REP'

DBCS support:
  Enable  N (Y or N)
```

Figure 115. SAA Postprocessor Panel

The panel's fields are as follows:

SAA Compare Dsn
: Name of a previously generated SAA output comparison data set, as specified in Figure 113 on page 290 or Figure 114 on page 293.

  Note: You can use a physical sequential (PS) data set or a partitioned data set (PDS) for the SAA Compare Dsn. You must include a member name if the comparison data set is a PDS.

Seed List Dsn
: Name of a data set containing a list of those variables that were to be changed and that you may be interested in monitoring if you run targeted summary.

  If no Seed List Dsn is specified, the change validation report is not generated.

Target Control Dsn
: Name of a prototype target control data set, which can be used as input to CA targeted summary to target on all variables found on changed *new* file records. (The COBOL or PL/I record may require modification to specify the correct listing DSName.)

Chg Validation Rpt Dsn
: Name you want the postprocessor to give to the change validation report it creates. This report lists the following information:
  - All of the changed lines that did not contain any seed variables.
  - Variables that did not appear in any of the changed lines.

  If no data set is specified for `Seed List Dsn`, any value entered in the `Chg Validation Rpt Dsn` field is not verified or used.

DBCS Support
: Select whether DBCS support is enabled.

  Enable Enter `N` if the source code does not contain DBCS characters. Enter `Y` if it does.

  DBCS support for the control file statements is explained in Appendix C, “DBCS Support” on page 391.

Press Enter to run the postprocessor. Upon completion, the change validation report data set is displayed for viewing (if you chose to create one).

You can also submit a batch job to run the SAA postprocessor. For an example of JCL that runs the SAA postprocessor, see the data set '*hi_lev_qual*.V1R5M0.SAMPLE.JCL(SAAPP)'.

For information about SAA change validation reports and about input and output file contents, see “Source Audit Assistant Postprocessor” on page 301.

Understanding the Comparison Report

SAA lists the differences between the two input files in a comparison report. Figure 116 shows a sample comparison report.

```
Source Audit Assistant V01.2             Date 12/02/1997  Time 11.07
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
         New File Name ->  ATC.V1R5M0.SAMPLE.SAA.NEW.COBOL(COBOLM)
         Old File Name ->  ATC.V1R5M0.SAMPLE.SAA.OLD.COBOL(COBOLM)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                       COBOL comments have been included
                       COBOL declares have been included
                       Reformatted lines have been included
                       The compare was from column 1 to column 176
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Line #  File  Contents
------  ----
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+----10---+----11---+----12

THE FOLLOWING LINE(S) HAVE BEEN DELETED

   312   Old               WHEN NUM-MINUTES > 0 AND <= 20
   313   Old                  COMPUTE INIT-COST = INIT-COST + (NUM-MINUTES * 8)

THE FOLLOWING LINE PAIR(S) HAVE BEEN REFORMATTED

   313   New                  COMPUTE INIT-COST = INIT-COST + ( NUM-MINUTES * 7 )
   315   Old                  COMPUTE INIT-COST = INIT-COST + (NUM-MINUTES * 7)


THE FOLLOWING LINE(S) HAVE BEEN INSERTED

   369   New             MOVE 3         TO COST.

THE FOLLOWING LINE PAIR(S) HAVE BEEN REPLACED

   401   New             VARYING SUB1 FROM 2 BY 1 UNTIL SUB1 = CALLS-MADE
   402   Old             VARYING SUB1 FROM 1 BY 1 UNTIL SUB1 = CALLS-MADE

   431   New             COMPUTE INIT-COST = INIT-COST + TOTAL-COST + 6.
   432   Old             COMPUTE INIT-COST = INIT-COST + TOTAL-COST + 5.

   460   New             COMPUTE INIT-COST = INIT-COST + TOTAL-COST + 20.
   461   Old             COMPUTE INIT-COST = INIT-COST + TOTAL-COST + 10.
```

Figure 116. Output Report Created by Comparing COBOL Source Files

Areas of the Comparison Report

At various intervals in the comparison report, SAA identifies the input data sets and lists any filters that you have selected. It also lists the range of columns that you compared.

The input data sets are called *New file* and *Old file* in the comparison report, corresponding to the values you entered for the file name fields on the `Execute Source Audit Assistant` panel.

© Copyright IBM Corp. 1997, 1999

When comparing listings, the comparison report shows the input listing files on the *New File Name* and *Old File Name* lines. SAA extracts the source code from these listings and creates two temporary source code input files. The comparison report shows changes in the source code that was extracted from the listings.

SAA identifies each difference as one of the following types: *insertion, deletion, replacement*, or *reformatted line.*

Source Audit Assistant Postprocessor

The SAA postprocessor allows you to verify the changes found in the SAA comparison report against your seed list in order to make sure that:

1. Only planned changes were made.
2. All seed variables were changed

In order to create an SAA change validation report, you need an SAA comparison report, which compares two COBOL or PL/I source files and a list of COBOL or PL/I variables (seeds) that you intended to change[44]. The SAA change validation report that is generated identifies:

1. All specified variables that did not appear in any new line in the SAA comparison report
2. All lines in the SAA comparison report that did not contain at least one of the specified variables

In addition, a skeleton CA targeted summary control file is generated, which selects all variables referenced in all mismatched new file lines. This file can be used as an input file for CA targeted summary. However, first you may need to modify the listing data set name to specify the correct listing data set name.

SAA Postprocessor Inputs

In order to create an SAA change validation report and a skeleton CA targeted summary control file, you need an SAA comparison report that compares two COBOL or PL/I source or listing files. The SAA comparison report is generated by running the SAA program against old and new COBOL or PL/I source or listing data sets as previously described.

In addition, a variable (seed) list file is required if you want to create a change validation report. This contains a list of COBOL or PL/I variables (seeds) that you intended to change. The variable (seed) list file is a sequential data set or a member of a partitioned data set and can have any DCB attributes. The format of this file is a free-form list of COBOL or PL/I variable names subject to the following restrictions:

1. Variable names cannot be continued from one line to the next.
2. When more than one variable name appears on a line, the variable names must be separated by one or more blanks.

DBCS support for the seed list file entries is explained in Appendix C, “DBCS Support” on page 391.

[44] “Change” in this context means that you want to change either the definition of the variable or references to the variable.

© Copyright IBM Corp. 1997, 1999

SAA Postprocessor Outputs

Two output data sets are created by the SAA postprocessor:

1. An SAA change validation report (optional)
2. A skeleton CA targeted summary control file

The SAA change validation report data set contains three sections. The first section simply identifies the old and new files used in the SAA comparison report, the date and time of the run, and so on. The second section of the report, titled "Input Seeds Not Found In Any Changed Lines," identifies all specified variables which were not found in any new file line in the SAA comparison report. The last section of the report, titled "New Lines Containing No Seed References," identifies new file lines in the SAA Comparison report that did not contain at least one of the specified variables.

Figure 117 shows a sample SAA change validation report. This report was generated from the COBOL source files indicated and from the variable list in ATC.V1R5M0.SAMPLE.SAA.SEEDLIST(COBOLM) which consists of a single line:

```
  INIT-COST  NOTHERE
```

```
 - - - - - - - - - - - - SAA Change Validation Report  - - - - - - - - - - -

Date:12/02/1997                        Time:11:08:06
SAA Report Dataset: 'ATC.V1R5M0.SAMPLE.SAA.COBOLM.CMP'
SAA Report Created: 12/02/1997 11.07
New Source Dataset: 'ATC.V1R5M0.SAMPLE.SAA.NEW.COBOL(COBOLM)'
Old Source Dataset: 'ATC.V1R5M0.SAMPLE.SAA.OLD.COBOL(COBOLM)'

 - - - - - - - - Input Seeds Not Found In Any Changed Lines  - - - - - - - -

 NOTHERE

 - - - - - - - - - New Lines Containing No Seed References - - - - - - - - -

 Line Contents

  369            MOVE 3         TO COST.
  401            VARYING SUB1 FROM 2 BY 1 UNTIL SUB1 = CALLS-MADE
```

Figure 117. Sample SAA Change Validation Report

The target control data set contains a skeleton targeted summary control file that can be used to create a CA targeted summary report for all variables found in mismatched new file records. However, before this data set can be used as a CA targeted summary control file, you must ensure that the listing data set name is correct. See "Targeted Summary Reports" on page 105 for more information on targeted summary.

The SAA postprocessor creates the listing data set name by replacing the low-level qualifier in the new file data set name with "LISTING." If this is not correct, you must modify the line to specify the correct listing data set name.

Reserved Data Set Names in Source Audit Assistant

As SAA executes in the foreground, it creates various data sets. The following is a list of these data sets. Ensure that you do not use these data set names for other purposes.

userid.TEMP.SAA.CMPOUT
userid.TEMP.SAA.CMP.COL
userid.TEMP.SAA.FCOMOUT
userid.TEMP.SAA.FDCLOUT
userid.TEMP.SAA.FINPOUT
userid.TEMP.SAA.FRNOOUT

userid.TEMP.SAA.NEW.CNTL
userid.TEMP.SAA.NEW.DATACT
userid.TEMP.SAA.NEW.DATAMOD
userid.TEMP.SAA.NEW.DATASW
userid.TEMP.SAA.NEW.DATASWU
userid.TEMP.SAA.NEW.DATASWUO

userid.TEMP.SAA.OLD.CNTL
userid.TEMP.SAA.OLD.DATACT
userid.TEMP.SAA.OLD.DATAMOD
userid.TEMP.SAA.OLD.DATASW
userid.TEMP.SAA.OLD.DATASWU
userid.TEMP.SAA.OLD.DATASWUO

userid.TEMP.SAA.NEW.LIST2SRC
userid.TEMP.SAA.NEW.LIST2SRC.*

userid.TEMP.SAA.OLD.LIST2SRC
userid.TEMP.SAA.OLD.LIST2SRC.*

When SAA executes in the background and uses compiler listings as input, it extracts source code from the listings and creates the following temporary data sets:

- userid.Dddd.Thhmmssu.NEW.LIST2SRC
- userid.Dddd.Thhmmssu.NEW.LIST2SRC.*
- userid.Dddd.Thhmmssu.OLD.LIST2SRC
- userid.Dddd.Thhmmssu.OLD.LIST2SRC.*

© Copyright IBM Corp. 1997, 1999

Appendixes

© Copyright IBM Corp. 1997, 1999

Appendix A. Problem Determination

This appendix describes abends that you may encounter during ATC installation, as well as error messages that you may receive during ATC operation.

Installation Abends

The following abends could occur during installation of the monitor SVCs. If an abend occurs, find the abend code in the following table, and then take the corrective action described.

Abend Code	Description	Corrective Action
444	Failure finding monitor program.	Verify that the monitor program (MONSVC) is installed in an authorized library in the load module search path for the installation job.
445	Cannot allocate ECSA (extended common service area) storage for monitor. The monitor program is installed in ECSA, but insufficient ECSA storage was available.	The amount of storage used by the monitor is listed in Appendix B, “ATC Requirements and Resources” on page 383. Verify with your system programmer that this much ECSA storage is available.
446	Failure to load monitor in ECSA.	Verify that the monitor program (MONSVC) is installed in an authorized library in the load module search path for the installation job.
447	Failure updating SVC table with new user SVC numbers.	Verify that the user SVC numbers passed to the install program (MONINSTS) are available.
448	Failure removing prior SVC token.	Contact ATC support.
449	Failure setting SVC token.	Contact ATC support.
450	Short SVC number supplied is not allowed. The short SVC number (the first number passed to MONINSTS) is not between C8 and FF (200 to 255 in decimal).	Pass a hexadecimal number between C8 and FF, without any surrounding quotes or any hexadecimal identifier, using the following command: `EXEC PGM=MONINSTS,PARM=(FE,FF)`
451	Long SVC number supplied is not allowed. The long SVC number (the second number passed to MONINSTS) is not between C8 and FF (200 to 255 in decimal).	Pass a hexadecimal number between C8 and FF, without any surrounding quotes or any hexadecimal identifier, using the following command: `EXEC PGM=MONINSTS,PARM=(FE,FF)`

© Copyright IBM Corp. 1997, 1999

Error Messages

The following error messages could occur during ATC operation. In this appendix, errors are grouped into categories and errors codes are arranged alphabetically.

If you receive an error message, find the error code prefix in the following list, go to the page specified, and find your message within that category of errors. Each error message explains why the error occurred, how your system responded to the error, and what action you should take to correct the error.

- COMBINE: Creating the JCL (ARCOxxxx). See page 309.
- SUMMARY: Creating the JCL (ARSUxxxx). See page 310.
- SETUP: Creating the JCL (ASETxxxx). See page 311.
- COMBINE: Executing the JCL (CMBxxxx). See page 312.
- COMMANDS: Executing User Commands (CMD5xxxx). See page 314.
- COMMON: Common Messages for CA/DA/UTA User Interface (COMNxxxx). See page 319.
- DEFAULTS: Defaults Processing (DFLTxxxx). See page 326.
- FILE WARP & DISTILLATION: File Conversion (FCVxxxx). See page 328.
- FILE WARP: Control File Reader (FWPxxxx). See page 332.
- COMMON: Additional Common Messages (KPDSxxxx). See page334 .
- CTLFILE: Control File Processing (N2Oxxxx). See page 335.
- TARGETED SUMMARY: Targeted Summary (OCUSxxxx). See page 340.
- VARIABLE REPORTS: Executing the JCL (PVRxxxx). See page 343.
- DISTILLATION: Read the Keylist (RKEYxxxx). See page 346.
- REPORT: Executing the JCL (RPT03xx). See page 347.
- SAA: Source Audit Assistant (SAACxxxx). See page 350.
- SAA POSTPROCESSOR: Source Audit Assistant Change Validation Report (SAAFxxxx). See page 353.
- SAA EXECUTION: Source Audit Assistant Execution (SAARxxxx). See page 355.
- SETUP: Executing the JCL (SP601xx). See page 359.
- SUMMARY: Executing the JCL (SUM04xx). See page 362.
- VAREAD: Executing the JCL (VAR0xxx). See page 364.
- EXECUTE: Buffer Monitor (WVAxxxx). See page 373.
- ZAPTXT: Executing the JCL (ZAP00yy or ZAP88xx). See page 375.

COMBINE: Creating the JCL (ARCOxxxx)

ARCO001E Quick Combine file is a PDS but no member name specified

Explanation: The file specified to contain the combined files is a partitioned data set but no member name was specified.

Combine Action: Combine terminates.

User Response: Specify a member name for the combined files or specify a sequential data set.

ARCO002E Combine input file format is incorrect, use panels to create file

Explanation: The format of at least one the files containing input to combine is not correct.

Combine Action: Combine terminates.

User Response: Use the ISPF panels to create the breakout files to be combined.

ARCO003E Only 1 input found, 2 files required

Explanation: Combine must have at least two breakout files to combine.

Combine Action: Combine terminates.

User Response: Specify at least two breakout files to be combined in the combine control file.

SUMMARY: Creating the JCL (ARSUxxxx)

ARSU001E Quick Summary found an invalid value for Internal/External

Explanation: An invalid value has been specified for the type of summary report to be generated.

Summary Action: Summary terminates.

User Response: Specify either an I for an Internal report or an E for an External report.

SETUP: Creating the JCL (ASETxxxx)

ASET011E	**Load and Object datasets 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.

Setup 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.

COMBINE: Executing the JCL (CMBxxxx)

CA may issue the following messages while executing the combine JCL:

CMB0101E CA: Error - Incorrect parameters supplied.

Explanation: A parameter was supplied to Combine, but it requires none.

Combine Action: Combine terminates.

User Response: Remove the parameter(s).

CMB0102E CA: Error opening input file: (dsn).

Explanation: Combine could not open the specified data set.

Combine Action: Combine terminates.

User Response: Ensure that the data set exists and is available.

CMB0103E CA: Error opening output file: (dsn).

Explanation: Combine could not open the specified data set.

Combine Action: Combine terminates.

User Response: Ensure that the data set is allocated and can be accessed.

CMB0501E CA: Error - insufficient storage to satisfy request.

Explanation: The CA 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 96 byte per PA and 32 bytes per logical file number).

Combine Action: Combine terminates.

User Response: Check the region size on your job card for the step, and increase it. CA obtains job card information from the site defaults file.

CMB0510E CA: Error - Input file 1 is not a BRKOUT or BRKTAB file.

Explanation: Input file 1 is not a BRKOUT or BRKTAB file.

Combine Action: Combine terminates.

User Response: Ensure that input file 1 is a valid BRKOUT or BRKTAB file.

CMB0511E CA: Error - Input file 2 is not a BRKOUT or BRKTAB file.

Explanation: Input file 2 is not a BRKOUT or BRKTAB file.

Combine Action: Combine terminates.

User Response: Ensure that input file 2 is a valid BRKOUT or BRKTAB file.

CMB0520E CA: Error - Input files don't 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).

Combine Action: Combine terminates.

User Response: Ensure that the file types (BRKOUT or BRKTAB) match.

COMMANDS: Executing User Commands (CMD5xxxx)

The following messages may be issued when commands are executed (all CA*xxxx* commands issued are from the command panel, directly from TSO, or while starting a monitor session [X*nnnnnnn* JCL]):

CMD5001E Invalid command

Explanation: An invalid command was issued.

Command Action: Ignored.

User Response: Since the commands are generated by REXX programs or created JCL, check that the REXX or JCL that issued the command has not been corrupted.

CMD5002E Error reading BRKTAB file

Explanation: A user tried to start a session, but there was an error reading the BRKTAB file of breakpoint data.

Command Action: Command not executed.

User Response: Check that the BRKTAB file (DD name BRKTAB in the execute JCL X*nnnnnn*) is accessible and is a valid BRKTAB file (starts with characters *BT8*).

CMD5003E Invalid BRKTAB file

Explanation: A user tried to start a session, but the BRKTAB file of breakpoint data is invalid.

Command Action: Command not executed.

User Response: Check that the BRKTAB file (DD name BRKTAB in the execute JCL X*nnnnnn*) is a valid BRKTAB file and was created by the V1R5M0 or later Setup program (starts with characters *BT8*).

CMD5004E Error opening BRKOUT file

Explanation: A user tried to stop a session, but the BRKOUT file of breakpoint data could not be opened.

Command Action: Command not executed.

User Response: Check that the BRKOUT file (DD name BRKOUT in the execute JCL X*nnnnnn*) exists and is accessible.

CMD5005E Error writing BRKOUT file

Explanation: A user tried to stop a session, but the BRKOUT file of breakpoint data could not be written.

Command Action: Command not executed.

User Response: Check that the BRKOUT file (DD name BRKOUT in the execute JCL X*nnnnnn*) exists and is accessible.

CMD5006E Error opening VARCTL file

Explanation: A user tried to start a session, but the VARCTL file of variable read data could not be opened.

Command Action: Command not executed.

User Response: Check that the VARCTL file (DD name VARCTL in the execute JCL X*nnnnnn*) exists and is accessible.

CMD5007E Invalid VARCTL file

Explanation: A user tried to start a session, but the VARCTL file of variable read data could not be opened.

Command Action: Command not executed.

User Response: Check that the VARCTL file (DD name VARCTL in the execute JCL X*nnnnnn*) is a valid VARCTL file produced by Setup and starts with the characters *VAR03*.

CMD5008E Error reading VARCTL file

Explanation: A user tried to start a session, but an error occurred reading the VARCTL file of variable read data.

Command Action: Command not executed.

User Response: Check that the VARCTL file (DD name VARCTL in the execute JCL X*nnnnnn*) is a valid VARCTL file produced by Setup and starts with the characters *VAR03*.

CMD5009E No ESQA space for BP table

Explanation: A user tried to start a session, but the monitor could not allocate sufficient space for the breakpoint (BP) table in ESQA.

Command Action: Command not executed.

User Response: For the storage requirements of the BP table, see Appendix B, “ATC Requirements and Resources” on page 383. Reduce the number of object modules being tested, or contact the system programmer to increase ESQA storage.

CMD5010E No ESQA space for PA table

Explanation: A user tried to start a session, but the monitor could not allocate sufficient space for the program area (PA) table in ESQA.

Command Action: Command not executed.

User Response: For the storage requirements of the PA table, see Appendix B, “ATC Requirements and Resources” on page 383. Reduce the number of object modules being tested, or contact the system programmer to increase ESQA storage.

CMD5011E No ESQA space for VA table

Explanation: A user tried to start a session, but the monitor could not allocate sufficient space for the variable read table in ESQA.

Command Action: Command not executed.

User Response: For the storage requirements of the breakpoint (BP) table, see Appendix B, “ATC Requirements and Resources” on page 383. Reduce the number of variable reads, or contact the system programmer to increase ESQA storage. Because the storage requirements of the variable (VA) table are small, this may be a symptom of an invalid VARCTL table.

CMD5015E Downlevel Monitor running - cannot execute command

Explanation: The installed monitor is at a previous level that is now incompatible.

Command Action: Command not executed.

User Response: Install the current monitor.

CMD5017E No storage in SQA for monitor tables

Explanation: During installation of the ATC monitor, the monitor could not obtain SQA space for its tables.

Command Action: The installation of the monitor is terminated.

User Response: See Appendix B, “ATC Requirements and Resources” on page 383 for the amount of SQA needed by the monitor during installation. Contact your systems programmer to determine the amount of SQA space available on your system.

CMD5023W Dup listing in ID: user_ID listing_name

Explanation: A monitor session was started that contains a BRKTAB for a listing/object module that is being monitored by another session.

Command Action: The session is started.

User Response: Note that if two or more sessions share a BRKTAB for a listing/object module, the first session started will get all coverage for the listing/object module (sessions started after the first will have incomplete coverage for any shared listings/object modules).

CMD5102E The requested session id sess_id was not found.

Explanation: A command that requires a session ID was issued, but the session ID (sess_id) was not found.

Command Action: Command not executed.

User Response: Check that the sess_id is valid. The session ID is usually your TSO session ID.

CMD5103E The requested session id is not active. Command rejected.

Explanation: A command that requires a session ID was issued, but the session ID was not active.

Command Action: Command rejected.

User Response: Check that the sess_id is active by using the CASESSN command.

CMD5104E The requested PA# pa_num was not found.

Explanation: A command that requires a program area (PA) number was issued, but the PA number (pa_num) was not found.

Command Action: Command not executed.

User Response: Check that the pa_num is valid. The CALIST command lists the number of PAs for each listing in a session.

CMD5105E No free sessions in session table - session not started

Explanation: A user attempted to start a new session, but there are new free sessions in the monitor session table.

Command Action: Command not executed.

User Response: Use the CASESSN to display active sessions. Stop or quit sessions that are not needed. A maximum of 32 sessions can be active.

CMD5106E Session already active, cannot be started again

Explanation: A user attempted to start a new session, but a session is already started with that session ID.

Command Action: Command not executed.

User Response: Use the CASESSN 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.

CMD5200E No previous copy of the monitor exists - cannot do commands to it

Explanation: The monitor is not installed, or has been corrupted.

Command Action: Command not executed.

User Response: Reinstall the monitor.

CMD5202E No session table exists

Explanation: The session table does not exist or has been corrupted.

Command Action: Command not executed.

User Response: Reinstall the monitor.

CMD5999E Error number err_num occurred. Contact ATC support.

Explanation: An undocumented error occurred.

Command Action: The operation terminates.

User Response: Contact ATC support with the error number and explain the circumstances that caused it.

COMMON: Common Messages for CA/DA/UTA User Interface (COMNxxxx)

These messages are common across most areas of CA/DA/UTA and can be issued by many of the the functions.

COMN001I *function* **is starting**

Explanation: The specified function is starting.

System Action: The function begins execution

User Response: None.

COMN002I *function* **is verifying your parameters**

Explanation: The specified function is verifying the parameters are correct.

System Action: The function continues.

User Response: None.

COMN003I *function* **is reading the CACTL file**

Explanation: The specified function is reading the CA/DA/UTA control file.

System Action: The function continues.

User Response: None.

COMN004I *function* **is creating the JCL** *member name*

Explanation: The specified function is creating the JCL for this function in the user's JCL library.

System Action: The function continues.

User Response: None.

COMN005R *function* **is done**

Explanation: The specified function has completed.

System Action: The function terminates.

User Response: None.

COMN006E *function* **JCL library does not exist:** *library dsn*

Explanation: The specified JCL library does not exist.

System Action: The function terminates.

User Response: Specify an existing JCL library and rerun the function.

COMN007E *function* **found this invalid report parameter:** *parameter*

Explanation: The specified parameter is invalid for this function.

System Action: The function terminates.

User Response: Correct the parameter and rerun the function.

COMN008I *data*

Explanation: This message displays information in support of the previous message.

System Action: See the corresponding message.

User Response: See the corresponding message.

COMN011I **Data** *data element* **saved**

Explanation: The specified data has been saved.

System Action: The function continues.

User Response: None.

COMN012E **Dataset** *dataset name* **not found**

Explanation: The specified data set was not found.

System Action: The function terminates.

User Response: Specify the name of an existing data set.

COMN013S **Unknown option**

Explanation: An unknown option was encountered when trying to execute a function.

System Action: The function terminates.

User Response: Contact ATC support.

COMN014I **JCL submitted**

Explanation: The JCL to execute the requested function has been submitted.

System Action: A batch job is submitted and the function terminates.

User Response: None.

COMN015I **Dataset** *dsn* **reset**

Explanation: The specified control data set has been reset to the default values.

System Action: The function terminates.

User Response: None.

COMN016E Copy error occurred trying to reset dataset *dsn*

Explanation: When trying to reset a control data set from the default template, an error occurred either reading the default template file (FORMS) or writing to the user control file.

System Action: The function terminates.

User Response: Ensure that the default template file (FORMS) is available on the system and that the user control file that is to be reset can be written to.

COMN018E *option* **is not a valid option'"**

Explanation: The specified option is invalid for this function.

System Action: The function terminates.

User Response: Correct the option and retry the function.

COMN019E Error *rc* **allocating** *dsn*

Explanation: An error occurred when trying to allocate the specified data set. The allocation return code is indicated by 'rc'

System Action: The function terminates.

User Response: Determine the allocation error and retry the function. Ensure that data sets required for this function exist.

COMN020I *data*

Explanation: This message displays information in support of the previous message.

System Action: See the corresponding message.

User Response: See the corresponding message.

COMN021E No output dataset specified

Explanation: The function requires the name of an output data set and none was specified.

System Action: The function terminates.

User Response: Specify the name of a data set to contain output from the function.

COMN022E Error processing *input]output* **dataset** *dsn* **:** *message*

Explanation: An error occurred processing the data set specified by 'dsn'. Further information about the error is contained in 'message'. For example, 'message' may indicate a data set or member not found.

System Action: The function terminates.

User Response: Correct the specified error and retry the function.

COMN023E Members specified but input not PDS

Explanation: A member has been specified for a data set that is not a partitioned data set.

System Action: The function terminates.

User Response: Do not specify a member name or specify the correct data set.

COMN024E Invalid operand *operand*

Explanation: An invalid operand has been specified.

System Action: The function terminates.

User Response: Correct the error and retry the function.

COMN025E 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.

System Action: The function terminates.

User Response: Specify a partitioned data set as the output data set.

COMN026E 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.

System Action: The function terminates.

User Response: Specify a sequential data set as the output data set or specify a member name.

COMN027E Error *rc* **from** *function*

Explanation: The specified function returned the indicated return code.

System Action: The function terminates.

User Response: Contact ATC support.

COMN028E ISPF not active

Explanation: This function requires ISPF to execute.

System Action: The function terminates.

User Response: Start ISPF and retry the function.

COMN029E Operand *operand* **of** *function* **is** *message*

Explanation: The indicated operand for this function is incorrect. Further information is contained in 'message'.

System Action: The function terminates.

User Response: Correct the operand and retry the function.

COMN030E *operation* **is not currently supported**

Explanation: The indicated operation for this function is not supported.

System Action: The function terminates.

User Response: Specify a supported operation.

COMN031E Error *rc* **deleting** *dsn*

Explanation: An error occurred when trying to delete the specified data set. *rc* contains the return code from delete.

System Action: The function terminates.

User Response: Correct the error indicated by the return code and retry.

COMN032E Error *rc* **writing** *dsn*

Explanation: An error occurred when trying to write to the specified data set. 'rc' contains the return code from EXECIO write.

System Action: The function terminates.

User Response: Correct the error indicated by the return code and retry.

COMN033E JCL submit error *rc*

Explanation: The TSO submit failed with rc=rc.

System Action: The function terminates.

User Response: None.

COMN034I Performing File Tailoring step jcldsn

Explanation: File Tailoring is being performed for step *step* on data set *jcldsn.*

System Action: The function continues.

User Response: None.

COMN035E FTOpen error rc=rc

Explanation: The FTOpen failed with rc=rc.

System Action: The function terminates.

User Response: None.

COMN036E FTIncl error rc=rc

Explanation: The FTIncl failed with rc=rc.

System Action: The function terminates.

User Response: None.

COMN037E FTClose error rc=rc

Explanation: The FTClose failed with rc=rc.

System Action: The function terminates.

User Response: None.

COMN038E Unable to alloc ddname dataset dsname

Explanation: The TSO alloc for ddname and dsname failed.

System Action: The function terminates.

User Response: None.

COMN039E Unable to read ddname dataset dsname

Explanation: Unable to read ddname *dsname.*

System Action: The function terminates.

User Response: None.

COMN040E Unable to write ddname dataset dsname

Explanation: Unable to write ddname *dsname.*

System Action: The function terminates.

User Response: None.

COMN041E DD not found ddname

Explanation: DD ddname not found allocated.

System Action: The function terminates.

User Response: None.

COMN042E ISPExec Error command rc=rc

Explanation: ISPExec Command *command* failed with rc=rc.

System Action: The function terminates.

User Response: None.

COMN043E Member in use member dsname

Explanation: Member *member* of data set *dsname* is already in use.

System Action: The function terminates.

User Response: None.

COMN044E Member not found member dsname

Explanation: Member *member* of data set *dsname* could not be found.

System Action: The function terminates.

User Response: None.

COMN045E DA and UTA may not be enabled together

Explanation: The Enable DA and Enable UTA process options on the `Create JCL for Setup` panel or the `Create JCL to Start the Monitor` panel cannot be set to YES at the same time.

System Action: The function terminates.

User Response: Enable only one (or neither) of these options.

COMN046E Data set *dataset 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.

System Action: The function terminates.

User Response: Reallocate *data set name* with a DCB characteristic of *valid value*.

DEFAULTS: Defaults Processing (DFLTxxxx)

DFLT001E Unable to allocate system defaults dataset *dsn*

Explanation: Could not allocate the ATC defaults data set.

Action: Defaults terminates.

User Response: Ensure the defaults file is available.

DFLT002E Unable to read system defaults dataset *dsn*

Explanation: The ATC defaults data set could not be read.

Action: Defaults terminates.

User Response: Ensure the data set is a proper ATC defaults data set and is accessible.

DFLT003E Unable to alloc import dataset *dsn*

Explanation: The import defaults data set could not be allocated.

Action: Defaults terminates.

User Response: Ensure the import defaults data set is accessible and the name is correct.

DFLT004E Unable to read import dataset *dsn*

Explanation: The import defaults data set could not be read.

Action: Defaults terminates.

User Response: Ensure the import defaults data set is accessible and is the proper format.

DFLT005E Unable to alloc export dataset *dsn*

Explanation: The export data set to contain the ATC defaults could not be allocated.

Action: Defaults terminates.

User Response: Ensure the export defaults data set is accessible and the name is correct.

DFLT006E Unable to write export dataset *dsn*

Explanation: The export defaults data set could not be written.

Action: Defaults terminates.

User Response: Ensure the export defaults data set is accessible and is the proper format.

DFLT007I ATC ISPF variables gotten

Explanation: The ISPF variables required to run ATC have been read.

Action: Defaults continues.

User Response: None.

DFLT008I ATC ISPF variables saved

Explanation: The ISPF variables have been saved.

Action: Defaults continues.

User Response: None.

DFLT009I ATC ISPF variable edit canceled

Explanation: The ISPF variables edit has been cancelled

Action: Defaults terminates.

User Response: None.

DFLT010I ATC ISPF variables reset from system defaults

Explanation: The ISPF variables have been reset to the system defaults.

Action: Defaults terminates.

User Response: None.

DFLT011I ATC ISPF variables imported from user dataset

Explanation: The ISPF variables have been imported from your import data set.

Action: Defaults terminates.

User Response: None.

DFLT012I ATC ISPF variables exported to user dataset

Explanation: The ISPF variables have been exported to your export data set.

Action: Defaults terminates.

User Response: None.

DFLT013E Unknown command

Explanation: An unknown command was encountered when trying to execute defaults.

Action: Defaults terminates.

User Response: Contact ATC support.

FILE WARP & DISTILLATION: File Conversion (FCVxxxx)

These messages are for errors in the distillation/file warp program. The error number corresponds to the return code from the program (PROCSTEP FWARP or DISTILL in the JCL output). The messages are printed in the SYSPRINT DD statement of the FWARP or DISTILL step in the JCL output.

FCV108E Key positions and/or length not specified

Explanation: The key position and/or length was not supplied on the create distillation JCL screen.

Action: Distillation is not done.

User Response: Supply the values on the screen and recreate the JCL.

FCV112E Unable to OPEN KEYFILE DCB

Explanation: The key file cannot be opened.

Action: Distillation is not done.

User Response: Verify that a key file of keys that caused new coverage was created. Usually the key file will be in a PDS with the last qualifier being VARDATA. If it is a sequential data set, the last qualifier is usually VARDATA. Look at the distillation JCL (Dxxxxxx) DISTILL step, KEYFILE DDNAME for the data set name used for the key file.

FCV116E Illegal function requested

Explanation: The option supplied to the file conversion program was illegal.

Action: Distillation/file warp is not done.

User Response: The option is supplied automatically within the ATRUNKEY REXX program, so you should never see this error. Contact ATC support.

FCV120E RECFM=VS is not currently supported

Explanation: You specified an input data set with VS record format.

Action: Distillation/file warp is not done.

User Response: VS record formats are not supported.

FCV124E Unable to OPEN MASTER(INPUT) DCB

Explanation: The input data set could not be opened.

Action: Distillation/file warp is not done.

User Response: Verify that you specified the correct input data set.

FCV128E Unable to OPEN NEWMASTER(OUTPUT) DCB

Explanation: The output data set could not be opened.

Action: Distillation/file warp is not done.

User Response: Verify that you specified correct qualifiers for the output data set.

FCV132E Specified key position lies outside of KeyFile record

Explanation: The key position supplied on the create distillation JCL screen is greater than the size of the record.

Action: Distillation is not done.

User Response: Correct the key position on the panel and recreate the JCL.

FCV136E DSORG of MASTER(INPUT) is not supported

Explanation: The organization of the input data set is not supported. Supported organizations are: PS, PSU, PO, POU, or VSAM.

Action: Distillation/file warp is not done.

User Response: Specify a correct input data set.

FCV140E Error from Dynamic Allocation. Error/Info Code: *xxxx*

Explanation: An error obtaining dynamic storage occurred.

Action: Distillation/file warp is not done.

User Response: Use the error code to obtain further information. Verify your parameters on the create distillation JCL screen.

FCV144E Specified key position lies outside of Master record

Explanation: The key position supplied on the create distillation JCL screen is greater than the size of the record.

Action: Distillation is not done.

User Response: Correct the key position on the panel and recreate the JCL.

FCV148E I/O error reading VSAM Master(INPUT). Feedback: *xxxx*

Explanation: An error occurred reading the input data set.

Action: Distillation/file warp is not done.

User Response: Check that the input data set is valid. Feedback contains the error returned from the I/O routine.

FCV152E I/O error writing VSAM Master(OUTPUT). Feedback: *xxxx*

Explanation: An error occurred writing the output data set.

Action: Distillation/file warp is not done.

User Response: Check that the output data set is valid. "Feedback" contains the error returned from the I/O routine.

FCV156E *xxxx* **is not a valid decimal parameter**

Explanation: A number passed to the distillation program (key position, length) is not valid.

Action: Distillation warp is not done.

User Response: Correct the number on the create distillation JCL screen.

FCV160E Unable to OPEN Warp File or file empty.

Explanation: The intermediate warp control file created by the warp control file conversion step could not be opened.

Action: File warp is not done.

User Response: Verify that the first step (warp control file conversion) executed successfully. Verify that your warp control file is correct.

FCV164E Warping signed, but value in file is unsigned offset: *xxxx*

Explanation: You specified that the field to warp was a signed number in the warp control file, but the field in the file contains an unsigned number.

Action: File warp is not done.

User Response: Correct the warp control file.

FCV168E Warping unsigned, but value in file is signed offset: *xxxx*

Explanation: You specified that the field to warp was an unsigned number in the warp control file, but the field in the file contains an signed number.

Action: File warp is not done.

User Response: Correct the warp control file.

FCV172E No space for warp array

Explanation: The dynamic allocation of storage for the warp action array failed.

Action: File warp is not done.

User Response: Check that the warp control file is valid, and that the warp file conversion program ran successfully.

FCV176E Unable to OPEN MASTER(INPUT) ACB. Error Code: *xxxx*

Explanation: The input data set could not be opened.

Action: Distillation/file warp is not done.

User Response: Verify that you specified the correct input data set.

FCV180E Unable to OPEN NEWMASTR(OUTPUT) ACB. Error Code: *xxxx*

Explanation: The output data set could not be opened.

Action: Distillation/file warp is not done.

User Response: Verify that you specified correct qualifiers for the output data set.

FILE WARP: Control File Reader (FWPxxxx)

These messages are for errors reported by the program that reads the file warp control file and checks it for syntax. The error number corresponds to the return code from the program (PROCSTEP CVTCTL in the JCL output). The messages are printed in the SYSTSPRT DD statement of the CVTCTL step in the JCL output.

FWP212E Error *xxxx* opening file warp dataset dsn

Explanation: The file warp control file cannot be opened.

Action: File warp is not done.

User Response: Verify the name of the file warp control file that you entered on the create file warp JCL screen.

FWP220E Return Code *xxxx* from ALLOCATE allocating dsn

Explanation: The program could not allocate the intermediate file of warp statements to be passed to the file warp program.

Action: File warp is not done.

User Response: Check that the file warp JCL has not been corrupted. This is an intermediate temporary file passed between job steps.

FWP224E Return Code *xxxx* from EXECIO reading dsn

Explanation: An I/O failure occurred reading the file warp control file.

Action: File warp is not done.

User Response: Verify that the file warp control file is readable.

FWP226E Double Byte Character Set error on record *xxxx*. Invalid DBCS or mixed string. Check DBCS name(s) on line *xxxx* of the control file

Explanation: There was a control file error with DBCS names.

Action: File warp is not done.

User Response: Check the DBCS names used for the DBCS control characters.

FWP228E *xxxx* is not a valid option for the *yyyy* statement

Explanation: An incorrect use of options for a statement was detected.

Action: File warp is not done.

User Response: Check the indicated statement for correct syntax.

FWP232E Required operand omitted from: *xxxx*

Explanation: A required warp operand was omitted (for example, warp value).

Action: File warp is not done.

User Response: Check the indicated statement for correct syntax.

FWP236E Return Code *xxxx* **from EXECIO writing** *yyyy*

Explanation: An I/O error occurred writing to the intermediate file.

Action: File warp is not done.

User Response: Check that the created JCL has not been corrupted. Recreate it if necessary.

FWP240E dsn file is not allocated

Explanation: The intermediate output file is not allocated.

Action: File warp is not done.

User Response: Check that the created JCL has not been corrupted. Recreate it if necessary.

FWP244E Return Code *xxxx* **from ALLOCATE allocating dsn**

Explanation: The intermediate output file could not be allocated.

Action: File warp is not done.

User Response: Check that the created JCL has not been corrupted. Recreate it if necessary.

FWP260E Record label mismatch detected at statement: *xxxx*

Explanation: A statement used to define a record label has incorrect syntax.

Action: File warp is not done.

User Response: Check the indicated statement for correct syntax.

FWP264E *xxxx* **is not a valid operand for** *yyyy*

Explanation: A statement has incorrect syntax.

Action: File warp is not done.

User Response: Check the indicated statement for correct syntax.

FWP268E Warp control file has no warps

Explanation: The warp control file contained no warp actions.

Action: File warp is not done.

User Response: Check that you used a valid warp control file.

COMMON: Additional Common Messages (KPDSxxxx)

KPDS001E Dataset *dsn* has an invalid 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.

CTLFILE: Control File Processing (N2Oxxxx)

N2O012E **Error** *xxxxxxxx* **for dataset** *dddddddd***.**

Explanation: The specified message (*xxxxxxxx*) was returned from the SYSDSN command indicating that the specified data set (*dddddddd*) could not be found or could not be processed.

Processor Action: Processing terminates.

User Response: Correct the data set name specification or make sure that the data set can be accessed.

N2O016E *oooooooo* **is not a valid option.**

Explanation: The specified option was found on the command invocation but is not a valid option recognized by this command.

Processor Action: Processing terminates.

User Response: Remove or correct the specified option.

N2O020E **Return Code** *rr* **from ALLOCATE allocating** *dddddddd***.**

Explanation: Return Code *rr* was returned from the ALLOCATE command attempting to allocate the specified input data set.

Processor Action: Processing terminates.

User Response: Correct the data set name specification or ensure that the data set can be allocated.

N2O024E **Return Code** *rr* **from EXECIO reading** *dddddddd***.**

Explanation: Return Code *rr* was returned from the EXECIO command attempting to read the specified data set.

Processor Action: Processing terminates.

User Response: Correct the indicated error or correct the data set name specification.

N2O026E **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 an invalid DBCS or mixed string. This statement was found on line number *num* in the control file.

Processor Action: Processing terminates.

User Response: Correct the DBCS usage in the statement indicated.

N2O028E *oooooooo* **is not a valid option for the** *ssss* **statement.**

Explanation: A keyword (*oooooooo*) was specified for the *ssss* statement which is not valid on that statement.

Processor Action: Processing terminates. The statement in error is shown in the second line of the error message.

User Response: Correct the specified option.

N2O029E **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 only be specified when the corresponding data set name is specified.

Processor Action: Processing terminates. The statement in error is shown in the second line of the error message.

User Response: Correct the specified option.

N2O030E *oooooooo* **is not a valid option for the** *ssss* **statement.**

Explanation: A keyword (*oooooooo*) was specified for the *ssss* statement which is not valid on that statement.

Processor Action: Processing terminates. The statement in error is shown in the second line of the error message.

User Response: Correct the specified option.

N2O031E **The operand of the COBOL or PL/I operand does not point to a statement of that type.**

Explanation: The SCOPE statement is coded with a COBOL or PL/I operand whose operand does not point to a matching COBOL or PL/I statement.

Processor Action: Processing terminates. The statement in error is shown in the second line of the error message.

User Response: Correct the control statement.

N2O032E **Required operand omitted from:**

Explanation: A required operand was omitted from the statement shown in the second line of the message.

Processor Action: Processing terminates. The statement in error is shown in the second line of the error message.

User Response: Correct the statement by adding all required operands.

N2O033E **LISTDSN=*dddddddd* contains an asterisk specification. However LISTMEMBER= was not specified.**

Explanation: When the data set name *dddddddd* contains an asterisk, **LISTMEMBER=** must be specified to replace the asterisk.

Processor 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.

N2O034E *////* **is a duplicate label on the following statement. It is ignored.**

Explanation: The specified label was previously found on a statement of the same type.

Processor 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.

N2O035E *////* **is not a previously defined label. The following statement is ignored.**

Explanation: The statement shown in the second line of the message referenced a label which was not previously defined.

Processor Action: Processing terminates. The statement in error is shown in the second line of the error message.

User Response: Correct the label reference.

N2O036E **Return Code *rr* from EXECIO writing *dddddddd*.**

Explanation: Return Code *rr* was returned from the EXECIO command attempting to write the specified data set.

Processor Action: Processing terminates.

User Response: Correct the indicated error or correct the data set name specification. *

N2O037E **Invalid NAME= operand in statement:**

Explanation: The statement shown in the second line of the message contains an invalid operand of the **NAME=**.

Processor Action: Processing terminates. The statement in error is shown in the second line of the error message.

User Response: Correct the variable name specification.

N2O038E A SCOPE cannot be defined in an ASM program.

Explanation: The SCOPE control statement is not supported for assembler programs.

Processor Action: Processing terminates. The statement in error is shown in the second line of the error message.

User Response: Remove the indicated control statement.

N2O039E The SCOPE referenced by the following statement must be COBOL.

Explanation: The indicated control statement is only supported for COBOL programs.

Processor Action: Processing terminates. The statement in error is shown in the second line of the error message.

User Response: Remove the indicated control statement.

N2O040E *ffffffff* **file is not allocated.**

Explanation: The specified DDNAME was not previously allocated.

Processor Action: Processing terminates.

User Response: Correct the DDNAME specification or ensure that the file is allocated.

N2O044E Return Code *rr* **from ALLOCATE allocating** *dddddddd***.**

Explanation: Return Code *rr* was returned from the ALLOCATE command attempting to allocate the specified output data set.

Processor Action: Processing terminates.

User Response: Correct the data set name specification or ensure that the data set can be allocated.

N2O048E 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.

Processor Action: Processing terminates.

User Response: Correct the control statement.

N2O052E Statement *ssssssss* **not recognized.**

Explanation: The indicated keyword is not a valid control statement type.

Processor Action: Processing terminates. The statement in error is shown in the second line of the error message.

User Response: Correct the control statement.

N2O056E *oooooooo* **is not a valid operand for the** *kkkkkkkk* **keyword.**

Explanation: The indicated operand is not valid for the indicated keyword.

Processor Action: Processing terminates. The statement in error is shown in the second line of the error message.

User Response: Correct the control statement.

N2O060E **Load and Object datasets 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.

Processor 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.

TARGETED SUMMARY: Targeted Summary (OCUSxxxx)

OCUS001E EXECIO read of *dsn* failed, RC= *rc*

Explanation: An error occurred while reading *dsn*. *rc* contains the return code from EXECIO.

Action: Target file creation terminates.

User Response: Correct the error indicated by the return code and retry.

OCUS005E EXECIO write of output dataset failed, RC = *rc*

Explanation: A failure occurred while writing to the output data set. *rc* contains the return code from the write.

Action: Target file creation terminates.

User Response: Correct the error indicated by the return code and retry.

OCUS006E Invalid statement in control file:

Explanation: An invalid statement was discovered in the targeted coverage control file. An accompanying message indicates the invalid statement.

Action: Target file creation terminates.

User Response: Correct the control statement and retry.

OCUS008E *procedure name* is not a valid procedure name in the specified report file

Explanation: The indicated procedure does not exist in the listing file.

Action: Target file creation terminates.

User Response: Specify the proper procedure name or listing name.

OCUS009E * is not a valid listing name when a variable name is specified

Explanation: You must specify a listing name if a variable name is specified.

Action: Target file creation terminates.

User Response: Specify the name of the listing where the variable is located.

OCUS010E Invalid control statement:

Explanation: An invalid statement was discovered in the targeted coverage control file. An accompanying message indicates the load module, procedure, and so on containing the invalid statement.

Action: Target file creation terminates.

User Response: Correct the control statement and retry.

OCUS017I Reading target control dataset.

Explanation: The target control data set is being read and processed.

Action: Processing continues.

User Response: None.

OCUS018I Processing listing *dsn*.

Explanation: The listing data set *dsn* is being read and processed.

Action: Processing continues.

User Response: None.

OCUS019I Generating Targeted Summary Report

Explanation: The targeted summary report is being generated.

Action: Processing continues.

User Response: None.

OCUS020E *DDname* cannot be allocated. *dsn* may not exist.

Explanation: The specified *dsn* could not be allocated to *DDname*.

Action: Processing terminates.

User Response: Ensure that the specified data set is accessible.

OCUS021E *DDname* cannot be allocated to *dsn*.

Explanation: The specified *dsn* could not be allocated to *DDname*.

Action: Processing terminates.

User Response: Ensure that the specified data set is accessible.

OCUS022E *DDname* cannot be allocated.

Explanation: The specified *DDname* could not be allocated.

Action: Processing terminates.

User Response: Contact your system programmer.

OCUS023E Error detected in Summary. RC = *rc*

Explanation: The summary report routine returned RC = *rc*.

Action: Processing terminates.

User Response: See the associated message(s) for more information.

OCUS024W Function messages issued. RC = *rc*

Explanation: The function routine has issued messages and returned RC = *rc*.

Action: See the associated messages.

User Response: See the associated message(s) for more information. This message is issued since the associated messages appear in a separate SYSOUT data set.

OCUS025W No statements targeted.

Explanation: There were no statements targeted.

Action: Processing terminates.

User Response: Ensure that the control data set contains the appropriate TARGETVAR and/or TARGETSTMT entries.

OCUS026E EXECIO write of *dsn* dataset failed, RC = *rc*.

Explanation: A failure occurred while writing to the output data set *dsn*. *rc* contains the return code from the write.

Action: Processing terminates.

User Response: Correct the error indicated by the return code and retry.

OCUS027E DD name *dd1* found when DD *dd2* was expected.

Explanation: The DD name *dd1* was found when *dd2* was expected.

Action: Processing terminates.

User Response: Ensure that all required parameters were provided with the correct DD names and in the correct order.

VARIABLE REPORTS: Executing the JCL (PVRxxxx)

PVR020E Variable Report: Missing Parameter(s)

Explanation: No parameter for the VARDATA output file was specified.

Report Action: Variable report terminates.

User Response: Verify that the VARDATA parameter in the report JCL exists.

PVR022E Variable Report: Variable Data List File Parameter is Invalid.

Explanation: The parameter for the VARDATA file is incorrect.

Report Action: Variable Report terminates.

User Response: Verify that the VARDATA parameter in the report JCL is valid. Refer to the *User's Guide* for a list of the valid parameters for the VARDATA Variable Report.

PVR0100E Variable Report: Debug Table File Open Failure

Explanation: The input debug table file (DDNAME DBGTAB) could not be opened.

Report Action: Variable Report terminates.

User Response: Verify that the name supplied in the Variable Report JCL for the DBGTAB file is valid and the file exists.

PVR0101E Variable Report: Variable ID List File Open Failure

Explanation: The output variable ID list file (DDNAME VARID) could not be opened.

Report Action: Variable Report terminates.

User Response: Verify that the name supplied in the report JCL for the VARID file is valid and there are no JCL errors in its definition.

PVR0102E Variable Report: Variable Table File Open Failure

Explanation: The input variable data file (DDNAME VARTAB) could not be opened.

Report Action: Variable Report terminates.

User Response: Verify that the name supplied in the report JCL for the VARTAB file is valid and the file exists.

PVR0103E Variable Report: Variable Data List File Open Failure

Explanation: The output variable data list file (DDNAME VARDATA) could not be opened.

Report Action: Variable Report terminates.

User Response: Verify that the name supplied in the report JCL for the VARDATA file is valid and there are no JCL errors in its definition.

PVR0104E Variable Report: Debug Table File is invalid

Explanation: The input debug table file is invalid (not a DBGTAB file or at the wrong release level).

Report Action: Variable Report terminates.

User Response: Verify that the name supplied in the report JCL for the DBGTAB file is valid and at the correct release level for the release you have installed (was not created by an earlier UTA release that is not compatible with this release). Valid DBGTAB files have *VAR01* as the first five characters of the file.

PVR0105E Variable Report: Array Bounds for Reading DBGTAB Records Has Been Exceeded.

Explanation: The number of DBGTAB records has exceeded the 10,000 array index limit.

Report Action: Variable Report Terminates.

User Response: Decrease the number of DBGTAB records per DBGTAB file to 10,000 or fewer or contact ATC support.

PVR0106E Variable Report: Variable Table File is invalid

Explanation: The input variable data file is invalid (not a VARTAB file or at the wrong release level).

Report Action: Variable Report terminates.

User Response: Verify that the name supplied in the report JCL for the VARTAB file is valid and at the correct release level for the release you have installed (was not created by an earlier UTA release that is not compatible with this release). A valid VARTAB file has *VTAB7* as the first five characters of the file.

PVR0200E Variable Report: Failure in Allocating Storage for Debug Table Header Section.

Explanation: The allocation of storage for the debug table header failed.

Report Action: Variable Report terminates.

User Response: Check your system for storage allocation problems and check that the DBGTAB file is valid.

PVR0201E Variable Report: Failure in Allocating Storage for Debug Table Character Data Section

Explanation: The allocation of storage for the debug table character data failed.

Report Action: Variable Report terminates.

User Response: Check your system for storage allocation problems and check that the DBGTAB file is valid.

PVR0202E Variable Report: Failure in Allocating Storage for Debug Table Statement Records Section

Explanation: The allocation of storage for the debug table statement records failed.

Report Action: Variable Report terminates.

User Response: Check your system for storage allocation problems and check that the DBGTAB file is valid.

PVR0203E Variable Report: Failure in Allocating Storage for Debug Table Variable Records Section

Explanation: The allocation of storage for the debug table variable records failed.

Report Action: Variable Report terminates.

User Response: Check your system for storage allocation problems and check that the DBGTAB file is valid.

PVR0204E Variable Report: Failure in Allocating Storage for Variable Table Records Section

Explanation: The allocation of storage for the variable table records failed.

Report Action: Variable Report terminates.

User Response: Check your system for storage allocation problems and check that the VARTAB file is valid.

PVR0300W Variable Report: Buffer Overflow—A Loss of Data May Have Occurred.

Explanation: Data was read from the input file faster than it could be written to the VARTAB file.

Report Action: Warning messages are generated at the beginning of VARDATA file and on lines where data was lost.

User Response: Scale down the amount of your variable data or contact ATC support.

DISTILLATION: Read the Keylist (RKEYxxxx)

RKEY001E Required operand omitted

Explanation: An operand required to run Distillation has not been specified.

Distillation Action: Distillation terminates.

User Response: Specify the omitted operand and retry.

RKEY002E Error *rc* from LISTDSI for *dsn*

Explanation: An error has occurred while obtaining data set information for *dsn*. *rc* contains the return code from LISTDSI.

Distillation Action: Distillation terminates.

User Response: Correct the error indicated by the return code and retry.

RKEY003E Error *rc* allocating *dsn* like *model dsn*

Explanation: An error has occurred while allocating *dsn* like the model data set. *rc* contains the return code from allocate.

Distillation Action: Distillation terminates.

User Response: Correct the error indicated by the return code and retry.

RKEY004E New Master DCB not identical to Master DCB

Explanation: The DCBs for the original data file and the distilled data file do not match.

Distillation Action: Distillation terminates.

User Response: Reallocate the data set to contain the distilled data with the same DCB attributes as the original data set.

RKEY005E Master New file volser specified without Unit (or vice versa)

Explanation: You have specified a UNIT or VOLSER, but not both, for either the original data file or the distilled data file.

Distillation Action: Distillation terminates.

User Response: If a data set is not cataloged, specify both a UNIT and VOLSER parameters.

REPORT: Executing the JCL (RPT03xx)

CA may issue the following messages while executing the report JCL:

RPT0301E REPORT: Error - No procedures or entry name found in listing.

Explanation: The CA report program (REPORT) did not find an entry name in the listing being processed. For 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 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.

Report 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.

RPT0302E REPORT: Error - Entry name: 'ename' not found in BRKTAB file

Explanation: The entry name of the listing 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.

Report Action: Report terminates.

User Response: Verify that the listing you are trying to process is one that has coverage data in the BRKTAB file being used. You should be able to find the first CSECT name of your listing in the BRKOUT file.

RPT0303E REPORT: Error - 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.

Report Action: Report terminates.

User Response: Verify that the listing you are trying to process 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.

RPT0305E REPORT: Error - No space for Bit map

Explanation: The CA Report program (REPORT) received an error code from the operating system command used to request storage for the Bit map.

Report Action: Report terminates.

User Response: Increase the region size on your job card for this step. CA obtains job card information from the site defaults file.

RPT0307E REPORT: Error - no space for BRKTAB table

Explanation: The CA report program (REPORT) received an error code from the operating system command used to request storage for the BRKOUT table. For storage requirements, see Appendix B, “ATC Requirements and Resources” on page 383.

Report Action: Report terminates.

User Response: Increase the region size on your job card for the report step. CA obtains job card information from the site defaults file.

RPT0310E REPORT: Error - Illegal listing input file

Explanation: Report detected something wrong with the assembler listing. Report expects a standard Assembler H or High Level Assembler listing format.

Report Action: Report terminates.

User Response: Check that you are using a standard Assembler H or High Level Assembler listing.

RPT0330E REPORT: Error - Invalid BRKTAB file

Explanation: The input BRKTAB file does not contain valid BRKTAB data.

Report Action: Report terminates.

User Response: Ensure that SETUP has completed creating the BRKTAB file successfully. If it has, contact ATC support, else correct any errors that SETUP issued and rerun the job.

RPT0331E REPORT: Error - Invalid BRKOUT file

Explanation: The input BRKOUT file does not contain valid BRKOUT data.

Report Action: Report terminates.

User Response: Ensure that CASTOP or CADATA has been issued in order to create the BRKOUT file. If it has, contact ATC support.

RPT0392E REPORT: Error - Invalid file name file_name cannot be opened

Explanation: The file `file_name` cannot be opened.

Report Action: Report terminates.

User Response: Ensure that the DDNAME is specified in the JCL, that the file it points to exists, is available to the Report job, and that the job that created the file completed successfully.

RPT0393E REPORT: Invalid invocation.

Explanation: One or more of the parameters passed to Report are invalid.

Report Action: Report terminates.

User Response: Refer to the *User's Guide* for a list of the valid parameters for Report.

SAA: Source Audit Assistant (SAACxxxx)

SAAC001E Old dataset *dsn* not found.

Explanation: The original source listing cannot be found.

SAA Action: Source Audit Assistant terminates.

User Response: Make sure the original source listing is accessible.

SAAC002E New dataset *dsn* not found.

Explanation: The changed source listing cannot be found.

SAA Action: Source Audit Assistant terminates.

User Response: Make sure the modified source listing is accessible.

SAAC003I New and old files are being compared

Explanation: The original source listing and the modified source listing are being compared.

SAA Action: Source Audit Assistant continues.

User Response: None.

SAAC004I Comparison results are being filtered

Explanation: The comparison results are being modified based on the filter criteria you specified (i.e. comment lines are being removed, etc.)

SAA Action: Source Audit Assistant continues.

User Response: None.

SAAC005I Reformatted lines are being removed

Explanation: Lines that did not changed except for their indentation are being removed from the comparison.

SAA Action: Source Audit Assistant continues.

User Response: None.

SAAC006I New and old files are being analyzed

Explanation: The original source and the modified source are being analyzed.

SAA Action: Source Audit Assistant continues.

User Response: None.

SAAC007I Comment lines are being filtered

Explanation: Comment lines are being removed from the comparison.

SAA Action: Source Audit Assistant continues.

User Response: None.

SAAC008I Declare lines are being filtered

Explanation: Data declarations lines are being removed from the comparison.

SAA Action: Source Audit Assistant continues.

User Response: None.

SAAC009I Output being formatted

Explanation: Comparison output is being formatted.

SAA Action: Source Audit Assistant continues.

User Response: None.

SAAC010I SAA Comparison done RC = *rc*

Explanation: The Source Audit Assistant has completed. *rc* indicates the success of the comparison.

SAA Action: Source Audit Assistant terminates.

User Response: None.

SAAC011I *data*

Explanation: This message is produced in association with another message and contains data pointed to by the previous message.

SAA Action: Source Audit Assistant continues.

User Response: See the action described by the associated message.

SAAC012W *data*

Explanation: This message is produced in association with another message and contains data pointed to by the previous message.

SAA Action: Source Audit Assistant continues.

User Response: See the action described by the associated message.

SAAC013W *data*

Explanation: This message is produced in association with another message and contains data pointed to by the previous message.

SAA Action: Source Audit Assistant terminates.

User Response: See the action described by the associated message.

SAAC014S *data*

Explanation: This message is produced in association with another message and contains data pointed to by the previous message.

SAA Action: Source Audit Assistant terminates.

User Response: See the action described by the associated message.

SAAC015I Listing files being stripped.

Explanation: This message is produced when source code is being stripped out of listing files for subsequent comparison by SAA.

SAA Action: Source Audit Assistant continues.

User Response: None.

SAA POSTPROCESSOR: Source Audit Assistant Change Validation Report (SAAFxxxx)

SAAF001E Output report file Dsname not found

Explanation: The data set to contain the SAA change validation report cannot be found.

Action: SAA postprocessor terminates.

User Response: Make sure the report data set is allocated and accessible.

SAAF002I SAA change validation report written to *dsn*

Explanation: The Source Audit Assistant change validation report has been written.

Action: SAA postprocessor terminates.

User Response: None.

SAAF003I SAA change validation report JCL has been created

Explanation: The change validation report JCL has been created in the JCL library.

Action: SAA postprocessor terminates.

User Response: None.

SAAF005E Error *rc* **from EXECIO reading** *dsn*

Explanation: The input data set *dsn* could not be read. *rc* contains the return code from EXECIO.

Action: SAA postprocessor terminates.

User Response: Ensure the input data set is in the proper format and is accessible.

SAAF006E *dsn* **was not generated from a COBOL source file**

Explanation: The specified data set is not a COBOL source listing.

Action: SAA postprocessor terminates.

User Response: Specify the proper source listing data set and retry.

SAAF007E Error *rc* **from EXECIO writing** *dsn*

Explanation: The output data set *dsn* could not be written. *rc* contains the return code from EXECIO.

Action: SAA postprocessor terminates.

User Response: Ensure the output data set is accessible.

SAAF008E Invalid DBCS string *str* on line *num* of the seed list data set.

Explanation: An invalid DBCS string *str* was found on line *num* in the seed list data set with DBCS scanning enabled.

Action: The input line is discarded and processing continues.

User Response: Ensure that any DBCS in the seed list complies with the rules for generating DBCS as required by the compilers, REXX, and so on. For information about DBCS support, see Appendix C, “DBCS Support” on page 391.

SAAF009E Invalid DBCS string in the compare data set at input *str*.

Explanation: An invalid DBCS string *str* was found in the compare data set.

Action: The input is discarded and processing continues.

User Response: Ensure that DBCS scanning is only enabled if the source was compiled with the DBCS (COBOL) or GRAPHIC (PL/I) compiler option and it compiled without errors.

SAA EXECUTION: Source Audit Assistant Execution (SAARxxxx)

SAAR001E Old dataset *dsn* not found.

Explanation: The original source listing cannot be found.

SAA Action: Source Audit Assistant terminates.

User Response: Make sure the original source listing is accessible.

SAAR002E New dataset *dsn* not found.

Explanation: The changed source listing cannot be found.

SAA Action: Source Audit Assistant terminates.

User Response: Make sure the modified source listing is accessible.

SAAR003E A member name or * must be specified for *dsn*

Explanation: *dsn* is a partitioned data set so a member name or * (that is, ALL) must be specified.

SAA Action: Source Audit Assistant terminates.

User Response: Specify a member name or *.

SAAR005E Error writing PDS header RC = *rc*

Explanation: An error occurred when writing the header for a partitioned data set. *rc* contains the return code from the write.

SAA Action: Source Audit Assistant terminates.

User Response: Correct the error indicated by the return code and retry.

SAAR006I Comparing member *member name*

Explanation: SAA is currently comparing the specified member in the old and new data sets.

SAA Action: Source Audit Assistant continues.

User Response: None.

SAAR007E Invalid new and old file DSORG combination

Explanation: The DSORG for the original source listing data set and the modified source listing data set must be the same.

SAA Action: Source Audit Assistant terminates.

User Response: Ensure both data sets have the same DSORG or specify a member name if one is a partitioned data set.

SAAR008I Comparing Partitioned Datasets

Explanation: The original source listing data set and the modified source listing data set are being compared.

SAA Action: Source Audit Assistant continues.

User Response: None.

SAAR009I Error writing Compare Column Dataset

Explanation: An error occurred when writing output to the *yourid*.TEMP.SAA.CMP.COL data set.

SAA Action: Source Audit Assistant continues.

User Response: None.

SAAR010E A member name must be specified.

Explanation: The `SAA Output Compare File` fields must include a member name if the compare file is a PDS (partitioned data set) and the `New Source File` is a physical sequential file.

SAA Action: Source Audit Assistant returns to the `SAA Background Execution Parameters` panel.

User Response: Provide a member name for the `SAA Output Compare File`.

SAAR011E Member, pattern, or * required.

Explanation: If the `New Source File` is a PDS (partitioned data set), then a member name, a matching pattern (for example, COB*), or an * is required.

SAA Action: Returns to the `SAA Background Execution Parameters` panel.

User Response: Provide a member name, a matching pattern, or an * for the `New Source File` as a member name.

SAAR012E No new members found in new data.

Explanation: SAA did not find any members in the new source file.

SAA Action: Returns to the `SAA Background Execution Parameters` panel.

User Response: Determine and enter the correct data set name or names.

SAAR013E No pattern matches found.

Explanation: The pattern given in the new source file and/or old source file resulted in no member name matches.

SAA Action: Returns to the `SAA Background Execution Parameters` panel.

User Response: Change the pattern to be used to a specific member name (in both fields) or to an *.

SAAR014E Invalid length for member name

Explanation: The length of the member name is greater than 8 characters.

SAA Action: Source Audit Assistant returns to the `SAA Foreground Execution Parameters` panel or the `SAA Background Execution Parameters` panel.

User Response: Check the member name for spelling errors, make any necessary corrections, and reenter the information.

SAAR015E Compare not partitioned

Explanation: A member name was specified with the Compare File Dsn that is physical sequential instead of partitioned data set organization.

SAA Action: Source Audit Assistant returns to the `SAA Background Execution Parameters` panel.

User Response: Check the spelling of the data set name and retry.

SAAR016E Global file-name character not allowed

Explanation: Global file-name character (*) for member name not allowed for the SAA Output Compare File on the `SAA Background Execution Parameters` panel.

SAA Action: Source Audit Assistant returns to the `SAA Background Execution Parameters` panel.

User Response: Enter a member name or eliminate the global file-name character from the SAA Output Compare File data set name and retry.

SAAR017E Compare File must be SEQ

Explanation: The SAA Output Compare File used for SAA Foreground processing must be a physical sequential (PS) data set.

SAA Action: Source Audit Assistant returns to the `SAA Foreground Execution Parameters` panel.

User Response: Enter the name of a physical sequential data set for the SAA Output Compare File data set name and retry.

SAAR018E Member name not allowed

Explanation: The SAA Output Compare File used for SAA Foreground processing will be a physical sequential (PS) data set.

SAA Action: Source Audit Assistant returns to the `SAA Foreground Execution Parameters` panel.

User Response: Remove the member name from the SAA Output Compare File data set name and retry.

SAAR019E Log File must be SEQ

Explanation: The SAA Output Log File used for SAA Foreground processing must be a physical sequential (PS) data set.

SAA Action: Source Audit Assistant returns to the `SAA Foreground Execution Parameters` panel.

User Response: Enter the name of a physical sequential data set for the SAA Output Log File data set name and retry.

SAAR020E Member name not allowed

Explanation: The SAA Output Log File used for SAA Foreground processing will be a physical sequential (PS) data set.

SAA Action: Source Audit Assistant returns to the `SAA Foreground Execution Parameters` panel.

User Response: Remove the member name from the SAA Output Log File data set name and retry.

SETUP: Executing the JCL (SP601xx)

The setup program ends with a return code corresponding to the number of the message. For example, if message SP60101E is issued, return code `0101` is displayed.

The following messages may be issued while executing the setup JCL:

SP60101E SETUP: Error - Illegal listing type in CA control file

Explanation: The `Listing type` value (first field) in the CACTL was neither `B` nor `P`. This character is passed to SETUP as parameter 1.

Setup Action: Setup terminates.

User Response: Correct the value. Rerun setup.

SP60108E SETUP: Error - No BRKTAB written. Too few bpoints.

Explanation: No BRKTAB was written because SETUP could not find assembler instructions at which to insert breakpoints.

Setup Action: Setup terminates.

User Response: Check the input listing to SETUP and verify that it is correct, including the assembler statements.

SP60109E SETUP: Error - Invalid PA type specified

Explanation: The PA type passed to SETUP is invalid. For COBOL and PL/I, it must be `0`.

Setup Action: Setup terminates.

User Response: Correct the PA type in the JCL.

SP60111E SETUP: Illegal assembler listing

Explanation: The assembler listing is not valid.

Setup Action: Setup terminates.

User Response: Check that the assembler listing passed to setup is a valid Assembler H or High Level Assembler listing.

SP60114E SETUP: Error - DBGTAB file not valid.

Explanation: The DBGTAB file does not contain valid data.

Setup Action: Setup terminates.

User Response: Ensure that a VAREAD step was run successfully to create the DBGTAB file. If it was, contact ATC support, else correct any errors that VAREAD issued and rerun the job.

SP60115E SETUP: Error allocating storage for DBGTAB file statements.

Explanation: SETUP could not acquire enough storage to process the DBGTAB file statements.

Setup Action: Setup terminates.

User Response: Specify a larger REGION size for the SETUP job.

SP60116E SETUP: Error allocating storage for DBGTAB file variables.

Explanation: SETUP could not acquire enough storage to process the DBGTAB file variables.

Setup Action: Setup terminates.

User Response: Specify a larger REGION size for the SETUP job.

SP60117E SETUP: Error - Invalid statement read type - must be R/C/N.

Explanation: A DBGTAB record contains an invalid statement read type.

Setup Action: Setup terminates.

User Response: Ensure that a VAREAD step was run successfully to create the DBGTAB file. If it was, contact ATC support, else correct any errors that VAREAD issued and rerun the job.

SP60118E SETUP: Error allocating storage for DBGTAB character data.

Explanation: SETUP could not acquire enough storage to process the DBGTAB character data.

Setup Action: Setup terminates.

User Response: Specify a larger REGION size for the SETUP job.

SP60119E SETUP: Error - Cannot find requested listing in DBGTAB.

Explanation: No DBGTAB entry was found for the compiler listing being processed.

Setup Action: Setup terminates.

User Response: Ensure that a VAREAD step was run successfully to create the DBGTAB file. If it was, contact ATC support, else correct any errors that VAREAD issued and rerun the job.

SP60120E SETUP: Error - Cannot find assembler line that reads TGT.

Explanation: For UTA and DA, SETUP reads the assembler listing for the statement where the TGT register is loaded. This line was not found.

Setup Action: Setup terminates.

User Response: Verify that you have passed a correct listing to SETUP, that includes the assembler listing. If so, contact ATC support.

SUMMARY: Executing the JCL (SUM0xxxx)

CA may issue the following messages while executing the summary JCL:

SUM0301E SUMMARY: Error - no space for BRKTAB file

Explanation: The CA Summary program (SUMMARY) received an error code from the operating system command used to request storage for the BRKTAB file. For storage requirements, see Appendix B, “ATC Requirements and Resources” on page 383.

Summary Action: Summary terminates.

User Response: Increase the region size on your job card for this step. CA obtains job card information from the site defaults file.

SUM0302E SUMMARY: Error - EOF of BRKTAB before finding the testid test case

Explanation: The testid test case was not found in the BRKTAB file provided.

Summary Action: Summary terminates.

User Response: Make sure that the BRKTAB file in the BRKTAB DD statement of the JCL is the same one that was used when running 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.

SUM0305E SUMMARY: Error - No space for BP table or Bit map

Explanation: The CA Summary program (SUMMARY) received an error code from the operating system command used to request storage for the BP table or Bit map.

Summary Action: Summary terminates.

User Response: Increase the region size on your job card for this step. CA obtains job card information from the site defaults file.

SUM0330E SUMMARY: Error - Invalid BRKTAB file

Explanation: The BRKTAB file is not valid.

Summary Action: Summary terminates.

User Response: Make sure you are passing the correct BRKTAB file in the BRKTAB DD statement of the JCL.

SUM0331E SUMMARY: Error - Invalid BRKOUT file

Explanation: The BRKOUT file is not valid.

Summary Action: Summary terminates.

User Response: Make sure you are passing the correct BRKOUT file in the BRKOUT DD statement of the JCL.

SUM0405E SUMMARY: Invalid invocation.

Explanation: The parameter passed to Summary is invalid.

Summary Action: Summary terminates.

User Response: Refer to “SUMMARY” on page 104 for a list of the valid parameters.

SUM0406E SUMMARY: Error - Invalid file name file_name cannot be opened

Explanation: The file `file_name` cannot be opened.

Summary Action: Summary terminates.

User Response: Ensure that the DDNAME is specified in the JCL, that the file it points to exists, is available to the Report job, and that the job that created the file completed successfully.

VAREAD: Extracting Variable Information from Listing (VAR0xxx)

DA, UTA, and targeted summary may issue the following messages while executing the VAREAD program:

VAR0010W VAREAD: Warning - Level 88 flag variables cannot be monitored.
Variable: str

Explanation: The level 88 flag variable *str* was listed to be monitored. These flags cannot be monitored.

Action: The variable is discarded.

User Response: Ensure that the variable was specified correctly.

VAR0011W VAREAD: Warning - Invalid control character found in internal control card.
At input: str

Explanation: The first character in the internal control card, *str,* is not a valid control character.

Action: Output message. The line is ignored.

User Response: Check for errors from the CVTCTL step of Setup. Look for errors in or around fields in the control file that match elements in *str*.

Note: The control file is converted by CVTCTL into internal control cards. The control cards may or may not contain data in a recognizable format, but will be provided for support purposes.

VAR0012W VAREAD: Warning - Internal control card sequence error.
At Input: str

Explanation: The internal control card *str* is not valid at this point in the internal control card sequence.

Action: Output message. The entry is ignored.

User Response: Check for errors from the CVTCTL step of Setup. Look for errors in or around fields in the control file that match elements in *str*.

VAR0013W VAREAD: Warning - No data found in internal control card.
At Input: str

Explanation: No data fields were found in the internal control card *str*.

Action: Output message. The entry is ignored.

User Response: Check for errors from the CVTCTL step of Setup.

VAR0014W VAREAD: Warning - Invalid variable level number in internal control card.
At Input: str

Explanation: The variable level number field on internal control card *str* was either more than 2 numeric digits or was an invalid alphabetic value.

Action: Output message. If numeric, truncate to 2 digits, otherwise the line is ignored. If ignored, this will typically be followed by `Variable not found` messages.

User Response: Check for errors from the CVTCTL step of Setup. Check for control file errors in any variable entries that contain a level with the variable name found in *str*.

VAR0015W VAREAD: Warning - Missing data field in internal control card.
At Input: str

Explanation: One or more required keyword fields were missing in the control file.

Action: Output message. The control card is ignored.

User Response: Check for errors from the CVTCTL step of Setup. Include all required fields in the control file.

VAR0016W VAREAD: Warning - Invalid variable residence entry in internal control card.
At Input: str

Explanation: In an internal control card, an invalid value was found for the variable residence. This relates to the VARIABLE and FILE keywords of the coverage statement in the control file. This error should only occur as a side effect of another problem.

Action: Output message. The control card is ignored, but processing continues.

User Response: Check for errors from the CVTCTL step of Setup.

VAR0017W VAREAD: Warning - Invalid variable type entry in internal control card.
At Input: str

Explanation: In an internal control card, an invalid value was found for the variable type field. This relates to the CHAR keyword of the coverage statement in the control file. This error should only occur as a side effect of another problem.

Action: Output message. The control card is ignored, but processing continues.

User Response: Check for errors from the CVTCTL step of Setup.

VAR0018W VAREAD: Warning - Invalid variable offset entry in control file.
At Input: str

Explanation: A nonnumeric character was found in the operand of either the OFFSET or POSITION keywords of a coverage statement, or the KEYOFFSET, or KEYPOSITION keywords in a file statement in the control file.

Action: Output message. The entry is ignored, but the processing continues.

User Response: Check for errors from the CVTCTL step of Setup. Correct the control file entry.

VAR0019W VAREAD: Warning - Invalid variable length entry in internal control card.
At Input: str

Explanation: A nonnumeric character was found in the operand *str* to the LENGTH or KEYLEN keyword in the control file.

Action: Output message. The entry is ignored, but processing continues.

User Response: Check for errors from the CVTCTL step of Setup. Correct the control file entry.

VAR0020W VAREAD: Warning - Invalid variable read times entry in internal control card.
At Input: str

Explanation: A nonnumeric character was found in the operand *str* of the READEVERY keyword in the control file.

Action: Output message. The entry is ignored, but processing continues.

User Response: Check for errors from the CVTCTL step of Setup. Correct the control file entry.

VAR0021W VAREAD: Warning - Invalid variable read max entry in internal control card.
At Input: str

Explanation: A nonnumeric character was found in the operand *str* of the MAXSAVE keyword in the control file.

Action: Output message. The statement is ignored, but processing continues.

User Response: Check for errors from the CVTCTL step of Setup. Correct the control file entry.

VAR0022W VAREAD: Warning - Invalid statement list entry in internal control card.
At Input: str

Explanation: A nonnumeric entry was found as the operand to the STMTS keyword in the control file.

Action: Output message. The statement is ignored, but processing continues.

User Response: Check for errors from the CVTCTL step of Setup. Correct the control file entry.

VAR0023I VAREAD: Informational - An unrecognized base locator type was found in the listing.
Base Locator: str

Explanation: An unknown base locator type was found in the listing file.

Action: This message is issued.

User Response: None required.

VAR0025W VAREAD: Error - Read exceeds the variable length.
Read length truncated to variable length.
Variable: *str1*

Explanation: The combined read length and offset given in the control file for the variable *str1* exceeds the actual length of the variable as read from the listing file data map.

Action: Reset the read length to the actual variable data length less the given offset.

User Response: Ensure that the variable read length and offset were correctly specified in the control file.

VAR0026W VAREAD: Warning - Read exceeds the maximum length supported.
Read length truncated to nnnn.
Variable: str1

Explanation: The read length specified for the variable `str1` exceeds the maximum supported read length of nnnn.

Action: Reset the read length to nnnn.

User Response: Ensure that the variable read length and offset were correctly specified in the control file. If the desired read length exceeds the supported maximum, use multiple reads with appropriate offsets.

VAR0027W VAREAD: Warning - Offset exceeds the variable data length.
Offset set to 0.
Variable: str1

Explanation: The read offset specified for the variable `str1` exceeds the actual data length of variable `str1`.

Action: Reset the offset to 0.

User Response: Ensure that the variable read offset is correctly specified in the control file.

VAR0028W VAREAD: Warning - Invalid wildcard character found in control file.
Variable: str

Explanation: The wildcard character * was found in an invalid location in the control file, associated with the variable *str.*

Action: Output message and attempt to process normally.

User Response: Check for errors from the CVTCTL step of Setup. Edit the control file to remove or correct the wildcard usage.

VAR0029W VAREAD: Warning - Failed to find a variable in the listing file.
Variable Name: var-name

Explanation: The variable *var-name* was not found in the listing file.

Action: Output message and continue with the next variable.

User Response: Edit the control file to verify that the variable name is spelled correctly. If there are multiple programs in the listing, also check that the correct program is being search for the given variable.

VAR0030I VAREAD: Informational - No statements were found for a variable.
Variable Name: str1
Program ID: str2

Explanation: The variable *str1* in the program *str2* was not accessed on any statements within the specified search.

Action: The variable is discarded.

User Response: Edit the control file to ensure that the variable was specified in the form desired. Try using the FULL option for the variable statement list. The variable may be unused.

VAR0031W VAREAD: Warning - Unknown line format in listing file data map.
Line: str

Explanation: The line *str* is not a recognized format for a listing file data map line.

Action: Ignore the line and continue.

User Response: Contact ATC support.

VAR0032W VAREAD: Warning - Invalid action entry in internal control card.
At Input: str

Explanation: An invalid action key was found in an internal control card. This error should only occur as a side effect of another problem.

Action: Output message. The control card is ignored, but processing continues.

User Response: Check for errors from the CVTCTL step of setup.

VAR0033W VAREAD: Warning - Invalid sign entry in internal control card.
At Input: str

Explanation: An invalid sign value was found in an internal control card. This error should only occur as a side effect of another problem.

Action: Output message. The control card is ignored, but processing continues.

User Response: Check for errors from the CVTCTL step of setup.

VAR0034W VAREAD: Warning - Invalid value entry in internal control card.
At Input: str

Explanation: A nonnumeric value was found in an internal control card where a numeric value was expected. This error should only occur as a side effect of another problem.

Action: Output message. The control card is ignored, but processing continues.

User Response: Check for errors from the CVTCTL step of setup.

VAR0050W VAREAD: Warning - Program name not found in listing.
File Name: str1

Explanation: The program name `str1` was not found in the listing file.

Action: Output message.

User Response: Ensure that the program name is entered under the correct listing file in the control file, and that it is spelled correctly.

VAR0051W VAREAD: Warning - Targeted statement code is not in executable code.
In listing: *list*

Explanation: Statement number num was targeted in listing *list*, but is not within executable code. The statement cannot be resolved to an executable statement.

Action: The statement is discarded and processing continues.

User Response: Ensure that only executable statements are targeted. See the description of the TARGETSTMT control statement in the “Target Control File” on page 106 of this document for the rules for determining statement numbers.

VAR0052W VAREAD: Warning - Targeted statement(s) was resolved to different executable statement(s).
In listing: *list*
Statement(s): *stmts*

Explanation: One or more statement numbers was targeted in listing *list*, but is not an executable statement. In each pairing in the statements list the first number was given, but was not an executable statement. The second is the statement number actually used in generating the targeted summary report.

Action: The second statement of each pair is used in place of the first.
Processing continues.

User Response: Ensure that only executable statements are targeted. See the description of the TARGETSTMT control statement in the “Target Control File” on page 106 of this document for the rules for determining statement numbers.

VAR0060W VAREAD: Warning - Key gathering and variable monitoring detected in the same run

Explanation: Keys from file reads (DA) are being read in the same run. as variables are being monitored (UTA). The values read from the keys and variables will be mingled in the output reports and key lists. You must separate these if you want to do distillation.

Action: This message is issued.

User Response: Create two control files, one for DA and the other for UTA, and then run DA and UTA as separate jobs.

VAR0061E VAREAD: Error - UTA or DA commands mixed with Targeted Coverage commands.

Explanation: UTA or DA commands were mixed with targeted coverage commands in the same run.

Action: The VAREAD program terminates.

User Response: Check for an invalid program invocation. Contact ATC support.

VAR0100E VAREAD: Error - Control file open failure.

Explanation: The program was unable to open the control file.

Action: Output message and terminate.

User Response: Ensure that the file name and path were correctly specified, and that the file exists.

VAR0101E VAREAD: Error - Listing file open failure.

Explanation: The program was unable to open the listing file.

Action: Output message and terminate.

User Response: Ensure that the file name and path were correctly specified, and that the file exists.

VAR0102E VAREAD: Error - DBGTAB file open failure.

Explanation: The program was unable to open the DBGTAB file.

Action: Output message and terminate.

User Response: Ensure that the file name and path were correctly specified, and that the file exists.

VAR0104E VAREAD: Error - Invalid listing file.

Explanation: The listing file passed to the VAREAD program could not be identified as a valid listing file for any supported language or compiler.

Action: The VAREAD program terminates.

User Response: Ensure that the file is a valid listing file from a supported compiler, and that the correct compiler options were specified. For a list of compiler options, see “Setup” on page 231.

VAR0107E VAREAD: Error - Memory allocation failure for statement record.

Explanation: A request for a memory allocation for a statement record failed.

Action: Output message and terminate.

User Response: Increase the region size on your job card for the setup step.

VAR0108E VAREAD: Error - Memory allocation for variable record failed.

Explanation: A request for a memory allocation for a variable record failed.

Action: Output message and terminate.

User Response: Increase the region size on your job card for the setup step.

VAR0120E VAREAD: Error - No Data Cross-reference found in the compiler listing.

Explanation: No data cross-reference was found in the compiler listing file.

Action: Output message and terminate.

User Response: Ensure that the compiler option to generate a cross-reference was used for the compile.

VAR0121E VAREAD: Error - No Data Map section found in the compiler listing.

Explanation: No data map section was found in the compiler listing file.

Action: Output message and terminate.

User Response: Ensure that the compiler option to generate the data map was used for the compile.

VAR0122E VAREAD: Error - No assembler code section found in the compiler listing.

Explanation: No assembler code section was found in the compiler listing file.

Action: Output message and terminate.

User Response: Ensure that the compiler option to include pseudo-assembler code in the listing was used for the compile.

VAR0123E VAREAD: Error - No Table of Offsets found in the compiler listing.

Explanation: The listing section labeled “TABLES OF OFFSETS AND STATEMENT NUMBERS” was not found. The OFFSET compiler option is required to produce this table.

Action: The VAREAD program terminates.

User Response: Ensure that you specify the OFFSET option when you compile.

VAR0150E VAREAD: Error - Listing file name argument required.

Explanation: The listing file name is a required parameter for this program.

Action: Output message and terminate.

User Response: Include the listing file name as a parameter exactly as it appears in the control file.

VAR0151E VAREAD: Error - No entry found in control file for specified listing.
File Name: str1

Explanation: The listing file name `str1` was not found in the control file.

Action: Output message and terminate.

User Response: Include the listing file name as a parameter exactly as it appears in the control file.

VAR0157E VAREAD: Error - A fully qualified variable name was too long for the buffer.
Variable Name: str1

Explanation: The fully qualified variable name `str1` was too long for the program buffer.

Action: Output message and terminate.

User Response: Check that an R, C, or N line follows each set of type `L` lines which fully qualify a variable in the control file.

Note: The buffer is longer than the longest valid COBOL variable name.

VAR0160E VAREAD: Error - Internal error 160.

Explanation: An internal error has caused a program failure.

Action: Output message and terminate.

User Response: Contact ATC support.

VAR0165E VAREAD: Error - Invalid internal control card.

Explanation: An invalid internal control card has caused a program failure, or you coded a control card for a tool that you are not licensed for.

Action: Output message and terminate.

User Response: Ensure that all parameters in the control file are specified correctly, or remove the control card that specifies the unlicensed function from your control cards.

EXECUTE: Buffer Monitor (WVAxxxx)

The following messages may be issued while executing the buffer monitor:

WVA5010E Error writing VARTAB file

Explanation: A user tried to stop a session, but the VARTAB file of variable read data could not be written.

Execute Action: VARMON3, the program that writes full buffers of variable data, terminates.

User Response: Check that the VARTAB file (DD name VARTAB in the execute JCL X*nnnnnn*) is valid.

WVA5101E Minimum wait time >= Maximum wait time

Explanation: The minimum wait time specified for the buffer monitor in the ATC defaults was greater than the maximum time.

Execute Action: The buffer monitor program terminates.

User Response: Terminate the monitor (using the CAQUIT command), correct the minimum and maximum times in the ATC defaults, and recreate the monitor JCL from the panels.

WVA5102W The lowest wait time is lower than 10 (.1 seconds). It is set to 10.

Explanation: The minimum wait time specified for the buffer monitor in the ATC defaults was less than 10 (.1 seconds). It is set to 10, and the buffer monitor continues.

Execute Action: The buffer monitor program continues.

User Response: Correct the minimum wait time in the ATC defaults, and recreate the monitor JCL so that you will not get this message.

WVA5103W The highest wait time is higher than 1600 (16 seconds). It is set to 16.

Explanation: The maximum wait time specified for the buffer monitor in the ATC defaults was greater than 1600 (16 seconds). It is set to 1600, and the buffer monitor continues.

Execute Action: The buffer monitor program continues.

User Response: Correct the maximum wait time in the ATC defaults, and recreate the monitor JCL so that you will not get this message.

WVA5104E Session name *xxxxxx* not found. Write buffer program ending.

Explanation: The session name xxxxxx was not found as an active session. This should only occur if you have modified your monitor JCL incorrectly.

Execute Action: The buffer monitor program terminates.

User Response: Terminate the monitor (using the CAQUIT command), and correct the session name in the VARMON3 step of the monitor JCL.

WVA5106E No session table exists. Cannot continue.

Explanation: The monitor is not installed, or has been corrupted.

Execute Action: VARMON3, the program that writes full buffers of variable data, terminates.

User Response: Reinstall the monitor.

WVA5108I The session has ended - variable write program for this session ending.

Explanation: The session being monitored has ended. The buffer monitor program is ending. This is the normal termination message.

Execute Action: The buffer monitor program terminates.

User Response: None.

WVA5109E Error opening VARTAB file

Explanation: A user tried to stop a session, but the VARTAB file of variable read data could not be opened.

Execute Action: VARMON3, the program that writes full buffers of variable data, terminates.

User Response: Check that the VARTAB file (DD name VARTAB in the execute JCL X*nnnnnn*) is valid.

ZAPTXT: Executing the JCL (ZAP00yy or ZAP88xx)

The following messages may be issued while executing the ZAPTXT or ZAPLM JCL:

ZAP8802E ZAPTXT: Error - TXT record not found to match this BRKTAB record.
ZAP8802E ZAPTXT: OFFSET(oooo) OPCODE(cccc)

Explanation: During the scan of a COBOL or PL/I 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 COBOL or PL/I source listings. The ZAPTXT program reads the BRKTAB data set and looks through the object for each breakpoint. If a breakpoint cannot be found, the ZAPTXT program terminates.

Zaptxt Action: ZAPTXT program terminates.

Possible Cause: Wrong object module used as input to ZAPTXT program.

User Response: Ensure the object module passed as input to the ZAPTXT program was created by the same source listing as the BRKTAB file. Rerun ZAPTXT JCL.

ZAP8803W ZAPTXT: Warning - Breakpoint record not found. Errors possible
ZAP8803W ZAPTXT: Check the RPTMSGS dataset for additional information.

Explanation: No match in the object module for this breakpoint in BRKTAB.

Zaptxt Action: ZAPTXT program continues.

Possible Cause: BRKTAB and object module do not match.

User Response: Ensure the object module passed as input to the ZAPTXT program was created by the same source listing as the BRKTAB file. Rerun ZAPTXT JCL.

ZAP8804E ZAPTXT: Error - BRKTAB header not found! Cannot continue.
ZAP8804E ZAPTXT: BRKTAB header is not record # nnnnn

Explanation: A BRKTAB header number passed as a PARM to the ZAPTXT program could not be found. The number passed as the PARM is outside the range of possibilities for this BRKTAB.

Zaptxt Action: ZAPTXT program terminates.

Possible Cause: Errors during QSETUP creation of setup JCL.

User Response: Complete the following steps:

1. Rerun setup.
2. Locate the header record in BRKTAB and update the PARM passed in the ZAPTXT JCL.

ZAP8805E ZAPTXT: Error - Storage could not be allocated. Program ending.
ZAP8805E ZAPTXT: Insufficient storage.

Explanation: ZAPTXT could not acquire enough storage in order to process the object module.

Zaptxt Action: ZAPTXT program terminates.

Possible Cause: Not enough storage was available to the job.

User Response: Specify a larger REGION size for the SETUP job.

ZAP8806E ZAPLM: ERROR - Cannot open BRKTAB file created during setup.

Explanation: The BRKTAB file of breakpoint data created in a previous step cannot be opened.

ZAPLM 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.

ZAP8807E ZAPLM: ERROR - Cannot open output file for VER/REP records.

Explanation: The output file containing update records for the AMASPZAP utility that instruments load modules cannot be opened.

ZAPLM 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.

ZAP8808E ZAPLM: ERROR - Illegal BRKTAB file.

Explanation: The BRKTAB file of breakpoint data created in a previous step is not a correct BRKTAB.

ZAPLM 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.

Understanding the SAA Log File

When SAA runs the scan phase (named ATCSXSCN), it creates a message data set. This data set contains information about ATCSXSCN processing, as well as error information.

Most of the information in this data set is self-explanatory. You may, however, encounter some numbered error messages as well. The remainder of this chapter explains the ATCSXSCN message format and the messages you may receive.

Description of the Message Format

ATCSXSCN issues only 60 messages per scan, using the following format:

```
n msglevel messagetext comment
```

Where:

n

The error message number.

msglevel

A letter indicating the severity level. The letter is associated with a numerical MSGLEVEL code or return code, as follows. ATCSXSCN issues all messages whose severity is equal to or greater than the message level you specify.

Message Level	Return Code	Description
I	0	Information
W	4	Warning
E	8	Error
S	12	Severe error
T	16	Terminating error condition

messagetext

The character string "_____" in the message text, denotes variable text in the message. When ATCSXSCN actually issues the message, it replaces the variable text with specific information such as a file name, table name, or record number.

comment

An explanation of the message, as appropriate.

Message Description

Informational

ATCSXSCN informs you of actions taken. You probably expect the action. These messages keep you informed of the progress. The list on the following pages does not include informational messages. An example of an informational message is:

```
XSC068  ATCSXSCN completed with Return Code '___'
```

Warning

An ATCSXSCN action was taken or a condition encountered that may not produce the correct results. The condition or action taken is given in the message.

Error

These errors are expected to result in incorrect data. For example, an INCLUDE control statement explicitly requests that a specific module be processed, but the module is not found in the library.

Severe Error

These messages indicate errors that can effect the entire run. No processing is done when this type of error is found.

Terminating error

ATCSXSCN terminates processing when this error occurs.

Restrictions Regarding Embedded Statements

Macro invocations, preprocessor statements, and compiler control statements are frequently embedded within code scanned by ATCSXSCN. ATCSXSCN can handle some embedded statements, such as the following:

```
?receiver_macro = source expression;
```

However, it is unable to process some embedded statements.

Additionally, ATCSXSCN suppresses repetitive messages, such as:

```
Embedded macro invocation skipped...
```

Message List

Number	Severity	Message and Explanation
3	S	*Statementtype* Overflow in *modulename* processing record *recordnumber* This error message occurs when a table in an ATCSXSCN system module overflows. 1. Make sure a source statement is not too long. 2. Make sure you are running with sufficient memory. 3. Ensure that the language you specified is the language in which the program is coded. 4. Make sure your source margins are correct. Your program must use default margins for the language in which it is written. Default margins are as follows: **Language Right Left** COBOL 7 72 C 1 256 C++ 1 256 PL/I 2 72 If you are certain that you have sufficient memory, you specified the correct language, and your source margins are correct, contact ATC support for possible program enhancements.
8	E	Sequential libraries must have exactly one include card with module name.
13	S	Control statement READ error. ATCSXSCN was unable to read the file containing the control statements. Contact ATC support for possible program enhancements.
14	E	CLASS table full. The current release allows up to 2000 classes per program. Contact ATC support for possible program enhancements.
15	E	OBJECT table full. The current release allows up to 2000 objects per program. Contact ATC support for possible program enhancements.
16	W	Analysis error in record *recordnumber* near column *columnnumber*. The following line(s) were ignored: This error might be due to an improper language specification, an ATCSXSCN scan misinterpretation, or a syntax error. ATCSXSCN displays the records that were skipped and not included in the intermediate data files for comment and declaration line determination.
17	S	Unable to OPEN the *filename*.
23	S	*Modulename* not found. The following Module was not found:*Modulename*.
34	T	Error in user control statements. Processing terminated. Contact ATC support and provide the control and list files for analysis.

Number	Severity	Message and Explanation
49	S	Bad PARM control statement. Contact ATC support and provide the control and list files for analysis.
56	E	No STAE Exit will be taken due to errors encountered when STAE was issued. ATCSXSCN attempted an unsuccessful recovery.
57	S	Unable to load *modulename*.
66	E	Language unknown. Contact ATC support and provide the control and list files for analysis.
67	E	Language unknown. Contact ATC support and provide the control and list files for analysis.
69	E	Maximum instantiations exceeded. Table Overflow Error.
72	W	Left Source Margin assumed as 6.
80	E	Failure in allocating storage in Module: *modulename*.
81	E	Control statement syntax error near column *columnnumber*. Contact ATC support and provide the control and list files for analysis.
82	E	Failure to release storage in Module: *modulename*.
84	W	Parsing error. The following lines were ignored:
99	W	Unrecognized Character in record *recordnumber* near column number *columnnumber*.
103	T	No STAE work area passed from supervisor. No retry possible.
104	E	Symbol Where Used Table overflow. Contact ATC support for possible tool enhancement.
110	T	Unable to recover. This error occurs during STAE processing.
115	S	X13 Abend opening SDDS. This error occurs during STAE processing.
117	S	X13 Abend opening PDS. This error occurs during STAE processing.
119	S	X13 Abend opening SYM. This error occurs during STAE processing.
121	S	X13 Abend opening SEQ. This error occurs during STAE processing.
123	S	X13 Abend opening CLR MOD. This error occurs during STAE processing.
125	S	This library cannot be processed because of problems reading SDDS (CLEAR) or directory (PDS).
126	S	Dynamic allocation failed for library *libraryname*. ATCSXSCN was unable to find the library.

Number	Severity	Message and Explanation
140	W	*Modulename* was not found in library. The following file was either empty or not found - *Modulename*. An INCLUDE control statement named a module that was not in the input source library.
145	E	*Modulename* is empty or file not found. The following file was either empty or not found - *Modulename*. This Module does not exist, or an I/O error occurred during search.
146	W	Message limit exceeded. No more messages will be printed. The default message limit is 60. To see all messages in the listing file, specify a message level of 1.
150	S	Called from module *modulename* via Link. This error occurs during STAE processing.
151	S	Called from module *modulename* via SVC *svc#(x)*. This error occurs during STAE processing.
152	S	Last module called *modulename*. This error occurs during STAE processing.
154	S	Called from module *modulename*. This error occurs during STAE processing.
156	W	Imbedded compiler control statement skipped in record *recordnumber*.
159	W	Imbedded macro invocation skipped in record *recordnumber*.
163	W	Imbedded preprocessor statement skipped in record *recordnumber*.
167	W	Empty Library *libraryname*. This error occurs most often on MVS when a PDS library is empty.

Appendix B. ATC Requirements and Resources

This chapter lists the:

- Environments and compilers that support ATC.
- Resources you need to setup, execute, and run reports in Coverage Assistant, Distillation Assistant, and Unit Test Assistant. Data set attributes and data set definition (DD) names are also provided.

Prerequisites and Supported Compilers

ATC was developed and tested in the following environments. These environments and newer releases support ATC.

- MVS/ESA 5.1.0, DFSMS/MVS 1.2, TSO/E 2.5 and ISPF 4.2.1
- MVS/ESA 5.2.2, DFSMS/MVS 1.3, TSO/E 2.5 and ISPF 4.2.1
- Language Environment for MVS & VM 1.5
- OS/390 2.4.0, DFSMS/MVS 1.4, TSO/E 2.6, and ISPF for OS/390 1.3

All IBM supported releases of OS/390 and the OS/390 versions of DFSMS/MVS, TSO/E, ISPF, and Language Environment support ATC.

Coverage Assistant supports the following compilers:

- IBM COBOL for OS/390 & VM 2.1 (plus Millennium Language Extensions [MLE])
- IBM COBOL for MVS & VM 1.2 (plus Millennium Language Extensions [MLE])
- IBM VS COBOL II Release 4.0
- IBM OS/VS COBOL Release 2.4
- IBM PL/I for MVS & VM 1.1.1 (plus Millennium Language Extensions [MLE])
- IBM OS PL/I Optimizing Compiler 2.3.0
- IBM PL/I Optimizing Compiler 1.5.1
- IBM High Level Assembler Version 1 Release 2 and Release 3
- IBM Assembler H Version 2

Distillation Assistant supports the following compilers:

- IBM COBOL for OS/390 & VM 2.1 (plus Millennium Language Extensions [MLE])
- IBM COBOL for MVS & VM 1.2 (plus Millennium Language Extensions [MLE])
- IBM VS COBOL II Release 4.0
- IBM OS/VS COBOL Release 2.4
- IBM PL/I for MVS & VM 1.1.1 (plus Millennium Language Extensions [MLE])
- IBM OS PL/I Optimizing Compiler 2.3.0
- IBM PL/I Optimizing Compiler 1.5.1

© Copyright IBM Corp. 1997, 1999

Unit Test Assistant supports the following compilers:

- IBM COBOL for OS/390 & VM 2.1 (plus Millennium Language Extensions [MLE])
- IBM COBOL for MVS & VM 1.2 (plus Millennium Language Extensions [MLE])
- IBM VS COBOL II Release 4.0
- IBM OS/VS COBOL Release 2.4
- The following PL/I compilers are supported for data warping of file input buffers **only**:
  - IBM PL/I for MVS & VM 1.1.1 (plus Millennium Language Extensions [MLE])
  - IBM OS PL/I Optimizing Compiler 2.3.0
  - IBM PL/I Optimizing Compiler 1.5.1

Source Audit Assistant supports the following languages in source code form (the reports option only supports seed analysis and template control file generation for COBOL):

- Assembler
- C
- C++
- COBOL
- PL/I

Source Audit Assistant supports the following languages in listing form:

- IBM COBOL for OS/390 & VM 2.1 (plus Millennium Language Extensions [MLE])
- IBM COBOL for MVS & VM 1.2 (plus Millennium Language Extensions [MLE])
- IBM VS COBOL II Release 4.0
- IBM OS/VS COBOL Release 2.4
- IBM PL/I for MVS & VM 1.1.1 (plus Millennium Language Extensions [MLE])
- IBM OS PL/I Optimizing Compiler 2.3.0
- IBM PL/I Optimizing Compiler 1.5.1
- IBM High Level Assembler Version 1 Releases 1, 2, and 3
- IBM Assembler H Version 2

CA Resources

The following system resources are required by CA.

Setup

CA provides two programs used during the setup process:

- SETUP creates the breakpoint data file (BRKTAB). It requires the following resources:

Description	Requirements
Disk space needed for the breakpoint table (BRKTAB) upon completion of SETUP	Determine the size of the BRKTAB data set in bytes using the following formula: 128 + (64 + PA name length) × number of PAs + 32 × number of breakpoints[45] [46]
Disk space needed for the breakout table (BRKOUT) upon completion of SETUP	96 + (number of breakpoints ÷ 8)

- ZAPTXT modifies user object modules to insert breakpoints. It requires the following resources:

Description	Requirements
Storage allocated while ZAPTXT is running	Size of object module
Disk space needed for modified object module on completion of ZAPTXT	Same size as unmodified object module

[45] For COBOL, a PA is a paragraph; for PL/I, a PA is a procedure; for assembler, a PA is a listing.

[46] For COBOL and PL/I, there is approximately one breakpoint per high-level executable statement. For assembler, there are approximately two breakpoints per assembler instruction that change program flow (that is, branch instructions).

Monitor ECSA, SQA, and ESQA Usage

Because the monitor must trace programs in any address space, it is loaded in ECSA and SQA space and uses ESQA storage for tables. The monitor uses the following system storage when it is installed:

Note: Each of these values is approximate.

- Fixed amount when monitor is installed/enabled:

  ECSA 10344 bytes
  SQA 13232 bytes

- Each session started uses the following storage in ESQA:
  - 96 bytes per program area (PA)
  - 52 bytes per breakpoint (BP)

  Note: Each of the following is a PA:
  - COBOL paragraph
  - PL/I procedure
  - Assembler object module

Breakpoints are placed at the start of each high-level instruction for COBOL and PL/I and at conditional branches in high-level statements that can cause a change of program flow (IF, DO WHILE, PERFORM, and others).

When variables are monitored for UTA/DA, about 132K of additional ESQA storage is used.

UTA/DA may have a few extra BPs per location where a variable is read.

If you want an exact count of BPs for any given session, count the number of VERIFY or REPLACE statements in the RPTMSGS DDs in the SETUP job, or issue a CASTATS command for a running monitor session and look at the BPS TOTALS column.

If you want an exact count of 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.

Reports

CA provides two reports programs:

- SUMMARY produces a summary report.
- REPORT produces an annotated listing.

The reports programs require the following resources:

Description	Requirements
Storage needed by breakpoint table while SUMMARY is running	24 bytes per breakpoint
Disk space needed for the summary report on completion of SUMMARY	Negligible
Storage size of REPORT	22KB
Storage needed by breakpoint table while high-level report program is running	40 bytes per breakpoint
Storage needed by breakpoint table while assembler report program (REPORT) is running	16 bytes per breakpoint
Disk space needed for the annotated listing on completion of REPORT program	Size of listing

Data Set Attributes

CA uses several data sets, whose requirements are as follows:

DDNAME	LRECL	BLKSIZE	RECFM
BRKOUT	256	4096	FB
BRKTAB	256	4096	FB
CACTL	255	27998[48]	VB
LISTINB[47]	133	27930[48]	FBA
LISTINP	125	27998[48]	VBA
LISTINA	133 for the High Level Assembler	27930[48]	FBM
	121 for Assembler H	27951[48]	
LISTOUT	133	27930[48]	FBA
SUM	133	27930[48]	FBA

DDNAMEs

FILEDEF	Description
BRKOUT	Contains CA test case results
BRKTAB	Contains all breakpoint data used by the monitor program
CACTL	Control file used for creation of CA runtime environment
LISTINB	Input listing from COBOL
LISTINP	Input listing from PL/I
LISTINA	Input listing from ASM
LISTOUT	Output from REPORT program (annotated listings)
SUM	Output from SUMMARY program

[47] LRECL=121, BLKSIZE=12100 (or smaller) for OS/VS COBOL listings.

[48] Any valid BLKSIZE can be used.

DA and UTA Resources

The following system resources are required by DA and UTA.

Setup

DA and UTA provide three programs used during the setup process:

- SETUP creates the breakpoint data file (BRKTAB). It requires the following resources:

Description	Requirements
Disk space needed for the breakpoint table (BRKTAB) upon completion of SETUP	Determine the size of the BRKTAB data set in bytes using the following formula: 128 + (64 + PA name length) × number of PAs + 32 × number of breakpoints[45] [46]
Disk space needed for the variable table (VARCTL) upon completion of SETUP	64 bytes for each occurrence of a monitored variable in a statement
Disk space needed for the breakout table (BRKOUT) upon completion of SETUP	96 + (number of breakpoints ÷ 8)

- ZAPTXT modifies user object modules to insert breakpoints. It requires the following resources:

Description	Requirements
Storage allocated while ZAPTXT is running	Size of object module
Disk space needed for modified object module upon completion of ZAPTXT	Same size as unmodified object module

- VAREAD creates the variable data file (DBGTAB). It requires the following resources:

Description	Requirements
Disk space needed for the debug table (DBGTAB) upon completion of VAREAD	128 bytes + file name lengths + 56 bytes per variable + variable name length + 16 bytes per statement record

Monitor ECSA, SQA, and ESQA Usage

This information is identical to the information in the Coverage Assistant topic “Monitor ECSA, SQA, and ESQA Usage” on page 386.

Data Set Attributes

DDNAME	LRECL	BLKSIZE	RECFM
BRKOUT	256	4096	FB
BRKTAB	256	4096	FB
CACTL	255	27998[48]	VB
LISTINB[47]	133	27930[48]	FBA
DBGTAB	128	4096	FB
VARCTL	64	27968[48]	FB
VARTAB	128	4096	FB
VARID	255	27998[48]	VB
VARDATA	255	27998[48]	VB

DDNAMEs

FILEDEF	Description
BRKOUT	Contains DA and UTA test case results
BRKTAB	Contains all breakpoint data used by the monitor program
CACTL	Control file used for creation of DA and UTA runtime environment
LISTINB	Input listing from COBOL
LISTINP	Input listing from PL/I
DBGTAB	Variable read information and associated statement numbers
VARCTL	Variable location information
VARTAB	Variable data read during execution
VARID	A table of data on the monitored variables defined in the control file. Also known as a Monitored Variables Report (MVR).
VARDATA	A table of data for each time a monitored variable was read during program execution. Also known as a Variable Data Report (VDR).

SAA Resources

SAA requires the Interactive System Productivity Facility (ISPF).

Appendix C. DBCS Support

This appendix describes ATC *DBCS* (*double-byte character set*)[49] support. ATC support for DBCS varies among the tools as follows.

DBCS Requirements for ATC Compilers and Assemblers

The compilers and assemblers supported by ATC 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. In identifiers, all lowercase DBCS EBCDIC[50] (0x42 in the first byte) will be converted to uppercase.
4. If an identifier is all DBCS EBCDIC, it will be converted to its SBCS (single-byte character set) equivalent.
5. If an identifier contains one or more non-EBCDIC DBCS characters, the whole identifier will be converted to its DBCS representation.
6. Any DBCS, SBCS, or mixed DBCS and SBCS identifiers that convert to the same identifier by the previous rules are considered equivalent.
7. The DBCS EBCDIC form of a character is only allowed in an identifier name if the SBCS version is allowed.
8. Each compiler/assembler has an option that tells the tool whether DBCS is to be recognized as such (also referred to as *enabled*).

   Notes:

   a. In the previous list, *identifier* only pertains to COBOL and PL/I.
   b. PL/I allows DBCS EBCDIC characters in keywords.

CA/DA/UTA DBCS Support

DBCS support is implemented in the following way by CA/DA/UTA:

- The CA/DA/UTA and targeted summary control files will accept DBCS identifiers and DBCS strings within comments.
- CA/DA/UTA will provide DBCS support that is consistent with support provided by the compilers, except that the DBCS space will not be supported within the control files except in comments.
- DBCS in control cards is always enabled.

---

[49] 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.

[50] Extended binary-coded decimal interchange code. A coded character set of 256 8-bit characters.

© Copyright IBM Corp. 1997, 1999

- DBCS identifiers will be normalized according to the above rules before any comparisons are performed.
- DBCS identifiers in all outputs will be 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 allowed.

SAA DBCS Support

DBCS support is implemented in the following way by SAA:

- SAA will accept DBCS identifiers, DBCS strings, and DBCS comments within input source files and listings for the supported COBOL compilers.
- SAA will accept DBCS strings and DBCS comments within input source files and listings for the supported assemblers.
- SAA will accept DBCS identifiers, DBCS keywords, DBCS strings, and DBCS comments within the input source files and listings for the supported PL/I compilers **only** within the general compare function and the Reformatted line filter function. The Comments and Declares filters produce unpredictable results if the input files contain DBCS characters.
- If the input source contains DBCS characters, the DBCS support `Enable` field must be set to `Y`.
- SAA will provide DBCS support that is consistent with support provided by the compilers/assemblers, except that there will be no character conversions (rules 3, 4, and 5 under "DBCS Requirements for ATC Compilers and Assemblers" on page 391).
- All control information and MVS data set names must be entered in SBCS.
- The COBOL LANGUAGE(JAPANESE) compiler option is allowed for COBOL listing input.

SAA Postprocessor DBCS Support

DBCS support is implemented in the following way by the SAA Postprocessor:

- The SAA postprocessor will accept DBCS identifiers (COBOL and PL/I), DBCS keywords (PL/I), DBCS strings, and DBCS comments within input source lines in the compare files.
- The seed list can contain DBCS entries.
- The SAA postprocessor will provide DBCS support consistent with that provided by the compilers.
- DBCS input identifiers and keywords will be normalized according to the above rules before any comparisons are done.
- DBCS identifiers and keywords in the output files will be in the normalized form.
- If the input source contains DBCS characters, the DBCS support `Enable` field must be set to `Y`.
- All control information and MVS data set names must be entered in SBCS.

Glossary and Index

© Copyright IBM Corp. 1997, 1999

Glossary

This glossary defines terminology and acronyms unique to this document or not commonly known.

A

annotated listing. Compiler or assembler listing that contains Coverage Assistant information about the execution.

annotated listing coverage 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. Breakpoint.

breakpoint (BP). The practice of replacing an instruction op code with a user SVC instruction so that the ATC 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) created during Coverage Assistant Execution and used during Coverage Assistant Report.

BRKTAB. The DDNAME of the file of breakpoint data (breakpoint table) created during Coverage Assistant Setup and used during Coverage Assistant Execution.

C

CABPDSP. An execution monitor command that displays the status of breakpoints.

CACTL. The DDNAME of the Coverage Assistant, Distillation Assistant, and Unit Test Assistant control file.

CADATA. An execution monitor command that writes the coverage statistics (BRKOUT) to the file name specified on the panel.

CAIDADD. An execution monitor command that allows you to add a unique test case ID.

CALIST. An execution monitor command that allows you to select listings for which to display statistics.

CAQUIT. An execution monitor command that functions like the CASTOP command. However, unlike the CASTOP command, CAQUIT does not write output to disk. Contrast with *CASTOP.*

CARESET. An execution monitor command that resets all statistics in the current monitor session to zero.

CASESSN. An execution monitor command that displays a list of the current active sessions.

CASTATS. An execution monitor command that allows you to select the session ID, listing number, and program areas (PAs) for which you want statistics displayed.

CASTOP. An execution monitor command that writes current statistics to disk and terminates the monitor session. Contrast with *CAQUIT.*

change validation report. A report that allows you to verify changes found in the Source Audit Assistant comparison report against your seed list in order to make sure that only planned changes were made and that all seed variables were changed.

code coverage. A measurement of the number of code statements that have been executed.

combined variable data report (CVDR). A table of data that is read during execution. A CVDR includes references to the monitored variable report (MVR) for variable identification and a fully-qualified variable name for each entry.

comparison report. A Source Audit Assistant report that allows you to see differences between original source data and changed source data and helps you to identify items that need closer examination.

compile unit (CU). The programs contained within one compiler listing.

control file. A file that contains information describing the compile units to be analyzed, the file that is to be monitored, and the values of the variables to be recorded. Coverage Assistant, Distillation Assistant, and Unit Test Assistant share the same control file.

CU. Compile unit.

CVDR. Combined variable data report.

© Copyright IBM Corp. 1997, 1999

D

dynamic data warping. The process of changing predefined variable values during runtime. Dynamic data warping might be used to age, or warp, dates in input data files, for example. Contrast with *file warping*.

DBCS. Double-byte character set.

DBGTAB. Debug table. The DDNAME of a file generated by the Setup step when Distillation Assistant or Unit Test Assistant is enabled. The DBGTAB is used during the Distillation Assistant Logical Distillation step and the Unit Test Assistant Report step. For a standard coverage run, this file contains no useful data and its DD card is coded DD DUMMY.

ddname. The symbolic representation for a name placed in the name field of a DD statement.

distillation. The reduction of a data set to the minimum size that provides the same test coverage as the complete data set.

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.

Execute. The Coverage Assistant step that monitors your program while it is being executed to collect test case coverage statistics.

F

file warping. The process of statically modifying predefined variables in copies of VSAM or QSAM input files to simulate input conditions for testing. File warping might be used to clear fields in test copies of production input files for privacy or security reasons. Contrast with *dynamic data warping*.

foreground execution. The execution of a computer program that preempts the use of computer facilities.

I

input master data set. A data set to be distilled into an output master data set by physical distillation.

J

jcldsn. JCL data set name.

K

key. One or more characters used to identify the record and establish the order of the record within an indexed file.

key list. A list of characters used to identify the record and establish the order of the record within an indexed file.

L

LISTINA. DDNAME of the Assembler H or High Level Assembler listing file used in Setup or Reports.

LISTINB. DDNAME of the COBOL assembler listing file used in Setup or Reports.

LISTINP. DDNAME of the PL/I assembler listing file used in Setup or Reports.

logical distillation. Instrumenting your object code and executing the instrumented code under the Distillation Assistant monitor. As the instrumented code reads records from the specified input master data set, the monitor determines which keys in the input master data set caused new code coverage in the instrumented code. The list of these keys is then saved for the physical distillation step.

logical key. As used in reference to distillation, a logical key is simply a field within a data record that can be used to identify the record. This field may, or may not, be identified as a physical key to the file system or database manager involved in actually reading the record.

Multiple records can have the same logical key. In this case, it is assumed that all records with this key are required to obtain the necessary code coverage.

M

Millennium Language Extensions (MLE). A compiler-assisted solution for the Year 2000 problem. Available for IBM's COBOL and PL/I compilers.

MLE. Millennium Language Extensions.

monitor. The program (MONSVC) that measures test case coverage during execution of your programs.

monitor session. A distinct invocation of the monitor program.

monitored variable report (MVR). A table of monitored variables and their read specifications.

MVR. Monitored variable report.

N

new source file. A data set you want Source Audit Assistant to compare. Typically, you want to compare a modified data set (new source file) with an original data set (old source file).

O

old source file. A data set you want Source Audit Assistant to compare. Typically, you want to compare an original data set (old source file) with a modified data set (new source file).

Op code. Operation code. A code for representing the operation parts of the machine instructions of a computer.

P

PA. Program area.

physical distillation. This step consists of creating a new master data set by reading the list of keys produced in the first step (logical distillation) and the input master data set. The new master data set consists of only those records in the input master data set whose logical key appears in the list of keys.

Program area (PA). Each specific PA contains all of the breakpoints for one COBOL paragraph, PL/I block, or assembler listing.

Q

QSAM. Queued sequential access method.

Queued sequential access method (QSAM). An extended version of the basic sequential access method (BSAM). When this method is used, a queue is formed of input data blocks that are awaiting processing or of output data blocks that have been processed and are awaiting transfer to auxiliary storage or to an output device.

R

reentrant program. A computer program that may be entered at any time before any prior execution of the program has been completed.

Report. The Coverage Assistant step that produces the summary and annotated listing reports after a test case run.

S

SAA postprocessor. Generates a change validation report or a prototype Coverage Assistant target control file using a compare file generated by a previous Source Audit Assistant run and a list of seed variables that you want to monitor.

SBCS. Single-byte character set.

seed list. A list of seed variables.

seed variable. A variable name used as input to the Source Audit Assistant postprocessor. This is generally a variable name that was identified as one needing to be changed.

session. A distinct invocation of the monitor program.

session ID. The identification of your session to the monitor program. This defaults to your TSO user ID.

SETUP. The Coverage Assistant program 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.*

site defaults. Default settings in *hi_lev_qual*.V1R5M0.MASTER.DEFAULTS. These defaults are used by all individuals at a particular location and are typically set by the person or persons who install ATC. Contrast with *user defaults*.

SQA. System queue area.

System queue area (SQA). An area of MVS storage used for running authorized programs or for storage allocation.

summary coverage report. A Coverage Assistant report that gives statistics on the coverage of all program areas (PAs) during the test run.

summary report. A Coverage Assistant report that provides the summary statistics for PAs.

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. Supervisor call.

T

targeted summary. A measurement of the number of specific code statements and/or variables that have been executed.

targeted summary report. A report that allows you to target certain statements and/or COBOL or PL/I variables. The format of a targeted summary report is identical to the format of a summary report, except that the content is restricted to statements that you specify. (You can specify a statement number, or you can specify all statements that reference specific COBOL or PL/I variables.)

U

user defaults. Default settings that only affect your personal ATC sessions. You can change these defaults using the ATC panels. Contrast with *site defaults*.

unit testing. A method of capturing and logging values assigned to selected variables in your application program at selected points during their execution.

V

VARCTL. A file used by Distillation Assistant and Unit Test Assistant that lists variable locations where the input data set is read by the monitor.

variable data report (VDR). A table of data that is read during execution with references to the monitored variable report (MVR) for variable identification.

VDR. Variable data report.

VSAM. Virtual storage access method.

Virtual storage access method (VSAM). An IBM licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability.

W

warping. The process of statically (file warping) or dynamically (dynamic data warping) modifying predefined variables to simulate input conditions for testing. See *dynamic data warping* and *file warping*.

Z

ZAPTXT. A program that uses breakpoint data to modify object modules by inserting breakpoints.

Index

A

abend errors 307, 380
age, data 161
annotated listings
  creating JCL for an annotated listing 57, 66, 79
  creating specific 101
  displaying execution counts 102
    debug mode 102
    frequency count mode 102
  reducing size 101
annotation symbols
  changes with Performance Mode 104
ARTT 3
assembler options required 233
ATC defaults, modifying 27
authorized data sets 16, 275
Automated Regression Testing Tool 3

B

background execution, SAA 283
branch, conditional 55
  breakpoints 258, 269
  coverage 83, 87, 89, 103, 141
  instruction 55, 65, 76, 95, 259
breakpoint data 116
breakpoints
  definition 42, 130, 165
  displaying status using CABPDSP 256
  errors executing 275
  purpose in execution step 42, 130, 165
BRKOUT
  definition 42
  specifying file name with CADATA command 259
BRKTAB
  creating 236
  definition 42
buffer monitor 254

C

C/370 access 36
CACTL 81, 145, 187, 209
  assembler
    editing the control file 71
    sample 72
  COBOL
    editing the control file 49, 81
    sample 50, 72, 138, 173, 180
  PL/I
    editing the control file 61, 71
    sample 62
change validation reports (SAA)
  description 301
  inputs 301
  outputs 302
CLASS table full error 379
coder, definition 115
Combine edit screen 119
Combine JCL, creating 117
Combine panel 117
combined variable data report 193
commands
  issuing 255
  monitor 256
  parameters 255—271
comment lines 291, 295
common CA, DA, and UTA information
  commands
    CABPDSP 256
    CADATA 259
    CAIDADD 261
    CALIST 262
    CAPRFOFF 263
    CAPRFON 264
    CAQUIT 265
    CARESET 266
    CASESSN 267
    CASTATS 268
    CASTOP 270
    CAVADSP 272
    issuing commands 255
  control file 209—229
  monitor
    commands 255—271
    execution 245—254
    problems 275
  setup 231—243
comparison range 292, 295, 299
comparison report 299
compiler options
  CA 231
  CA targeted coverage 234
  DA and UTA 234
compilers supported 7, 39, 125, 161, 383, 384
conditional branch 55
  breakpoints 258, 269
  coverage 83, 87, 89, 103, 141
  instruction 55, 65, 76, 95, 259
conditional branch coverage
  overhead 253
  poor performance when measuring 276
  suppressing with Performance Mode 87

© Copyright IBM Corp. 1997, 1999

control file 81, 145, 187, 209
assembler
editing the control file 71
COBOL
editing the control file 49, 137, 172, 179
sample 50, 138, 173, 180
common CA, DA, and UTA 209
contents 211
statement syntax 212
PL/I
editing the control file 61, 71
sample 62, 72, 147
warp 197, 200—206
control statements error 379, 381
conversion, data 3
coverage
conditional branch, measuring 276
overhead 253
suppressing conditional branch 87
Coverage Assistant (CA)
compiler options 231, 234
compilers supported 7, 39
control file 81—82
description 39
execution 42
files 13
flow diagram 41
languages supported 39
large project environments 115—121
overview 40
process overview 39
report differences with DA or UTA 103
reports 43, 83—113
requirements and resources 383
samples 45—80
terminating 265, 270
customizing ATC
running sample test cases 36

D

DA setup 130
data conversion 3
data set, site defaults 20
data sets 13
attributes 387, 389
installation 13
reserved, SAA 303
data warping
conversion tables 226
statement 224
DBCS support 391
debug mode 30, 102, 242
declaration statements 291, 295
defaults file, creating and modifying 27
defaults, site 20
defaults, user 27
diagnosing problems
See also error messages
047 abend 275
0C1 abend on user program 275
conditional branch coverage, poor performance 276
SQA storage depleted 276
distillation
logical
description 125
outputs 150
requirements 127
physical
DASD data 152
description 125
editing JCL 156
generating JCL 152
parameters 150
process 149
return codes 158
running 151
submitting JCL 157
tape data 154
Distillation Assistant (DA)
CA report differences when enabled 103
compiler options 234
compilers supported 7, 125
control file 145—147
description 125
execution 130
flow diagram 129
languages supported 125
logical distillation 125, 127
panel 151
physical distillation 149—158
process overview 127
requirements and resources 383
samples 133—143
setup 130
distillation, logical 127

E

editing the control file (CACTL)
embedded statements 378
error messages
description 377
list 379—381
error messages and return codes
See also troubleshooting
ARCOxxxx 309
ARSUxxxx 310
ASETxxxx 311
CMBxxxx 312
CMD5xxxx 314

error messages and return codes *(continued)*
COMNxxxx 319
DFLTxxxx 326
KPDSxxxx 334
N2Oxxxxx 335
OCUSxxxx 340
PVRxxxx 343
RKEYxxxx 346
RPT03xx 347
SAACxxxx 350
SAAFxxxx 353
SAARxxxx 355
SP601xx 359
SUM0xxxx 362
VAR0xxx 364
WVAxxxx 373
ZAP00yy or ZAP88xx 375
errors
abend 380
CLASS table full 379
control statements 379, 381
find 379, 381
library 380
macro 381
margin 379, 380
message limit 381
OBJECT table full 379
OPEN 379
parsing 380
preprocessor 381
READ 379
recovery 380
retry 380
scan 379
sequential libraries 379
severe 377
STAE 380, 381
storage 380
table overflow 380
terminating 377, 378
unrecognized character 380
warning 377, 378
ESQA usage 386, 389
execution
CA overview 42
DA overview 130
monitor 245
multiple user sessions 249
UTA overview 165

F

file
input 289, 293, 296
installation 389
output 291, 294
file warp
control file 197, 200—206
JCL 199
return codes 206
samples 199
filters, SAA
comment lines 291, 295
declarations 291, 295
reformatted lines 291, 295
find error 379, 381
foreground execution, SAA 285
frequency count mode 30, 102, 143, 242

H

help, getting xx
high-level qualifier (*hi_lev_qual*) 15

I

information message 377, 378
input data sets 141
input master data set 126
inputs, SAA 289, 293, 296, 301
insertions 300
installation
See system installation
introducing
Coverage Assistant (CA) 39
Distillation Assistant (DA) 125
Source Audit Assistant (SAA) 279
Unit Test Assistant (UTA) 161

J

JCL
CA
creating combine 117
creating report 91
for assembler examples 68—80
for COBOL examples 47—59
for PL/I examples 59—68
CA/DA/UTA Setup
creating monitor 246
creating Setup 239
creating Setup for compile job stream 241
DA
editing distillation 156
for COBOL and PL/I examples 135—140
generating for physical distillation 152
submitting for physical distillation 157
file warp 199
UTA
creating report 194
for COBOL examples 169—182

L

Language Environment runtime library 36
language, source 291, 294
languages supported 8, 39, 125, 161
large project environments 115
libraries, ATC 13
  modifying FORMS and REXX 15
library error 380
listings
  assembler 75
    example 95
  COBOL 54, 279, 291, 294, 352
    example 55, 95
  PL/I 64
    example 65, 95
log file (SAA)
  embedded statements, restrictions 378
  message format description 377
  message list 379
logical distillation, requirements 127

M

macro 378
macro error 381
margins, source 379, 380
menu
  primary option 282
  Source Audit Assistant 289
message limit error 381
messages, displaying
  See also error messages
  reports 312
  setup 239
messages, error 377
Millennium Language Extensions
  definition 397
  support 7
MLE
  See Millennium Language Extensions
module, definition 115
monitor program 42, 130, 165, 245
  commands 255
  problems, diagnosing 275
  SQA and ESQA usage 386
monitored variables report 189
multiple user sessions 245

O

OBJECT table full error 379
operating systems
  CA 40
  DA 126
  SAA 279
  UTA 162
options, compiler
  CA 231
  CA targeted coverage 234
  DA and UTA 234
output
  areas 299
  comparison report 299
  examples 299
  explanation 299
overhead 253
overview, ATC 3

P

PA
  See program area
panel interface 40, 126
panels
  ATC Defaults 29
  ATC Primary Option Menu 27
  CA
    Coverage Reports 90
    Create JCL for Combining Multiple Runs 117
    Create JCL for Summary and Annotation Report 93
    Create JCL for Summary Report 91
    Create JCL for Targeted Summary Report 112
  Common CA, DA, and UTA
    Control the CA/DA/UTA Monitor 256
    Coverage, Distillation and Unit Test Assistant 209
    Create JCL for Setup 239
    Create JCL to Start the Monitor 246
    Monitor: Add ID 261
    Monitor: Display Breakpoint Status 256
    Monitor: Display Statistics 268
    Monitor: Quit Monitor 265
    Monitor: Reset All Data in Monitor 266
    Monitor: Stop Monitor 270
    Monitor: Take Snapshot of Data 259
    Work with the CA/DA/UTA Control File 209
  DA
    Edit Distillation JCL 156
    Generate JCL to Generate Key List and Distill DASD Data 152
    Generate JCL to Generate Key List and Distill Data 151
    Generate JCL to Generate Key List and Distill Tape Data 154
    Submit Distillation JCL 157
  Manipulate ATC Defaults 27
  Reset Defaults to System Defaults 32
  SAA
    Execute Source Audit Assistant 289
    Postprocessor 296
    SAA Background Execution Parameters 289
    SAA Foreground Execution Parameters 293

panels *(continued)*
UTA
Unit Test Report 194
parameters
command 255—271
physical distillation 150
PRINTVAR program 196
SETUP and ZAPTXT 241
debug mode 242
frequency count mode 242
parsing error 380
Performance Mode 87, 253
changes in annotation symbols 104
physical distillation 149
postprocessor, SAA 280, 296
panel 288
preprocessor error 381
prerequisites 7
problem diagnosis
See also error messages
047 abend 275
0C1 abend on user program 275
conditional branch coverage, poor performance 276
SQA storage depleted 276
program area
definition 45
displaying statistics for 268

R

range of comparison, SAA 292, 295, 299
READ error 379
recovery error 380
reformatted lines 292, 295
regression testing 3
replacements, SAA 300
REPORT program
errors 312
parameters 196
resources required 386
reports
annotated listings 95
creating specific listings 101
CA with DA or UTA 103
combined variable data 193
monitored variables 189
printing 105
summary coverage 83
summary for assembler 88
summary without annotated listings 94
targeted summary 105
variable data 191
requirements, ATC 383, 391
reserved data sets, SAA 303
resources required
CA 385
resources required *(continued)*
compilers, supported 383
DA and UTA 388
prerequisites 383
SAA 389
restrictions, SAA 303
results 116
combining 116, 120
measuring individual test cases 120
rules used 120
test case 120
retry error 380
return codes
file warp 206
physical distillation 158
runtime library, accessing 36

S

sample
annotated listings
assembler 75
COBOL 54
PL/I 64
CA 45
combining coverage results 120
DA 133, 141
file warp 199
files, list of 13
running 36
SAA 281
summary of test case coverage 47
assembler 68
COBOL 47
PL/I 59
test cases 36, 120
using to test installation 33
UTA 167, 183
scan error 379
scan phase 377
sequential libraries error 379
sessions
definition 249
determining active sessions 267
multiple 249
setup
creating JCL using the panels 239
for summary and listings 231
JCL for the compile job stream 241
overview 42
parameters 241
SETUP 241
ZAPTXT 243
when to create or submit JCL 241
SETUP program
parameters 241

SETUP program *(continued)*
resources required 385, 388
severe error 377
site defaults data set
editing 20
example 22
source
language 291, 294
margins 379
Source Audit Assistant (SAA)
comparison report 299
description 279
error log file 377
installation, verifying 281
languages supported 279
postprocessor 296
prerequisites 8, 384
process overview 280
requirements 279
reserved data set names 303
samples 281—288
starting 289
verification 282
SQA storage depleted 276
SQA usage 386, 389
STAE error 380, 381
starting ATC 27
statements
compiler control 378
control 379, 381
declaration 379
embedded 378
statistics, resetting 266
stopping the monitor 270
storage error 380
SUMMARY program
parameters 104
resources required 386
summary test case coverage
assembler
create monitor JCL 73
create summary JCL 73
execute the JCL 74
COBOL
create JCL to link the modified object modules 52
create JCL to run the GO step 52
create JCL to start a monitor session 51
create setup JCL 50
create summary JCL 52
editing the control file 49
execute the JCL 53
summary 47
PL/I
create JCL to link the modified object modules 63, 74
create JCL to run the GO step 63, 74
summary test case coverage *(continued)*
PL/I *(continued)*
create monitor JCL 62
create setup JCL 62
create summary JCL 63
editing the control file 61, 71
execute the JCL 64
summary 59, 68
supervisor call (SVC)
definition 398
installing and enabling 18
problems, diagnosing 275
used as breakpoints 162, 245
support, technical xx
SVC
definition 398
installing and enabling 18
problems, diagnosing 275
used as breakpoints 162, 245
symbols, annotation
changes with Performance Mode 104
syntax
conversion tables 226
statements and options
ASM 219
COBOL 215
compilation unit definition (COBOL or PL/I) 107
COVERAGE 223, 224
defaults 213
examples 228
FILE Read 227
INCLUDE 109, 213
PL/I 217
reading xviii
SCOPE 107, 220
TARGETSTMT 110
TARGETVAR 109
VARIABLE 222
WARP 224
system considerations
CA 40
DA 126
SAA 279
UTA 162
system installation
abends 307
allocating data sets 33
installing data sets 13
modifying the FORMS and REXX libraries 15
setting up the authorized data set 16
site defaults file, editing 27

T

table overflow error 380

target control file 106
technical support xx
terminating ATC
  CAQUIT command 265
  CASTOP command 270
terminating error, SAA 377, 378
test cases
  combining results 116
  individual, measuring coverage 121
  multiple compile unit 176
  rules used 116
  running samples 36
  sample 25
  specifying ID 261
tester, definition 115
testing, regression 3
timeout, monitor 266
troubleshooting 377

U

unexecuted 88
Unit Test Assistant (UTA)
  CA report differences when enabled 103
  compiler options 234
  compilers supported 161
  control file, editing 187—188
  description 161
  execution 165
  flow diagram 164
  languages supported 161
  prerequisites 7
  process overview 162
  reports 166, 189—196
  requirements 162
  samples 167—184
  setup 165
unrecognized character error 380
user defaults
  editing 28
  panel 29
  resetting 32
user SVCs 42, 130, 165

V

variable data report 191
variable read operation
  where the variable is read 183
variable warp operation
verification of installation 25

W

warning error 377, 378
warping
  control file 197, 200—206
  conversion tables 226
  dynamic data 161
  file 197
  JCL 199
  return codes 206
  samples 199
  statement 224

Z

ZAPTXT program
  errors 375
  parameters 243
  resources required 385, 388