z/OS
Version 2 Release 3

ISPF Software Configuration and Library Manager (SCLM) Guide and Reference

SC19-3625-30

Note

Before using this information and the product it supports, read the information in "Notices" on page 647.

This edition applies to Version 2 Release 3 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2019-06-21

© Copyright International Business Machines Corporation 1990, 2019.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures ..... **xi**

Tables ..... **xxi**

Preface ..... **xxiii**
Who should use this xxiii
Content xxiii
How to read the syntax diagrams xxiv

z/OS information ..... **xxix**

How to send your comments to IBM ..... **xxxi**
If you have a technical problem xxxi

Summary of changes ..... **xxxiii**
Summary of changes for z/OS Version 2 Release 3 (V2R3) xxxiii
Summary of changes for z/OS Version 2 Release 2 (V2R2) xxxiii
Summary of changes for z/OS Version 2 Release 1 (V2R1) xxxiii

What's in the z/OS V2R3 ISPF library? ..... **xxxv**

Part 1. Project Manager's Guide **1**

Chapter 1. Defining the project environment 3
Running different versions of SCLM in multiple partitions 3
Overview of project manager tasks 3
Project definition data 3
Generating a project environment 4
Step 1: Determine the project's hierarchy 4
Primary non-key group testing techniques 6
Step 2: Identify the types of data to support 8
Step 3: Establish authorization codes 8
Using authorization codes to control SCLM operations 9
Allowing parallel updates 10
Step 4: Allocate the PROJDEFS data sets 11
Step 5: Allocate the project partitioned data sets 12
Data set naming conventions 12
Flexible naming of project partitioned data sets 12
Number of data sets to allocate 13
Versioning partitioned data sets 16
Project partitioned data sets 17
Space considerations 17
Step 6: Allocate and create the control data sets 17
Create the accounting data sets 18
Create the export data sets 20
Create the audit control data sets 20
Create the Cross-dependency data set 22
Create the SCLM control data set 22
Step 7: Protect the project environment 23

PROJDEFS data sets ... 23
Project partitioned data sets ... 23
Control data sets ... 23
Step 8: Create the project definition ... 23
Alternate project definitions ... 24
Create the hierarchy definition ... 25
Set the project control options ... 26
Define the language definitions ... 32
Step 9: Assemble and link the project definition ... 38
Assemble and link example ... 39
Project manager scenario ... 39
Prerequisites for defining an SCLM project ... 39
Example project overview ... 40
Preparing the example project hierarchy ... 42
Understanding the sample project definition ... 46
Preparing the example project data ... 47

Chapter 2. User exits ... 53
Specify the change code verification routine ... 55
Change code verification routine example ... 56
Specify the Build and Promote User Exit routines ... 57
Build and Promote User Exit routine requirements ... 58
Build and Promote User Exit output data sets ... 60
Specify the Audit Version Delete User Exit routine ... 61
Audit Version Delete User Exit routine requirements ... 61
Specify the Delete User Exit routine ... 62
Delete User Exit Routine requirements ... 62
Delete User Exit output data set ... 63
User exit routine example ... 64

Chapter 3. Additional project manager tasks ... 69
Splitting project VSAM data sets ... 69
Backing up and recovering the project environment ... 70
Synchronizing accounting data sets ... 70
Maintaining accounting data sets ... 71
Modifying the Delete from Group dialog interface ... 71
Implementing package backout ... 72

Chapter 4. Converting projects to SCLM ... 75
Prerequisites for existing hierarchies ... 75
Create alternate project definitions ... 75
Create architecture definitions for the project ... 75
Register existing PDS members with SCLM ... 76
Introducing fixes to the converted hierarchy ... 77

Chapter 5. Language definition considerations ... 79
Using multiple translators in a language definition ... 79
Invoking user-defined parsers ... 82
Defining information tracked by SCLM ... 83
Writing the parser ... 83
Telling SCLM how to invoke your parser ... 84
Processing conditionally saved components ... 94
Example of processing conditionally saved components ... 94
Setting up the project definition ... 95
Specifying the locations of included members ... 95
Example ... 96
Dynamic include tracking ... 99
Input list translators ... 100

Configuring the input list translators....101
Defining a new language to SCLM....101
Using DDnames and DDname substitution lists....102
Showing users how to write CC architecture definitions....111
Convert your JCL decks to architecture definitions....112
Defining a preprocessor to SCLM....113
Passing the source to the compiler....114
Converting JCL to SCLM language definitions....118
Before you begin....118
Capabilities and restrictions....118
Converting JCL cards to SCLM macro statements....119

Chapter 6. Using SCLM and Tivoli Information Management for z/OS....127
Required environment....127
Description of user program interaction....127
Input parameters....127
Option list format....127
Information Management parameters....128
SCLM parameters....128
Program flow....129
Error processing....129
Example....130

Chapter 7. Understanding and using the customizable parsers....131
The parsers as provided....131
Sample language definitions....131
Parser error listings....132
Modifying the parsers....132
Adding more elaborate parsing error messages....133
Appending to the error listing file....134
Compiling the parsers....135

Part 2. Developer's Guide....137

Chapter 8. The Software Configuration and Library Manager....139
SCLM project environment....139
User application data....139

Chapter 9. Using SCLM functions....143
Name retrieval with the NRETRIEV command....143
SCLM considerations for NRETRIEV....144
SCLM main menu....145
SCLM main menu options....145
SCLM main menu action bar choices....146
SCLM main menu panel fields....146
View (option 1)....146
SCLM View - Entry Panel action bar choices....147
Edit (option 2)....149
SCLM Edit - Entry Panel fields....150
Comparison of SCLM and ISPF editors....152
SCLM command macros....152
Utilities (option 3)....156
Library Utility....157
Migration Utility....177
Database Contents Utility....179
Architecture Report Utility....188
Export Utility....195

Import Utility ... 199
Audit and Version Utility ... 203
Delete from Group Utility ... 213
Package Backout Utility ... 216
Unit of Work Utility ... 224
SCLM Explorer ... 230
SCLM Search ... 232
Build (option 4) ... 237
Build Report example ... 240
Build Information Example ... 243
Promote (option 5) ... 245
Promote Report ... 248
Processing errors ... 251
Command (option 6) ... 252
Easy Cmds (option 6A) ... 252
Batch Processing ... 252
Output disposition ... 253
Sample Project Utility (option 7) ... 254
Maintaining SCLM administrators (option A) ... 255

Chapter 10. Development scenario ... 257
Understanding the hierarchy and the SCLM main menu ... 257
Understanding the architecture definition ... 258
Sample SCLM development cycle ... 260
Using the SCLM editor ... 262
Understanding the library utility ... 263
Using Build ... 264
Editing the member to correct errors ... 265
Attempting to promote a member before performing a build ... 265
Rebuilding the changed member ... 265
Using the Database Contents Utility ... 266
Promoting a member successfully ... 268
Drawing down a promoted member ... 269
Performing project housekeeping activities ... 269

Chapter 11. Architecture definition ... 271
Architecture members ... 271
Kinds of architecture members ... 271
Defining compiler processed components ... 272
Compilation control architecture members ... 272
Specifying source members ... 273
Defining link-edit processed components ... 273
SCLM build and control timestamps ... 274
Defining application and subapplication components ... 274
Generic architecture members ... 275
Build and promote by change code ... 275
Architecture statements ... 277
Statement format ... 277
Statement uses ... 278
Sample application using architecture definitions ... 283
Ensuring synchronization with architecture definitions ... 287
Build outputs ... 288
Multiple build outputs ... 288
Sequential build outputs ... 288
Default output member names ... 288
Languages of output members ... 288

Part 3. Advanced Topics.....291

Chapter 12. Managing complex projects..... 293
Impact assessment techniques..... 293
Dependency processing.....293
Propagating applications to other databases.....294

Chapter 13. SCLM support for DB2..... 295
Restrictions..... 296
Information for project administrators.....296
The FLMCSPDB DB2 bind/free translator..... 296
Generating a project environment..... 296
Information for developers.....299
Getting started..... 300
Create a program that has SQL statements.....300
Create a generic architecture definition to control the bind..... 301
Create a high-level (HL) architecture definition to link link-edit to bind.....301
Alternative High Level (HL) architecture definition to link link-edit to bind.....301
Other architecture definition considerations.....301
Create DB2 CLIST..... 302
More complex scenarios..... 304
Storing bind options in a bind control file..... 305
Binding on different LPARs.....305
Rebinding at lower levels after a promotion.....305

Chapter 14. SCLM support for workstation builds.....307
Requirements.....307
Overview of workstation build..... 307
Information for the project manager.....309
Naming conventions..... 309
Languages..... 309
What workstation tools will you use?.....310
Workstation information.....310
More information on SCLM, ISPF, and workstation builds..... 310
ISPF Sample and Macro libraries.....311
Information for the developer..... 312
Migrating applications into SCLM.....312
Architecture definition members for workstation applications..... 312
Specifying options..... 313
Including outputs from other build steps..... 313
Running multiple workstation commands..... 313
Sample language definition..... 314
Workstation setup..... 317
Directories and file names.....317
Multiple builds on one workstation..... 318

Chapter 15. Leaving a Member Behind on Promotion.....319
Setting a member as not being promotable.....319
Using the N line command in Library Utilities (option 3.1) or Unit of Work (option 3.11).....320
FLMCMD NOPROM service..... 321
FLMLNK NOPROM service..... 321
Process of not promoting a member (REBUILD).....322
Process of not promoting a member (NOREBUILD)..... 326
SCLM project setup when promoting with no rebuilding of build maps..... 327
Build containing a non-promotable member (NOREBUILD).....327

Promote containing a non-promotable member (NOREBUILD) from the same level containing the NOPROM member....329
Viewing the non-promoted backup member....330
Promote containing a non-promotable member (NOREBUILD) from a level not containing the NOPROM member....330
Build containing a non-promotable member (NOREBUILD) at a level which does not contain the NOPROM member....330
Build after promotion of the non-promotable member (NOREBUILD)....331
Restricting the setting of non-promotable....332

Chapter 16. Member encoding and decoding....333
Setting up encoding and decoding....335
Removing encoding and decoding....335

Chapter 17. SCLM security....337
SCLM internal security....338
Enabling security....338
Determining the type of security to implement....338
Setting up SCLM DSN security....341
Setting up SCLM subproject security....343
Setting up SCLM service security....348
Working with subproject security....349
Migrating members into SCLM subproject security....349
Viewing the subproject members....351
Resolving authority problems....352

Part 4. SCLM Reference....357

Chapter 18. Invoking the SCLM services....359
Invoking the SCLM services....359
Command invocation of the SCLM services....359
The FLMCMD interface....359
Call invocation of the SCLM services....362
The FLMLNK subroutine interface....362
Selecting a service from the FLMCMD Services Menu....365
Entering a command to invoke a specific service panel....366
Types of parameters....366
ISPF variables....372
SCLM service return codes....377
FLMCMD command processor return codes....377
FLMLNK call processor return codes....378
SCLM service messages....378

Chapter 19. SCLM services....379
SCLM service descriptions....379
ACCTINFO—Retrieve Accounting Information....380
AUTHCODE—Retrieve or Set Authorization Code for Selected Members....384
BUILD—Build a Member....388
CCEXITS—Run User Exits without Edit....392
DBACCT—Retrieve Accounting Records for a Member....394
DBUTIL—Generate a Tailored Output Data Set and Report....395
DELETE—Delete Database Components....400
DELGROUP—Delete from Group Database Components....402
DSALLOC—Allocate Data Sets for Group/Type....407
EDIT— Edit a Member of a Controlled Library....410
END— End an SCLM Services Session....413
ENDEC— Encode and Decode members....414

EXPORT—Extract SCLM Accounting Information for a Group....416
FREE—Free an SCLM ID....419
GETBLDMP—Retrieve Build Map Information....420
GETXDEP—return cross-dependency information....423
IMPORT—Import SCLM Accounting Information to Current Project....426
INIT—Generate an SCLM ID....429
LOCK—Lock a Member or Assign an Access Key....430
MIGRATE—Create Accounting for Selected Members....434
NEXTGRP— Retrieve the Next Group in an SCLM Hierarchy....438
NOPROM—Change Promote Processing....441
PARSE—Parse a Member for Statistical and Dependency Information....442
PROMOTE—Promote a Member from One Library to Another....444
RPTARCH—Generate an SCLM Architecture Report....448
SAVE—Lock, Parse, and Store a Member....451
SCLMINFO—Return Project Information....455
START—Generate an Application ID for a Services Session....458
STORE—Store Member Information in an Accounting Record....459
UNLOCK—Unlock a Member in a Development Library....462
VERDEL—Delete Version and Audit Information....464
VERHIST—Retrieve Versioned Member Information....466
VERINFO—Retrieve Version and Audit Information....468
VERRECOV—Recover a Version....471
XDEPUPDT—Update Cross-dependency Information....474

Chapter 20. Sample programs using SCLM services....477
Pascal example....477
Main program FLMSRV1....477
Included member FLMSRV1D....481
Included member FLMSRV1S....483
PL/I example....487

Chapter 21. SCLM macros....491
Notes on using the SCLM macros....491
Using SCLM variables in SCLM macros....492
FLMABEG macro....493
FLMAEND macro....493
FLMAGRP macro....494
FLMALLOC macro....494
FLMALTC macro....510
FLMATVER macro....513
FLMCNTRL macro....516
FLMCPYLB macro....538
FLMGROUP macro....540
FLMINCLS macro....542
FLMLANGL macro....545
FLMLRBLD macro....548
FLMPROJ macro....549
FLMNPROM macro....549
FLMSYSLB macro....551
FLMTCOND macro....552
FLMTOPTS macro....556
FLMTRNSL macro....557
FLMTYPE macro....562

Chapter 22. SCLM translators....565
FLMCSPDB DB2 Bind/Free translator....566
FLMDTLC DTL Processor Build translator....570
FLMLPCBL COBOL Parser....571

FLMLPFRT FORTRAN Parser ... 574
FLMLPGEN General Purpose Parser ... 578
FLMLRASM REXX Assembler Parser ... 583
FLMLRCBL REXX COBOL Parser ... 587
FLMLRCIS MVS C/C++ parser with include set support ... 590
FLMLRC2 C, C++, and Resource file parser for workstation source ... 594
FLMLRC37 REXX C370 Parser ... 597
FLMLRDTL REXX DTL Parser ... 601
FLMLRIPF Script and OS/2 IPF Source Parser ... 601
FLMLSS General Purpose Parser ... 605
FLMLTWST Workstation Build translator ... 608
FLMTBMAP Build Map Print - Build translator ... 619
FLMTMJI Interface to JOVIAL Compiler ... 621
FLMTMMI Interface to DFSUNUB0 (phase 2 of MFSUTL and MFSTEST) ... 621
FLMTMSI Interface to SCRIPT/VS ... 622
FLMTPRE ... 623
FLMTPST ... 625
FLMTXFER Workstation Transfer - Build translator ... 627
SCLM parser restrictions ... 629
Non-explicit references ... 629
Separation of references ... 630

Chapter 23. SCLM Variables and Metavariables ... 631
SCLM variable and metavariable descriptions ... 631
SCLM variable and metavariable tables ... 632
SCLM variable descriptions, variable names, and their SCLM functions ... 633
SCLM variables and their SCLM functions ... 636
SCLM metavariable descriptions, metavariable names, and their SCLM functions ... 639
SCLM metavariable contents ... 639
Description of group variables ... 640

Appendix A. Accessibility ... 643
Accessibility features ... 643
Consult assistive technologies ... 643
Keyboard navigation of the user interface ... 643
Dotted decimal syntax diagrams ... 643

Notices ... 647
Terms and conditions for product documentation ... 648
IBM Online Privacy Statement ... 649
Policy for unsupported hardware ... 649
Minimum supported hardware ... 650
Programming Interface Information ... 650
Trademarks ... 650
Glossary of SCLM Terms ... 651

Index ... 659

Figures

1. Sample syntax diagram....xxv

2. Example of SCLM Hierarchies.... 5

3. Example of SCLM Hierarchies.... 6

4. Default (Primary) Project Hierarchy Structure....7

5. Alternate Project Hierarchy Structure with Primary Non-key Integration Group....7

6. Sample Hierarchy with Authorization Codes.... 9

7. Valid Hierarchy with Unallocated Data Sets.... 15

8. Invalid Hierarchy for Intended Operation.... 16

9. Accounting File Example (Part 1 of 2).... 19

10. Accounting File Example (Part 2 of 2).... 19

11. Audit Control Data Set Example (Part 1 of 2).... 21

12. Audit Control Data Set Example (Part 2 of 2).... 21

13. SCLM Control Data Set Example.... 22

14. Sample Alternate Project Definition.... 25

15. Example Project Definition (Part 1 of 3).... 30

16. Example Project Definition (Part 2 of 3).... 31

17. Example Project Definition (Part 3 of 3).... 32

18. Enterprise COBOL Language Definition Example (Part 1 of 2).... 37

19. Enterprise COBOL Language Definition Example (Part 2 of 2).... 38

20. Example Project Hierarchy.... 40

21. Example Project Architecture.... 42

22. Change Code Verification User Exit.... 57

23. Promote User Exit (Part 1 of 3).... 65

24. Promote User Exit (Part 2 of 3)..... 66

25. Promote User Exit (Part 3 of 3)..... 67

26. JCL to Restore the Primary Accounting Data Set..... 70

27. Sample project definition..... 73

28. COBOL II with DB2 Preprocessor (Part 1 of 2)..... 81

29. COBOL II with DB2 Preprocessor (Part 2 of 2)..... 82

30. SKELS Parser Definition..... 85

31. Parser for ISPF skeletons (Part 1 of 8)..... 86

32. Parser for ISPF skeletons (Part 2 of 8)..... 87

33. Parser for ISPF skeletons (Part 3 of 8)..... 88

34. Parser for ISPF skeletons (Part 4 of 8)..... 89

35. Parser for ISPF skeletons (Part 5 of 8)..... 90

36. Parser for ISPF skeletons (Part 6 of 8)..... 91

37. Parser for ISPF skeletons (Part 7 of 8)..... 92

38. Parser for ISPF skeletons (Part 8 of 8)..... 93

39. LISTINFO Module..... 93

40. STATINFO Module..... 94

41. Sample Language Definition for Conditionally Saved Components..... 94

42. Source member with includes in different include sets..... 96

43. Language definition to support multiple include sets (Part 1 of 3)..... 97

44. Language definition to support multiple include sets (Part 2 of 3)..... 98

45. Language definition to support multiple include sets (Part 3 of 3)..... 99

46. Record Layout Used to Store Dynamic Includes..... 100

47. Finnoga 4 Language Definition (Part 1 of 2)..... 110

48. Finnoga 4 Language Definition (Part 2 of 2)..... 111

49. Panda Universal Preprocessor....114
50. Finnoga/PUPP Language Definition (Part 1 of 2).... 116
51. Finnoga/PUPP Language Definition (Part 2 of 2).... 117
52. Architecture Definition Example.... 117
53. JCL: Execute IEFBR14....119
54. SCLM: Execute IEFBR14....119
55. JCL: Execute GAC....120
56. SCLM Language Definition: Execute GAC....120
57. JCL: Conditional Execution....120
58. SCLM Language Definition: Conditional Execution.... 120
59. JCL: Complex Conditional Execution....120
60. SCLM Language Definition: Complex Conditional Execution....121
61. JCL: Invoke COBOL Preprocessor and Compiler (Part 1 of 2).... 123
62. JCL: Invoke COBOL Preprocessor and Compiler (Part 2 of 2).... 124
63. SCLM Language Definition: Invoke COBOL Preprocessor and Compiler (Part 1 of 2).... 125
64. SCLM Language Definition: Invoke COBOL Preprocessor and Compiler (Part 2 of 2).... 126
65. Sample Project Hierarchy.... 140
66. Key and Non-Key Groups Within the Project Hierarchy....141
67. SCLM Main Menu Panel (FLMDMN).... 145
68. SCLM View - Entry Panel (FLMEB#P).... 147
69. SCLM Edit - Entry Panel (FLMED#P)....150
70. SCLM Edit Profile (FLMEINFO)Edit Profile Panel....155
71. SCLM Utilities (FLMUDU#P)....157
72. SCLM Library Utility (FLMUS#P).... 158
73. Member Selection List (FLMUSL#P)....161

74. Member Selection List with Hierarchy and Member Description View (FLMUSM#P)....162
75. Accounting Record (FLMUSA#P)....164
76. Accounting Record Statistics (FLMUSS#P)....167
77. Change Code List - Records That Can Be Deleted (FLMUSC#P)....168
78. Include List (FLMUSI#P)....169
79. User Data Entries (FLMUSE#P)....170
80. Build Map Record (FLMUSB#P)....171
81. Build Map Contents (FLMUSBRP)....172
82. Authorization Code Update (FLMUSU#P)....174
83. Where-used panel (FLMUSF#P)....175
84. SCLM Not Promoted Member Update panel (FLMUSN#P)....176
85. SCLM Migration Utility (FLMUM#P)....178
86. SCLM Database Contents Utility (FLMRC#P)....180
87. SCLM Database Contents - Additional Selection Criteria (FLMRCA)....181
88. Database Contents Utility Report....183
89. SCLM Database Contents - Customization Parameters (FLMRCT)....184
90. Database Contents Utility Tailored Output....186
91. Change Code Report, Page 2....187
92. Accounting Statistics Report, Page 2....187
93. Source Listing Report....188
94. Cleanup Report....188
95. SCLM Architecture Report (FLMRA#P)....189
96. Architecture report with cutoff of NONE (Part 1 of 3)....191
97. Architecture report with cutoff of NONE (Part 2 of 3)....192
98. Architecture report with cutoff of NONE (Part 3 of 3)....193

99. Architecture report with cutoff of LEC (Part 1 of 2)..... 194
100. Architecture report with cutoff of LEC (Part 2 of 2)..... 195
101. SCLM Export Utility (FLMDXE#P)..... 196
102. Export Report (Part 1 of 2)..... 198
103. Export Report (Part 2 of 2)..... 199
104. SCLM Import Utility (FLMDXI#P)..... 200
105. Import Report (Part 1 of 2)..... 202
106. Import Report (Part 2 of 2)..... 203
107. SCLM Audit and Version Utility (FLMVUS#P)..... 204
108. SCLM Version Selection Panel (FLMVSL#P)..... 205
109. Audit and Version View panel (ISREDDE2) - sample data with history..... 207
110. Audit and Version View panel (ISREDDE2) - sample data..... 208
111. SCLM Audit/Version Record Panel (FLMVBA#P)..... 208
112. SCLM Audit and Version Utility - Compare Panel (FLMVSC#P)..... 210
113. SCLM Audit and Version Utility - External Compare Panel (FLMVSX#P)..... 211
114. SCLM Audit and Version Utility - Retrieve Panel (FLMVSR#P)..... 212
115. SCLM Delete from Group Utility (FLMDDG#P)..... 214
116. Delete Group Report..... 216
117. Package Backout—Backup Phase..... 218
118. Package Backout—Restore Phase..... 220
119. SCLM Package Functions Utility (FLMPF#P)..... 221
120. SCLM Package List Panel (FLMPFL#P)..... 222
121. SCLM Package Member Details Panel (FLMPML#P)..... 223
122. SCLM Unit of Work processing - Entry Panel (FLMUW#P)..... 225
123. SCLM Unit of Work Options Action Bar choices..... 226

124. Set Work Data Set Prefix....227
125. SCLM Unit of Work List Commands panel....228
126. UOW Member List panel....229
127. Work Element List panel....230
128. SCLM Explorer panel (FLMUDEP0)....231
129. SCLM Search Entry panel (FLMUC#P)....233
130. SCLM Search strings panel (FLMDSS#P)....235
131. SCLM Search member list panel....236
132. Example of an SCLM search report....237
133. SCLM Build (FLMB#P)SCLM Build Entry Panel....238
134. Build Report (Part 1 of 2)reportexamplesreportbuild....242
135. Build Report (Part 2 of 2)....242
136. Build Information Reportinformationbuild report....243
137. SCLM Promote (FLMP#P)panelspromotepromote functionpanel....246
138. Promote Report (Part 1 of 3)....249
139. Promote Report (Part 2 of 3)....250
140. Promote Report (Part 3 of 3)....251
141. SCLM Command Shell (FLMTSO)....252
142. Verify Batch Job Information (FLMDSU#P)....253
143. Output Disposition (FLMDEXT)....254
144. Sample Project Hierarchy....257
145. Application FLM01AP1....258
146. Development Cycle....262
147. Valid keywords for architecture member statements....278
148. Application FLM01AP1....284

149. Architecture Members for Application Sample FLM01AP1 (Part 1 of 2)..... 285

150. Architecture Members for Application Sample FLM01AP1 (Part 2 of 2)..... 286

151. Example of Synchronization..... 287

152. DB2CLIST example for plans..... 299

153. DB2CLIST example for packages..... 299

154. Defining DBRMTYPE in DB2CLIST translator..... 299

155. DB2CLIST memberprocessing flowDB2CLIST: flow of processing through the translators..... 300

156. Sample LEC architecture definition..... 301

157. Sample generic architecture definition for bind member..... 301

158. Sample HL architecture definition for overall compilation, link-edit and bind..... 301

159. Sample HL architecture definition for overall compilation, link-edit and bind (no generic architecture definition)..... 301

160. Sample HL architecture definition for overall compilation, link-edit and bind (no generic architecture definition)..... 302

161. DB2CLIST membergeneric exampleDB2CLIST generic example..... 303

162. Specifying Options in a Workstation Architecture Definition..... 313

163. Including Outputs as Inputs..... 313

164. Multiple Workstation Commands..... 314

165. Workstation C Language Definition (Part 1 of 2)..... 315

166. Workstation C Language Definition (Part 2 of 2)..... 316

167. SCLM Not Promoted Member Update panel (FLMUSN#P)..... 320

168. SCLM Not Promoted Member Update panel (FLMUSN#P)..... 324

169. Example of turning off SCLM DSN security for an SCLM project/alternate..... 339

170. Example of turning off SCLM subproject security for an SCLM project/alternate..... 340

171. Example of turning off SCLM service security for an SCLM project/alternate..... 341

172. Example of setting up SCLM DSN security..... 342

173. Example of setting up SCLM subproject security..... 344

174. SCLM Edit Profile panel....350
175. Valid Sub-projects panel....350
176. SCLM Migration Utility entry panel....350
177. SCLM Library Utility entry panel....352
178. Member list panel showing associated subprojects....352
179. XFACILIT SECDBG profile....353
180. SCLM security debug information....353
181. Additional SCLM security debug information....354
182. Sample Interactive Command Session (ISRTSO)....362
183. SCLM FLMCMD Services Menu panel....365
184. $msg_array Contents....368
185. $list_info Contents....372
186. ACCTINFO Service panel....381
187. AUTHCODE Service panel....385
188. BUILD Service panel....389
191. ISPF Interface Panel....392
192. DBUTIL Service panel....397
193. DELETE Service panel....401
194. DELGROUP Service panel....404
195. DSALLOC Service panel....408
196. EDIT Service panel....411
197. EXPORT Service panel....417
198. GETBLDMP Service panel....421
199. IMPORT Service panel....427
200. LOCK Service panel....432

201. MIGRATE Service panel....436
202. NEXTGRP Service panel....439
203. PROMOTE Service panel....445
204. RPTARCH Service panel....449
205. SAVE Service panel....452
206. SCLMINFO Service panel....457
207. UNLOCK Service panel....463
208. VERDEL Service panel....465
209. VERINFO Service panel....469
210. VERRECOV Service panel....472
211. Sample language definition for Assembler....498
212. Sample language definition that calls a preprocessor....503
213. Hierarchy Example for Group Description....641

Tables

1. The steps to generate a project environment....4

2. Authorization Code Allowances....10

3. Versioning Data Set Attributes....16

4. Data Set Attributes....17

5. Language Definitions Supplied with SCLM....32

6. SCLM Macros for Language Definition....35

7. Exits and Exit Routine Specifications....53

8. Initial and Save Change Code Exit Routine Parameters....55

9. User Exit Parameters....59

10. User Exit Output Data Set Format....60

11. User Exit Parameters....61

12. User Exit Parameters....63

13. User Exit Output Data Set Format....64

14. DDname Substitution List Example....102

15. DDnames and KEYREFs....111

16. DDnames Used by a Hypothetical Preprocessor....113

17. Pre-defined entities....244

18. Description of the Elements and Attributes in the Build information XML....244

19. The architecture members....259

20. Uses of Architecture Members....271

21. SCLM System Status Index Field Data....274

22. The conditions under which SLCM builds and promotes by change code....276

23. SCLM Data Set Attributes for DB2 Types....297

24. Language definitions for DB2 298

25. Examples of FLMLTWST combining components 318

26. Summary of methods to secure an SCLM environment 337

27. Subproject access for each SCLM function 345

28. SCLM processes that can be secured 348

29. Pattern Examples 367

30. ISPF variables used in SCLM services 372

31. Macros 491

32. Valid IOTYPEs for each function 504

33. Valid DISP values for IOTYPE values 505

34. Examples of include and nclude-set names derived from source statements 591

35. Examples of include and include-set names derived from source statements 594

36. Examples of dependancies derived from include directives 598

37. Examples of include and include-set names derived from source statements 602

38. SCLM Variable Descriptions, Names, and Their SCLM Functions 633

39. SCLM Variables and Their SCLM Functions 636

40. SCLM Metavariable Descriptions, Names, and Their SCLM Functions 639

41. SCLM Metavariables and Their Corresponding Variables 640

42. SCLM Group Variable List 640

43. SCLM Group Variable Description 641

Preface

This document provides reference and usage information, along with conceptual and functional descriptions of the Software Configuration and Library Manager (SCLM). It also contains step-by-step information for setting up and maintaining an SCLM project environment. It describes how to establish and monitor a database and explains the library functions.

Who should use this

This is for application developers whose projects are controlled by SCLM. This is also for project managers who use SCLM to manage the development process.

All SCLM users should read the first three chapters in Part 2, “Developer's Guide,” on page 137.

Content

This assumes that you are familiar with the operation of ISPF in the z/OS® environment.

Part 1, “Project Manager's Guide,” on page 1 of this document is the **Project Manager's Guide**:

- Chapter 1, “Defining the project environment,” on page 3, describes how to generate a project definition. It explains the steps that enable you to create the database that best meets the needs of your project. It includes step-by-step instructions for setting up the SCLM sample project included with the ISPF product. After completing the steps described here, you can experiment with basic SCLM operations using the sample project hierarchy.

  Chapter 2, “User exits,” on page 53, describes the customization of user exit points so that SCLM can be integrated with other products. It lists the available exit routines and describes how you can customize these for your users.

  Chapter 3, “Additional project manager tasks,” on page 69, describes additional tasks that project managers perform to maintain SCLM projects. It discusses backing up and recovering a project database, using authorization codes to control SCLM operations, developing and maintaining projects concurrently, and implementing verification and exit routines for SCLM projects.

  Chapter 4, “Converting projects to SCLM,” on page 75, describes the steps required to convert existing ISPF software development projects to SCLM.

  Chapter 5, “Language definition considerations,” on page 79 describes setup operations you must perform to create a language definition for SCLM to use. The subsection “Defining a new language to SCLM” on page 101 describes the control structures used to manage SCLM functions and illustrates how to define new languages. It also contains information on converting JCL decks to language definitions.

  Chapter 6, “Using SCLM and Tivoli Information Management for z/OS,” on page 127, illustrates the interaction between SCLM and Information Manager through the use of a sample program.

  Chapter 7, “Understanding and using the customizable parsers,” on page 131, describes the REXX parsers supplied with SCLM and provides examples of how to customize them.

Part 2, “Developer's Guide,” on page 137 of this document is the **Developer's Guide**:

- Chapter 8, “The Software Configuration and Library Manager,” on page 139, provides information on the SCLM project database and the terminology used. It describes the library structure and naming conventions used when you define and maintain SCLM projects.

  Chapter 9, “Using SCLM functions,” on page 143, describes how to use the ISPF dialog interface, select SCLM functions to retrieve or process certain information, and generate reports on the information

© Copyright IBM Corp. 1990, 2019

stored in project databases. It also describes information stored in accounting, cross-reference, and intermediate records for members in the project databases.

Chapter 10, “Development scenario,” on page 257, is a programmer scenario that describes the tasks typically performed by SCLM users. It provides step-by-step instructions on how to use the basic SCLM functions to control development projects.

Chapter 11, “Architecture definition,” on page 271, describes architecture configuration and dependency control statements and their uses. It provides examples of each kind of architecture member and describes the special command statements that the architecture members require. It also provides an example of the format of each statement and lists any restrictions.

Part 3, “Advanced Topics,” on page 291 of this document contains **Advanced Topics**:

- Chapter 12, “Managing complex projects,” on page 293, describes techniques that aid in managing complex configurations.

  Chapter 13, “SCLM support for DB2,” on page 295, describes how to configure SCLM and DB2® to work together.

  Chapter 14, “SCLM support for workstation builds,” on page 307, describes how to set up and use SCLM to do builds on the workstation.

  Chapter 15, “Leaving a Member Behind on Promotion,” on page 319, describes how to leave a member behind during promotion.

  Chapter 16, “Member encoding and decoding,” on page 333, describes how to encode and decode members.

Part 4, “SCLM Reference,” on page 357 of this document is the **SCLM Reference**:

- Chapter 18, “Invoking the SCLM services,” on page 359 introduces services you can use to retrieve and process information that is stored in SCLM project hierarchies. It describes the FLMCMD command processor interface and FLMLNK subroutine call interface, and lists the general categories of parameters, variables, and return codes relevant to invoking SCLM services. It also explains the notation conventions used to document the services.

  Chapter 19, “SCLM services,” on page 379 provides the command and call invocation formats, ISPF interface panel, parameters, and return codes for each service.

  Chapter 20, “Sample programs using SCLM services,” on page 477 provides sample programs in Pascal and PL/I that allow you to invoke SCLM services.

  Chapter 21, “SCLM macros,” on page 491 introduces and describes the macros that are used to create project definitions for SCLM. It also explains the notation conventions used to document the macros.

  Chapter 22, “SCLM translators,” on page 565 describes the translators delivered with SCLM. For each translator, there is a brief description, a list of input parameters, and a list of return codes with the appropriate user and project administrator responses.

  Chapter 23, “SCLM Variables and Metavariables,” on page 631 lists the SCLM variables and identifies each function with which they can be used.

The Glossary of SCLM Terms and the Index sections are available for your reference.

How to read the syntax diagrams

The syntactical structure of commands described in this document is shown by means of syntax diagrams.

Figure 1 on page xxv shows a sample syntax diagram that includes the various notations used to indicate such things as whether:

- An item is a keyword or a variable.
- An item is required or optional.

- A choice is available.
- A default applies if you do not specify a value.
- You can repeat an item.

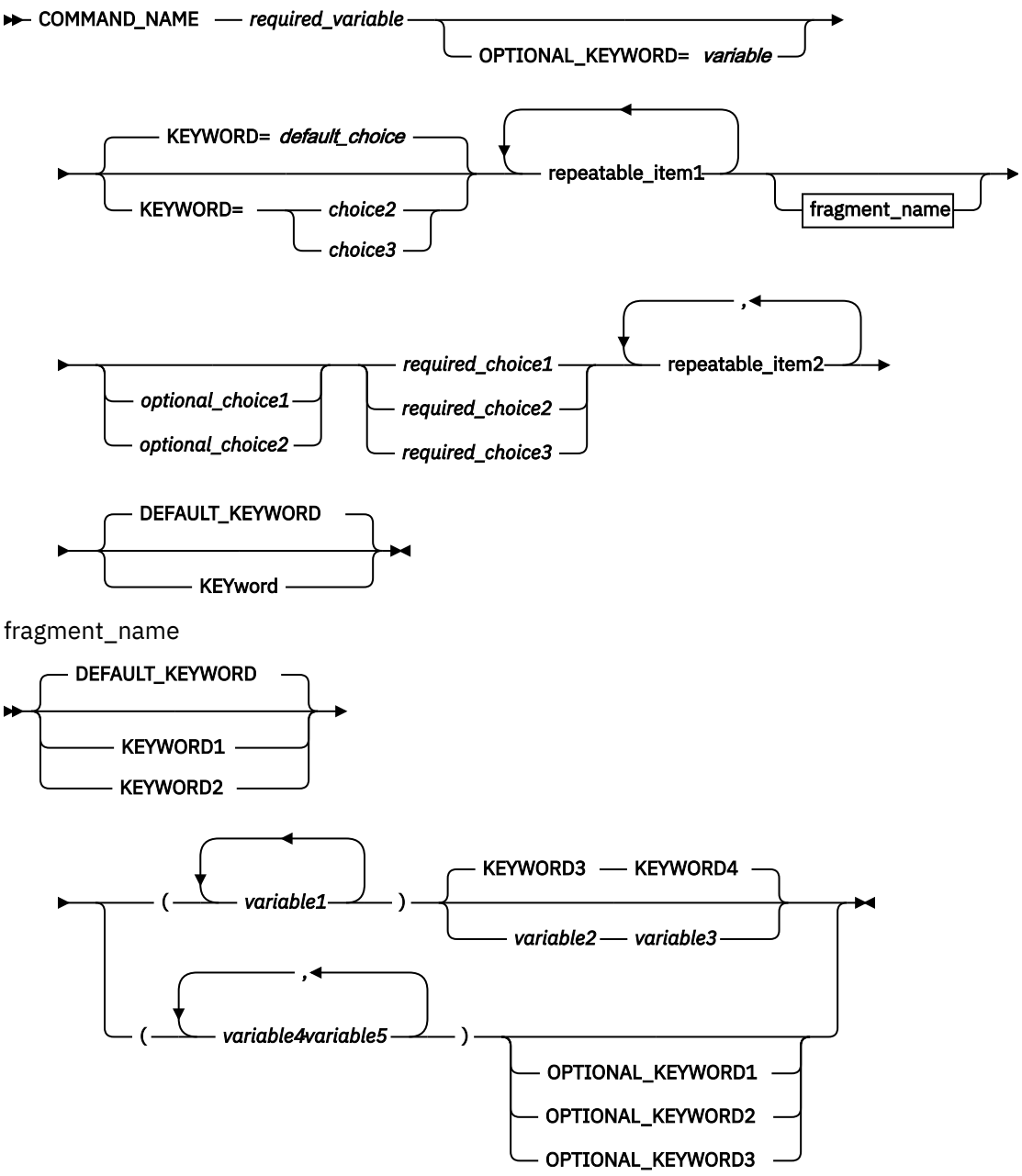


Figure 1. Sample syntax diagram

Here are some tips for reading and understanding syntax diagrams:

Order of reading

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.

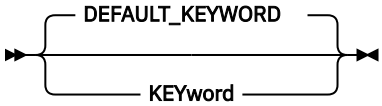
The ——► symbol indicates that a statement is continued on the next line.

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

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

Keywords
Keywords appear in uppercase letters.

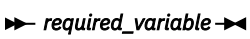
Sometimes you only need to type the first few letters of a keyword, The required part of the keyword appears in uppercase letters.



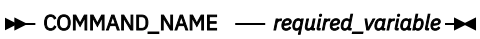
In this example, you could type "KEY", "KEYW", "KEYWO", "KEYWOR" or "KEYWORD".

The abbreviated or whole keyword you enter must be spelled exactly as shown.

Variables
Variables appear in lowercase letters. They represent user-supplied names or values.



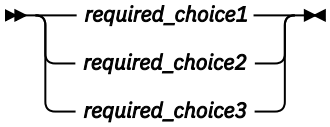
Required items
Required items appear on the horizontal line (the main path).



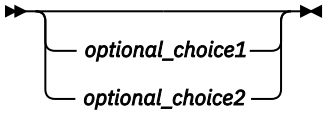
Optional items
Optional items appear below the main path.

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

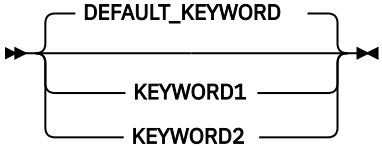
If you *must* choose one of the items, one item of the stack appears on the main path.



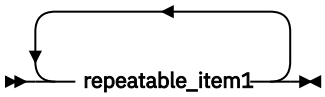
If choosing one of the items is optional, the entire stack appears below the main path.



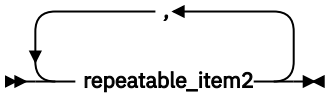
If a default value applies when you do not choose any of the items, the default value appears above the main path.



Repeatable items
An arrow returning to the left above the main line indicates an item that can be repeated.

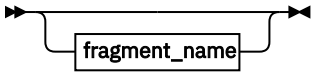


If you need to specify a separator character (such as a comma) between repeatable items, the line with the arrow returning to the left shows the separator character you must specify.



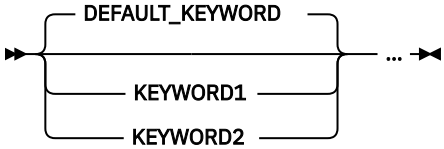
Fragments

Where it makes the syntax diagram easier to read, a section or *fragment* of the syntax is sometimes shown separately.



⋮

fragment_name



z/OS information

This information explains how z/OS references information in other documents and on the web.

When possible, this information uses cross document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

To find the complete z/OS library, go to IBM Knowledge Center (www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

© Copyright IBM Corp. 1990, 2019

How to send your comments to IBM

We invite you to submit comments about the z/OS product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

Important: If your comment regards a technical question or problem, see instead "If you have a technical problem" on page xxxi.

Submit your feedback by using the appropriate method for your type of comment or question:

Feedback on z/OS function
If your comment or question is about z/OS itself, submit a request through the IBM RFE Community (www.ibm.com/developerworks/rfe/).

Feedback on IBM® Knowledge Center function
If your comment or question is about the IBM Knowledge Center functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Knowledge Center Support at ibmkc@us.ibm.com.

Feedback on the z/OS product documentation and content
If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to mhvrcfs@us.ibm.com. We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

To help us better process your submission, include the following information:

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS ISPF SCLM Guide and Reference, SC19-3625-30
- The section title of the specific information to which your comment relates
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the IBM Support Portal (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.

© Copyright IBM Corp. 1990, 2019

Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Summary of changes for z/OS Version 2 Release 3 (V2R3)

The following changes are made for z/OS Version 2 Release 3 (V2R3).

There are maintenance changes to this document for z/OS V2R3 ISPF.

Summary of changes for z/OS Version 2 Release 2 (V2R2)

The following changes are made for z/OS Version 2 Release 2 (V2R2).

Changed

- The amount of ABC data the system uses decreased. See the topic about *ABC data optimization*.

Deleted

No content was removed from this information.

Summary of changes for z/OS Version 2 Release 1 (V2R1)

The following changes are made for z/OS Version 2 Release 1 (V2R1).

There are no technical changes to this document for z/OS V2R1 ISPF.

© Copyright IBM Corp. 1990, 2019

What's in the z/OS V2R3 ISPF library?

You can order the ISPF books using the numbers provided below.

Title
Order Number

z/OS ISPF Dialog Developer's Guide and Reference
SC19-3619–30

z/OS ISPF Dialog Tag Language Guide and Reference
SC19-3620–30

z/OS ISPF Edit and Edit Macros
SC19-3621–30

z/OS ISPF Messages and Codes
SC19-3622–30

z/OS ISPF Planning and Customizing
GC19-3623–30

z/OS ISPF Reference Summary
SC19-3624–30

z/OS ISPF Software Configuration and Library Manager Guide and Reference
SC19-3625–30

z/OS ISPF Services Guide
SC19-3626–30

z/OS ISPF User's Guide Vol I
SC19-3627–30

z/OS ISPF User's Guide Vol II
SC19-3628–30

© Copyright IBM Corp. 1990, 2019

Part 1. Project Manager's Guide

© Copyright IBM Corp. 1990, 2019

Chapter 1. Defining the project environment

This chapter describes the tasks performed by project managers to set up and maintain an SCLM project environment. The required steps are described in detail, with examples and recommended procedures where applicable. After you understand the steps discussed in the first part of this chapter, you can experiment with installing an actual project by completing the steps in “Project manager scenario” on page 39. The data sets used in the scenario are included with the ISPF product. You can use ISPF Option 10.7 to create a small sample project.

If SCLM does not appear on any of your menu panels or on the Menu pull-down, enter TSO SCLM on any ISPF command line. If SCLM is available to your terminal session, the SCLM Main Menu is displayed.

Running different versions of SCLM in multiple partitions

When you upgrade to a new version of SCLM (or apply a PTF), if you have two or more partitions running different versions of SCLM accessing the same SCLM project, you must:

1. Ensure the SCLM project definition load modules are assembled using the earliest version of the SCLM macros.
2. Upgrade the various partitions as required.
3. After all partitions have been upgraded to use the new version of SCLM, assemble the SCLM project definition using the new SCLM macros.

Note:

1. You cannot utilise the features in a new version of SCLM (or PTF) until all the partitions accessing the SCLM project have been upgraded to use the new version of SCLM.
2. In the past, if you had different partitions running different versions of SCLM, you may have experienced problems when upgrading SCLM. To get around this, a process was developed to upgrade an existing SCLM project definition to that of the version of SCLM you were running. However, if you assembled the SCLM project definition using z/OS 1.9 macros and then attempted to access them using a z/OS 1.8 (or earlier) version of SCLM, SCLM generated the message "FLM81204 - - ERROR INITIALIZING THE PROJECT DEFINITION".

Overview of project manager tasks

The primary function of the project manager is to create and manage the project environment. The SCLM project environment consists of three types of information associated with an individual project:

- User Application Data (see “User application data” on page 139)
- Project Definition Data (see “Project definition data” on page 3)
- SCLM Control Data (see “Step 6: Allocate and create the control data sets” on page 17).

Project definition data

The project manager uses the SCLM project definition to generate and maintain the project environment. A project definition defines the desired development environment to SCLM for an individual project. Using the project definition, the product manager can define:

- The structure of the project hierarchy using groups and types
- The languages to use, such as COBOL and Pascal
- The rules to move data within the hierarchy (authorization codes)
- The SCLM options, such as audit and versioning

© Copyright IBM Corp. 1990, 2019

More than one project definition can be generated for a single project. The main project definition for an SCLM project is the *primary* project definition. All other project definitions for the same project are *alternate* project definitions. Alternate project definitions are usually used for performing specific tasks that cannot or should not be done with the primary project definitions. Use of alternate project definitions, if any are required, should be kept to a minimum.

Generating a project environment

To create the project environment, the project manager should be familiar with VSAM data sets and MVS™ high-level qualifiers. It is also helpful if the project manager understands Job Control Language (JCL).

The project manager should determine which compatible programs (such as DB2), if any, are to be used with SCLM, then use the following steps to generate a project environment:

Table 1. The steps to generate a project environment

With standard SCLM	With DB2
“Step 1: Determine the project's hierarchy” on page 4	“Step 1: Determine the project's hierarchy” on page 296
“Step 2: Identify the types of data to support” on page 8	“Step 2: Identify the types of data to be supported” on page 296
“Step 3: Establish authorization codes” on page 8	“Step 3: Establish authorization codes” on page 297
“Step 4: Allocate the PROJDEFS data sets” on page 11	“Step 4: Allocate the PROJDEFS data sets” on page 297
“Step 5: Allocate the project partitioned data sets” on page 12	“Step 5: Allocate the project partitioned data sets” on page 297
“Step 6: Allocate and create the control data sets” on page 17	“Step 6: Allocate and create the control data sets” on page 297
“Step 7: Protect the project environment” on page 23	“Step 7: Protect the project environment” on page 297
“Step 8: Create the project definition” on page 23	“Step 8: Create the project definition” on page 297
“Step 9: Assemble and link the project definition” on page 38	“Step 9: Assemble and link the project definition” on page 299

Step 1: Determine the project's hierarchy

As a project manager, you are responsible for generating and updating the hierarchy of the project to accommodate project requirements. This step helps you plan the project hierarchy. When you have completed this step, you should have a diagram of the hierarchy with all the groups labeled, as well as an understanding of how each group is used.

It is usually easier to draw a diagram of your hierarchy, to help you visualize what the hierarchy looks like. The following rules govern the creation of hierarchies:

- Each group can have no more than one parent.
- Each group can have multiple groups promoting into it.
- There is no restriction on the total number of groups a hierarchy can have.

- A hierarchical view can contain no more than 123 groups. This is because MVS has a limit of 123 extents for a concatenated partitioned data set.
- Each hierarchy has one root group, the topmost group.
- It is possible to have more than one hierarchy defined for one project.
- Defining no more than four layers makes it easier to use ISPF tools on the SCLM-controlled members.

The following two figures show two examples of hierarchies. These hierarchies are set up based on the development phases potential projects might use. You can create hierarchies other than those presented here. As a project evolves, the requirements that the project has on the hierarchy will change. With SCLM, you can change the hierarchy to meet the needs of the project.

The reasoning behind the hierarchy shown in Figure 2 on page 5 follows:

- The development groups (USER1, USER2, and USER3) are where all modifications to SCLM-controlled members are made.
- The INT group is for integrating (combining) all the SCLM-controlled members from the development groups.
- The TEST group is the group where system or function testing of the application will take place.
- The RELEASE group will contain the final version of the application being developed. It is from this group that the application could be put into production.

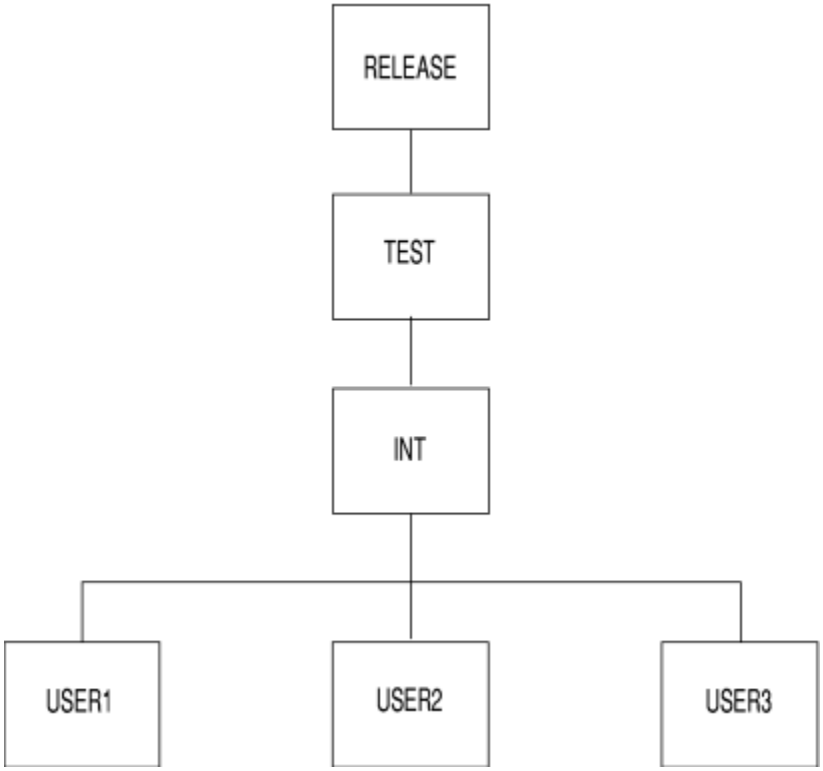


Figure 2. Example of SCLM Hierarchies

The second hierarchy, shown in Figure 3 on page 6, is different. This hierarchy has two separate legs. Each leg of the hierarchy contains a separate subsystem of the application being developed. The stage groups (STAGE1 and STAGE2) in each hierarchy leg are used for integrating and unit testing the subsystems within each hierarchy leg. The SYSTEST group is used to combine the subsystems from both legs of the hierarchy for delivery to a system test organization.

Step 1: Determine the project's hierarchy

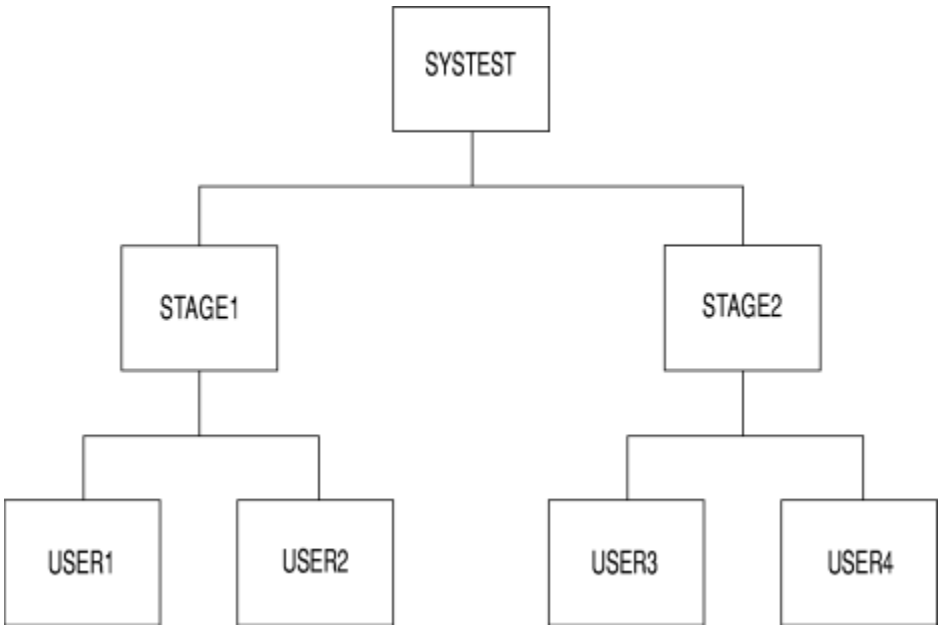


Figure 3. Example of SCLM Hierarchies

Use the preceding rules and the requirements of your project to draw your hierarchy and label each group.

Primary non-key group testing techniques

You can use primary non-key groups as a technique to allow integration and testing of a software application. The technique is useful where integration work can have far-reaching and undesirable effects, for example, when a global change to an application affects the majority of developers. The technique is also useful when schedule or other pressures are such that you must perform high-risk integration of software. SCLM does not allow you to promote from a primary non-key group.

In a normal SCLM scenario, you promote code from individual development libraries to a common integration group before performing integration testing. However, you can generate an alternate project definition that deviates from the default project definition. The alternate project definition defines an intermediate non-key group for integrating subsets of development groups. Define the non-key group so that only key groups promote into the non-key group. Developers authorized to this intermediate group can then promote code to it for unit and function testing. Testing takes place in this group before promotion to the normal integration group. Because being at a non-key group does not cause members to be purged from a key group during a promote, no members are removed from the default project definition. In this way, you avoid potential integrity problems.

Using this technique, the activities of small groups of integrators do not affect the normal hierarchy until their testing is complete. By switching to the alternate project definition, developers can easily test their integration by promoting to the primary non-key group. When promoting to a non-key group, code still exists in the normal hierarchy in the development libraries. SCLM promotion from the development libraries, using the default project definition, would then incorporate the code into the normal integration group. New code can go through an accurate configuration test before being applied to the normal hierarchy. Code developed using this scenario is potentially more complete and accurate than code developed in a normal scenario.

Use Figure 4 on page 7 and Figure 5 on page 7 to compare a default hierarchy structure with an alternate hierarchy structure. Figure 4 on page 7 shows a default hierarchy structure for a project. You can perform all normal development activities within the default hierarchy structure.

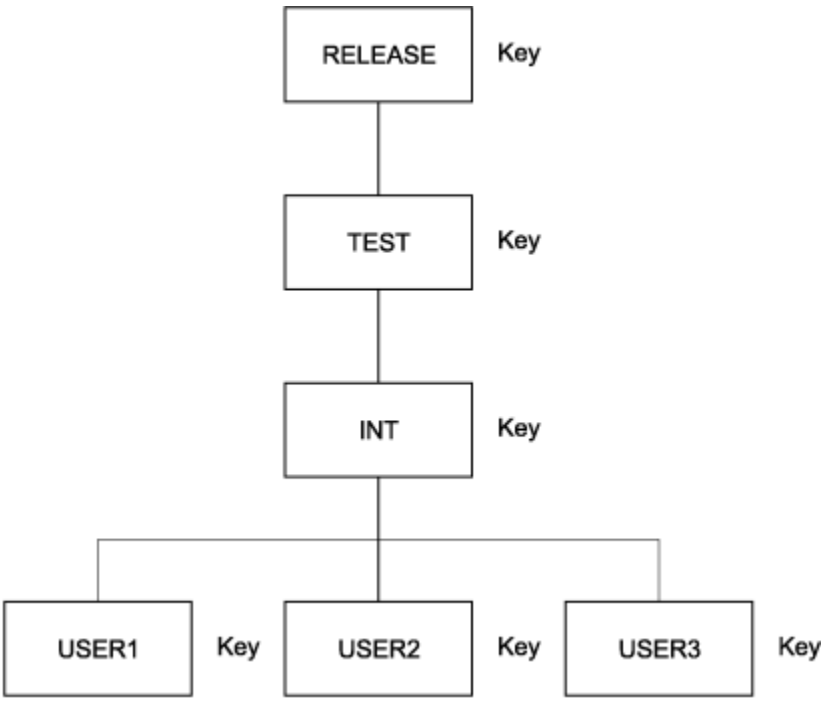


Figure 4. Default (Primary) Project Hierarchy Structure

Figure 5 on page 7 shows an alternate hierarchy structure with a primary non-key integration group for the project shown in Figure 4 on page 7.

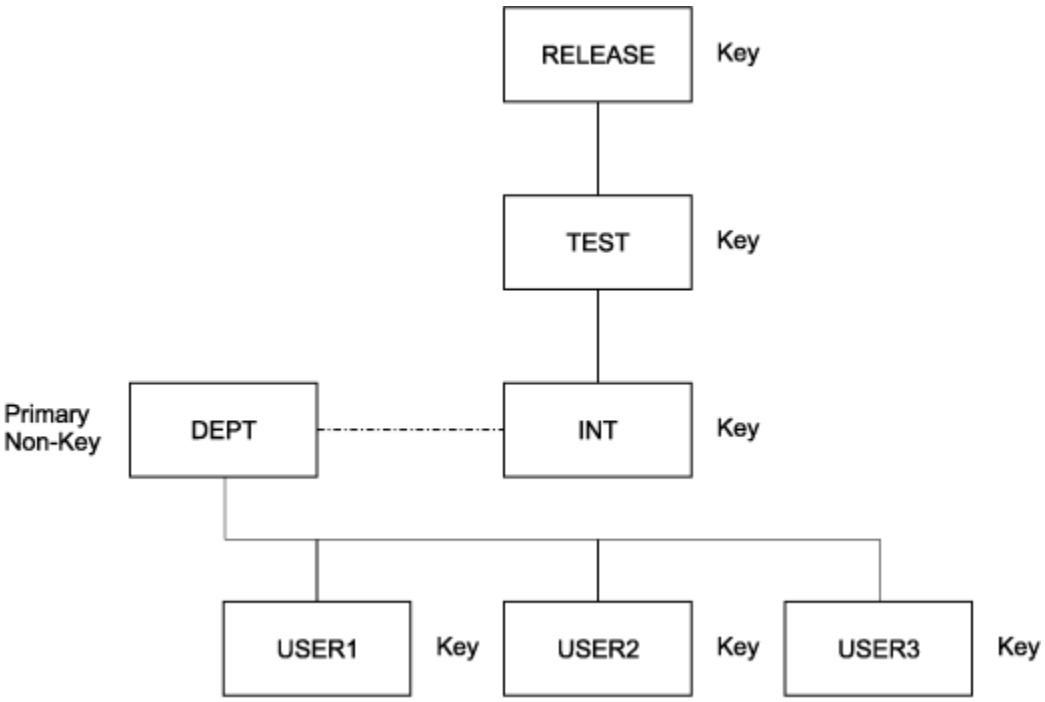


Figure 5. Alternate Project Hierarchy Structure with Primary Non-key Integration Group

In the example, the developers (USER1, USER2, USER3) can use the alternate project definition to promote code into the primary non-key group. You cannot promote up from the primary non-key group, but you can draw down from it.

Promotion to a non-key primary group does not cause deletion of the components from the respective development libraries. Building in the primary non-key group allows the developers to integrate and test

pieces of code still under development. Code that is then complete can be promoted through the default project definition from the development libraries into the normal integration group. The promotion to the normal integration libraries causes the components to be deleted from the respective development libraries, but not from the primary non-key group. Deletion from the primary non-key group must be done manually using the SCLM Library Utility, the Delete from Group Utility or through SCLM services, such as DELGROUP.

Step 2: Identify the types of data to support

This step identifies the types of data required by the applications under development for your project. Some examples of the types of data used are source code, object modules, load modules, and source listings. The list of types developed in this step is used in later steps.

SCLM supports the same kind of data supported by MVS partitioned data sets. The amount of data is also a factor in determining the types of data needed. Different types (such as objects and listings) of data should not reside in the same SCLM type. Determine the number of types you need based on the data you want to maintain for the project. For example, if you want to maintain compiler listings, a listing type is necessary. At a minimum, use four types to produce executable code:

- Source type for application source code
- Object type for generated object code
- Load type for generated load modules
- Architecture type for architecture definition members.

Similar kinds of data can reside in separate types. For example, you can divide source code into assembler source code and Pascal source code. To do this, identify an assembler type and a Pascal type.

Step 3: Establish authorization codes

Authorization codes control the movement of data within the hierarchy. The purpose of this step is to assign authorization codes to the hierarchy. Authorization codes restrict the draw down and promotion of members to certain groups within the hierarchy.

At least one authorization code must be defined for a project. If no authorization codes are defined, SCLM will not permit members to be drawn down or promoted. Authorization codes work only on editable types such as source, not on build outputs. Authorization codes are assigned to each group in the hierarchy. Groups can have any number of authorization codes assigned to them. Members are assigned authorization codes when they are registered with SCLM. Members can only exist in groups that have been assigned the same authorization codes as the members.

It is not necessary to define more than one authorization code for the entire project. A single authorization code allows each member under SCLM control to be drawn down to any development group and be promoted to the top of the hierarchy. If tighter restrictions on the movement of your data are required for your project, you must identify those situations and define additional authorization codes.

An example of when multiple authorization codes can be used is when an application has multiple subsystems being developed in different legs of the hierarchy and you need to ensure that the members of the two subsystems do not get mixed in the development groups in the hierarchy legs. Authorization codes can be set up to prevent the members from one subsystem from being drawn down into the development groups of the other subsystem. This requires at most two authorization codes. For additional possible uses of authorization codes, see “Using authorization codes to control SCLM operations” on page 9.

Using the diagram that you drew for Step 1, examine the flow of members and determine if any restrictions on the movement of members are required. Label each group with at least one authorization code. Authorization codes can be up to 8 characters and cannot contain commas.

Using authorization codes to control SCLM operations

Authorization codes restrict promotions and draw downs on a member-by-member basis for source code only. This section discusses some uses of authorization codes.

First, some facts about authorization codes:

- An authorization code is a character string up to 8 characters and cannot contain commas.
- When you create the project definition, you assign zero or more authorization codes to each group.
- Each member of every group within an SCLM-controlled project is assigned one authorization code.
- In order to put a member into a group, the authorization code of that member must match one of the authorization codes that have been assigned to the group.
- When all the authorization codes are removed from a group, no members can be promoted into or out of that group.
- When you promote a member from one group to the next, the member retains its authorization code. Thus, the group being promoted into and the group being promoted from must have a matching authorization code. If, as a result of a promote, an older version of the module was replaced, the authorization code assigned to that older version is not kept.

Figure 6 on page 9 shows a simple hierarchy with four groups: RELEASE, TEST, DEV1 and DEV2. The group RELEASE has been assigned only one authorization code: DEV. Group TEST has two authorization codes: DEV and TESTONLY. Three authorization codes (DEV, PROTO, and TESTONLY) have been assigned to DEV1. Group DEV2 has DEV and L0 as its authorization codes.

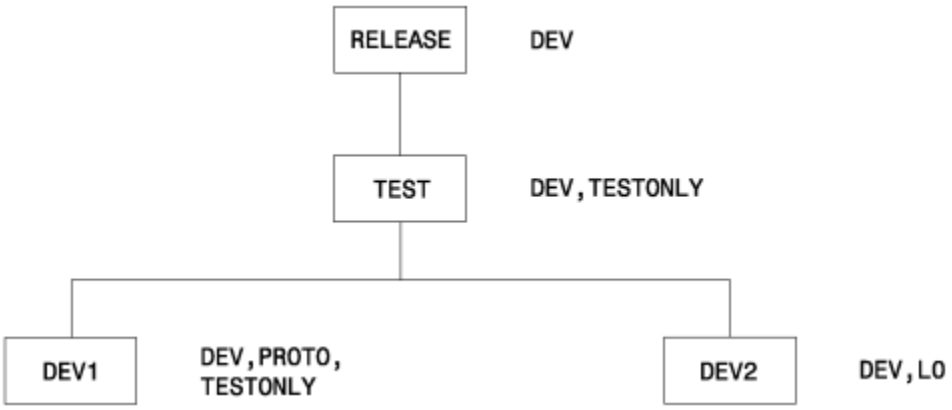


Figure 6. Sample Hierarchy with Authorization Codes

Code this information in the project definition as follows:

```
RELEASE  FLMGROUP   KEY=Y,AC=(DEV)
TEST     FLMGROUP   KEY=Y,AC=(DEV,TESTONLY),PROMOTE=RELEASE
DEV1     FLMGROUP   KEY=Y,AC=(DEV,TESTONLY,PROTO),PROMOTE=TEST
DEV2     FLMGROUP   KEY=Y,AC=(DEV,L0),PROMOTE=TEST
```

In Figure 6 on page 9, the following relationships exist:

- A member in DEV1 with an authorization code of PROTO cannot be promoted because group TEST does not have PROTO as an authorization code.
- For the same reason, a member in DEV1 with an authorization code of TESTONLY can be promoted to TEST, but cannot be promoted to RELEASE.
- Similarly, a member in DEV1 or DEV2 with an authorization code of DEV can be promoted all the way up to group RELEASE.
- A member in DEV2 cannot have an authorization code of TESTONLY or PROTO; it must be either DEV or L0.

- A member in DEV2 with an authorization code of L0 cannot be promoted because group TEST does not have L0 as an authorization code.

When you edit a member in a development group, SCLM looks at the authorization code you specified on the edit panel and tells you the following information:

- If that authorization code is not valid for that development group, you must enter an authorization code that is assigned to that group. If you enter an invalid authorization code and then press the help key, SCLM shows authorization codes for that group.
- If use of that authorization code prevents promotion of that member at some point in the group hierarchy, SCLM gives you the name of the group into which promotion is not allowed.
- If use of that authorization code leads to a potential promotion conflict with another member of the same name, SCLM does not allow the edit. An example of this problem follows.

  SCLM allows you to have two members of the same name and type residing in two different development groups (such as DEV1 and DEV2 in Figure 6 on page 9) under certain conditions. Each of those members has an authorization code assigned to it. Those codes, along with the authorization codes assigned to the higher groups in the hierarchy, determine how far up the hierarchy each of those members can be promoted. If the two promotion paths do not intersect, SCLM lets you edit those members in those groups. However, if there is at least one group through which both members can be promoted, changes made to one member would be lost when the other member is promoted. In that case, SCLM does not let you edit the members in those groups.

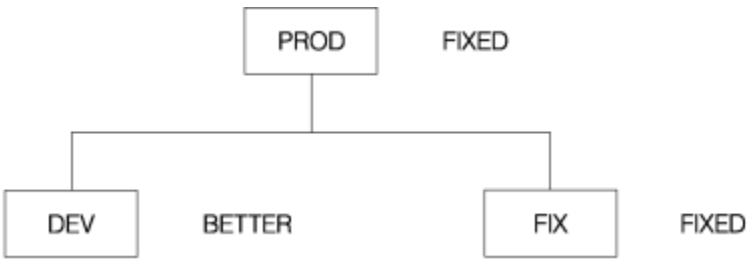
  If a member exists in group DEV1, SCLM uses authorization codes to determine whether you can edit a member with the same name and type in group DEV2:

Table 2. Authorization Code Allowances

Auth. Code for member in DEV1	Auth. Code for member in DEV2	Allowed?	Why?
DEV	DEV	No	Both members can be promoted through TEST.
DEV	L0	Yes	Promotion paths do not intersect.
PROTO	TESTONLY	No	TESTONLY is not a valid authorization code for DEV2.
PROTO	L0	Yes	Promotion paths do not intersect.
TESTONLY	DEV	No	Both members can be promoted through TEST.
TESTONLY	L0	Yes	Promotion paths do not intersect.

Allowing parallel updates

You can use the information in the previous section to set up a project in which you can make modifications to what you have in production (development) while being able to make quick fixes to production modules (maintenance). The simple hierarchy is illustrated in the following example. An actual hierarchy can contain many groups and layers.



Define the groups as follows:

```
PROD      FLMGROUP   KEY=Y,AC=(FIXED)
DEV       FLMGROUP   KEY=Y,AC=(BETTER),PROMOTE=PROD
FIX       FLMGROUP   KEY=Y,AC=(FIXED),PROMOTE=PROD
```

There are three groups: PROD is the production library, DEV is the development library, and FIX is the maintenance library. In practice, there would be a much larger subhierarchy under both DEV and FIX in order to allow for both multiple developers and for testing of applications before moving them to production.

DEV, FIX, and PROD each have a single authorization code, BETTER, FIXED, and FIXED respectively, and could have more. More importantly, no authorization code is assigned to both DEV and PROD. It is this aspect of the project definition that prevents the promotion of any modules from group DEV into group PROD. When the development code is ready to move into production, the authorization code BETTER must be added to the valid authorization codes for the PROD group.

A programmer planning to make changes to a module for the next release of an application draws the module down from PROD into DEV, specifying an authorization code of BETTER on the SCLM EDIT-ENTRY PANEL. Changes are made and tested in DEV.

Suppose that while the module is being changed and tested in the DEV group, a user encounters a problem with the application and another programmer determines that the fix requires a change to the module that has been drawn down to DEV.

The programmer can draw down the module into FIX even though that same module has been drawn down into DEV. This is possible because the promotion paths of the two modules do not intersect; the module in DEV cannot be promoted into PROD because of authorization codes. Therefore, changes made to one module do not overwrite changes made to the other copy.

When the fix has been made to the module in FIX and the application has been rebuilt at that group, the user can run the application from group FIX until the fix has been verified and then promoted to PROD.

Before the fix is promoted, the changes must be incorporated into the copy of the modules in DEV. This is a manual change made by the current owner of the modules in DEV with the assistance of the person who made the changes in FIX.

Keep in mind that although authorization codes can be used to restrict promotion paths, they do not provide security against modifications to SCLM-controlled data made outside of the SCLM environment. You should use RACF® (or the functional equivalent) for that purpose.

Step 4: Allocate the PROJDEFS data sets

The PROJDEFS data sets are used to store the project definition data for an individual project. The purpose of this step is to allocate the PROJDEFS data sets.

The PROJDEFS data sets are partitioned data sets with the following naming convention:

```
project_id.PROJDEFS.*
```

SCLM requires that the load data set be named:

```
project_id.PROJDEFS.LOAD
```

When a user invokes SCLM for a specific project, SCLM uses the current assembled version of the project definition located in the LOAD data set.

The data sets containing the project definition's source and object code are not required by SCLM to follow the PROJDEFS naming convention, but it is recommended to make maintaining the project definition easier. Therefore, following the naming convention would produce the following data sets:

```
project_id.PROJDEFS.SOURCE
project_id.PROJDEFS.OBJ
```

Allocate the PROJDEFS data sets using the attributes defined in Table 4 on page 17. The PROJDEFS data sets should be protected from access by general users. Protecting the PROJDEFS data sets is discussed in “Step 7: Protect the project environment” on page 23.

Step 5: Allocate the project partitioned data sets

The project partitioned data sets are used to store the user application data. These data sets are organized into a hierarchy and controlled by the project definition. Allocate the project partitioned data sets using either the ISPF Data Set Utility (option 3.2) or a JCL process. Use the information in this step to determine the names, number, and physical characteristics of the project partitioned data sets.

Data set naming conventions

SCLM expects all the project partitioned data sets to use the default naming convention of `project.group.type`. Because some projects cannot use the default naming convention, SCLM allows the project manager to specify an alternate naming convention either for all the project partitioned data sets or for the project partitioned data sets associated with individual groups in the hierarchy.

If your data already exists, the existing data sets can be used in conjunction with SCLM's flexible data set naming capability. The next section provides additional information on using this capability.

Flexible naming of project partitioned data sets

With SCLM, product managers can use the SCLM-supplied default data set naming convention or a user-defined naming convention. The default naming convention is PROJECT.GROUP.TYPE. If the SCLM default naming convention is not used, the project manager's convention must use the MVS naming conventions. For example, it is possible to use four or five qualifiers in the data set names instead of the three qualifiers that are used by the SCLM naming convention. (The PROJDEFS data sets are exceptions; these data sets must use the naming convention defined in “Step 4: Allocate the PROJDEFS data sets” on page 11.)

To define a naming convention other than SCLM's default naming convention, you must specify data set names that correspond to specific groups or the entire project. While the names of the data sets used by SCLM can use more than three qualifiers, the developers still see the PROJECT.GROUP.TYPE naming convention on the SCLM dialog panels and service calls. The project definition creates a mapping between the PROJECT.GROUP.TYPE name and the user-defined data set names associated with each group in the hierarchy.

Note: This mapping is only maintained while users are executing SCLM functions. If ISPF utilities are used on data controlled by SCLM, the users should know the mapping between the PROJECT.GROUP.TYPE name and the fully qualified data set name.

The data set names are defined in the project definition with the FLMCNTRL and FLMALTC macros. Each macro has a DSNAME parameter that allows the project manager to specify the data set names for the entire project or for individual groups. The FLMCNTRL macro defines the data set names for the entire project; the FLMALTC macro defines the data set names on a group-by-group basis. “FLMALTC macro” on page 510 includes an example of how to set up the macros to use flexible naming of partitioned data sets.

The DSNAME parameters on both macros work the same way and can be used within the same project definition. The value specified on the DSNAME parameter is a pattern for the data set name. This pattern must meet MVS naming conventions and can contain the SCLM variables @@FLMPRJ, @@FLMGRP, and @@FLMTYP. If DSNAME is not specified, SCLM uses the default naming convention of PROJECT.GROUP.TYPE. The use of variable @@FLMTYP is required. SCLM verifies that the variable @@FLMTYP is used on each DSNAME parameter when the project definition is loaded into memory. The variable @@FLMGRP is **very strongly** recommended. The use of these variables minimizes the risk that data set names associated with different groups are the same and prevents data from being overwritten. The variable @@FLMPRJ is optional.

The SCLM variable @@FLMDSN is created from the value of the DSNAME parameter. Therefore, if the data set name pattern is @@FLMPRJ.component_name.@@FLMGRP.@@FLMTYP, the value of @@FLMDSN will be @@FLMPRJ.component_name.@@FLMGRP.@@FLMTYP.

The versioning partitioned data sets can also use a naming convention other than SCLM's default naming convention. The VERPDS parameter on the FLMCNTRL and FLMALTC macros is used to specify the name of the versioning partitioned data sets. SCLM uses a default of @@FLMDSN.VERSION for the names of the versioning data sets. If a pattern other than the default is used, the variables @@FLMGRP and @@FLMTYP must be part of the data set name pattern. Using two variables minimizes the risk that the versioning data set names associated with different groups are the same, and prevents data from being overwritten.

Attention:

SCLM does not guarantee the uniqueness of the data set names or check the validity of values entered on the DSNAME parameter.

Number of data sets to allocate

Normally, a data set should be allocated for every possible PROJECT.GROUP.TYPE combination in the hierarchy. However, if the intent is to develop code in several hierarchies that merge in one main hierarchy, there might be no need to allocate some data sets. Allocating only the data sets that are actually used saves time when creating the hierarchy and minimizes DASD use and catalog entries. See Figure 7 on page 15 for an example of a hierarchy that does not have all data sets allocated.

Only those data sets actually used in the hierarchy must be physically allocated. SCLM functions will execute successfully for hierarchies that contain unallocated data sets, as long as the unallocated data sets are not used. If a data set is not allocated and SCLM attempts to use the data set, an error message is issued.

Data sets can be added at any time. If you leave a data set unallocated and later find you need it, simply allocate the data set then.

Determining when data set allocation is necessary

You can leave the data sets for the intermediate groups in your project unallocated until the first time they are needed for a promote. You can also leave the data sets for types that will not be used at a particular group unallocated. As an example, if a developer is responsible for source code but not panels, then you can leave the data set for the type containing panels unallocated for that developer's group.

A data set need not be allocated if an EXTEND type is being used and the hierarchy is designed so that the source code for the EXTEND type is always at a higher group.

For example, consider a project definition with the FLMTYPE macro written as follows:

```
CMNSRC    FLMTYPE
BLDSRC    FLMTYPE  EXTEND=CMNSRC
```

In this situation, the type CMNSRC can contain members referenced by members in the BLDSRC type. However, if the source code in CMNSRC will always be at a higher layer in the hierarchy (for example, IVV), you do not need to allocate data sets for type CMNSRC below the IVV layer in the main hierarchy.

How SCLM functions use data sets

SCLM uses a data set when it expects that the data set already contains a member (for example, when attempting to delete a member), or when the data set will contain a member (for example, when saving a new member). The following list details how SCLM functions use a data set:

Build
: Uses a data set if it contains a member that has a corresponding accounting record and that member is being built or referenced by another member that is being built. Build also uses data sets for output (those referenced by the LOAD, OBJ, or LIST architecture keywords, for example).

Promote
: Uses a data set if it contains a member that has a corresponding accounting record and that member is being promoted. If these data sets contain members that need to be promoted, they must be present in the current group and in the group being promoted to; otherwise, an error message is

issued. If a promotion occurs from a non-key group to a key group, the corresponding data sets at the previous key group will also be used.

Delete
Uses a data set when deleting a member.

Delete from Group
Uses a data set when deleting a member.

Library Utility
Uses a data set when deleting a member or when Edit, View or Build are invoked.

Import
Uses a data set when VSAM records are being imported into the hierarchy. The member imported must exist somewhere in the hierarchy view for the group being imported into.

Edit
Uses a data set when storing or retrieving a member.

View
Uses a data set when retrieving a member.

Migrate
Uses a data set to retrieve information about a member that is being migrated into the SCLM hierarchy.

Parse
Uses a data set when parsing a member.

Package Backout
The package details file contains an entry for each package, listing the members in that package. This is built by Promote and used by Package Backout.

Manipulating VSAM records for unallocated data sets

A build map can be created for a member that is higher in the hierarchy but for which there is no source data set allocated for the group where the build is occurring. If you delete a data set, the corresponding accounting records and build maps can still exist in the VSAM databases.

Using the following utilities and services, you can browse or delete VSAM records that correspond to an unallocated data set.

Library Utility
Browse and delete accounting records and build maps that correspond to an unallocated data set.

Delete
Delete accounting records and build maps that correspond to an unallocated data set.

Delete from Group
Delete accounting records and build maps that correspond to an unallocated data set.

Examples of hierarchies with unallocated data sets

A valid hierarchy that contains unallocated data sets is shown in Figure 7 on page 15. Member B INCLUDES member C. A build of member B from group USR1 will succeed, although a data set was not allocated for Cmnsrc at the INT group. The build will locate and use member C from the IVV group.

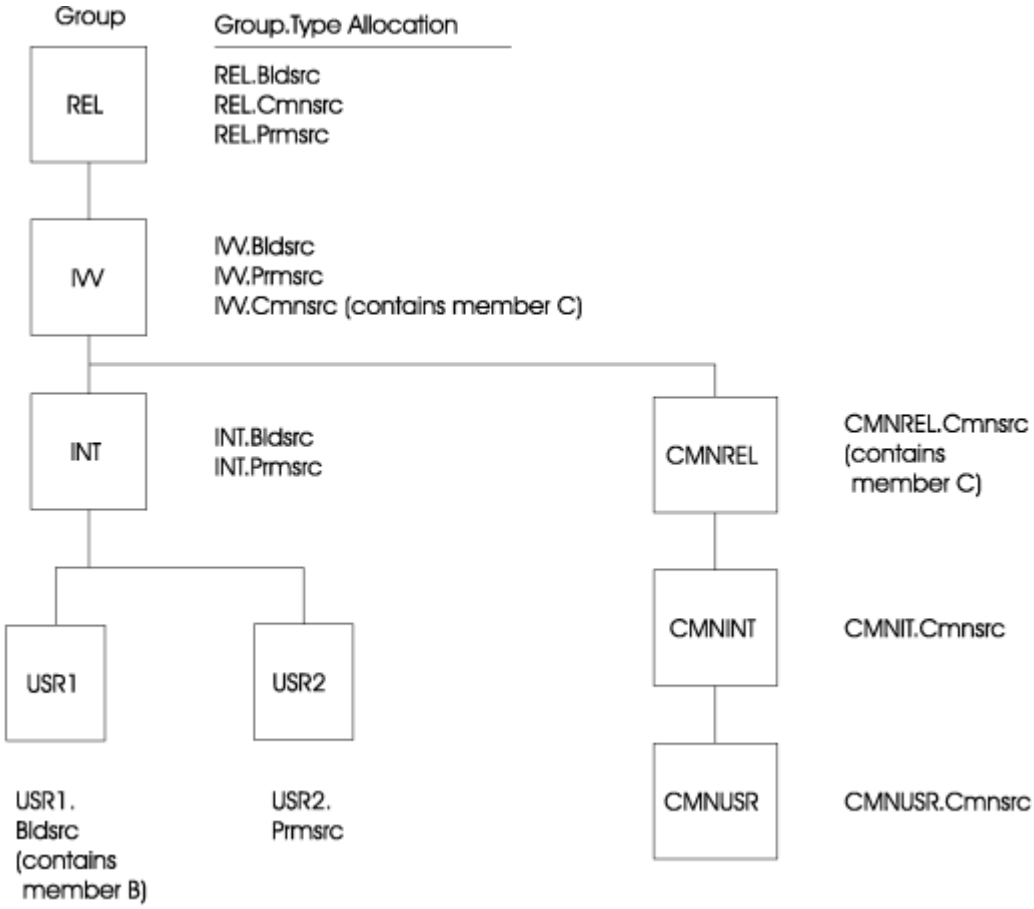


Figure 7. Valid Hierarchy with Unallocated Data Sets

A hierarchy that is not valid for the intended operation is shown in Figure 8 on page 16. A promote of member B from the IVV group, which INCLUDES member C, will fail, because promote will attempt to copy member C in IVV.Cmnsrc to REL.Cmnsrc.

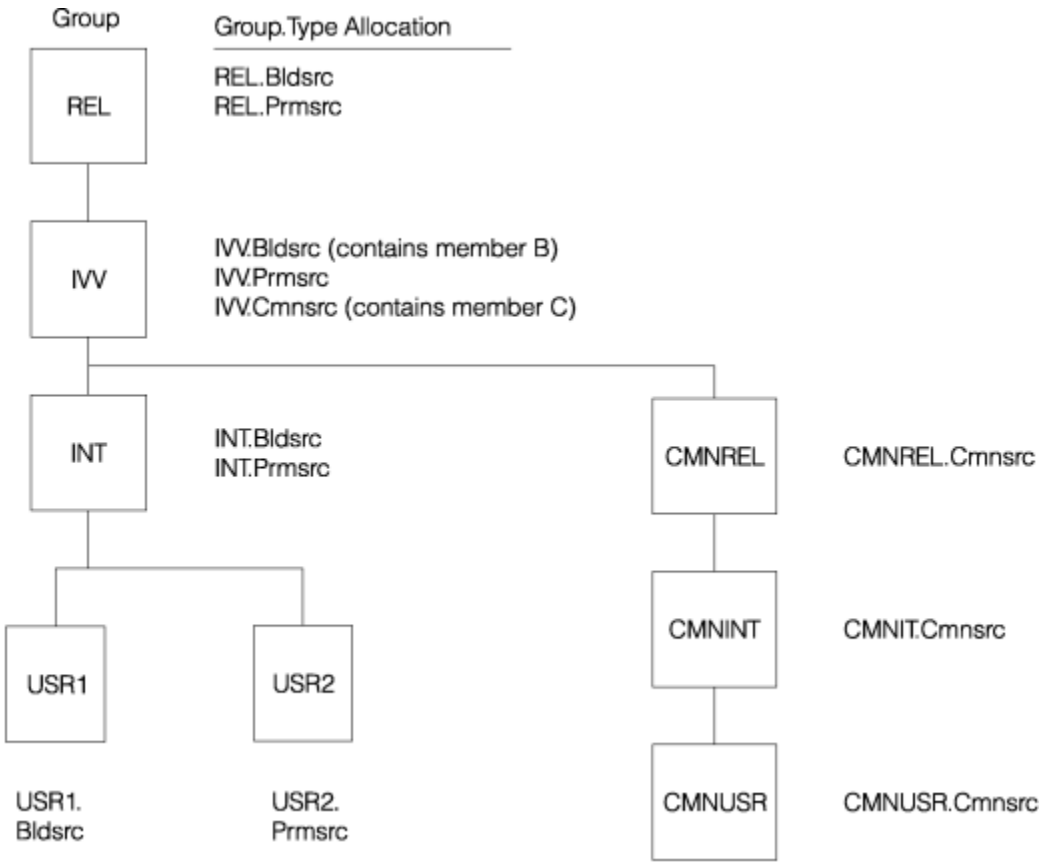


Figure 8. Invalid Hierarchy for Intended Operation

Versioning partitioned data sets

If the versioning capability is going to be used, at least one versioning partitioned data set must be allocated. If you intend to use the VERCOUNT parameter on the FLMCNTRL macro to specify that two or more versions be maintained, you must specify at least one versioning partitioned data set for each group to be versioned. Otherwise, errors can occur during version retrieval. You can also choose to have a versioning partitioned data set associated with each 'group.type' to be versioned.

Table 3 on page 16 shows the attributes required for the versioning partitioned data set. All attributes must be coded as shown, with the exception of LRECL, which defines the minimum LRECL allocation required for versioning. The LRECL value must be at least 259 and must be 4 bytes more than the LRECL of the largest source data set to be versioned.

Table 3. Versioning Data Set Attributes

Attribute	Description
LRECL =	The larger of 259 and the source data set's LRECL + 4
RECFM =	Variable Blocked (VB)
BLKSIZE =	The largest of the LRECL + 4 Bytes, the source data set's BLKSIZE, and the optimal block size for your system.

The 4 bytes in the block size calculation are for MVS control information, specifically for the *blocklength* field. For example, with a blocking factor of 10 the block size would be calculated as follows:

```
(259 x 10) + 4 = 2594
```

Project partitioned data sets

This section provides guidance on what data set attributes should be used for the project partitioned data sets. SCLM does not restrict the format of a data set.

Note: Data sets of the same type must be allocated with the same attributes.

Table 4 on page 17 lists recommended data set attributes for some typical types. For best performance, specify blocksize=0 to use the system-determined block size. Load module data sets should be allocated with a block size of 6144 or greater.

Table 4. Data Set Attributes

Type	RECFM	LRECL
Source	FB	80
Object	FB	80
Load	U	0
Listings	VB	137
Link-edit Maps	FBA	121
Architecture definitions	FB	80
Other Text	FB	80

Space considerations

SCLM has no special considerations that require the allocation of additional space in the project partitioned data sets. Allocate the size of the project partitioned data sets according to the amount of data that will be stored in them.

Step 6: Allocate and create the control data sets

Control data sets are used to track and control application programs within the hierarchy. SCLM stores accounting and audit information in VSAM data sets whose names are defined in the project definition. VSAM data sets consist of VSAM clusters. A *VSAM cluster* is a named structure consisting of a group of related components. While it is not required that the first qualifier of VSAM data sets match the project name, it makes project maintenance easier. There are seven types of VSAM data sets that can be associated with a project.

Primary Accounting
The accounting data set contains information about the software components in the project including statistics, dependency information and build maps (information about the last build of the member). At least one accounting data set is required for a project.

Secondary Accounting
The secondary accounting data set is a backup of the information in the accounting data set.

Export Accounting
The export accounting data set contains accounting information that has been exported from the accounting data set.

Primary Audit Control
The audit control data set contains audit information about changes to the software components in the project for groups that have auditing enabled.

Secondary Audit Control
The secondary audit control data set is a backup of the information in the audit control data set.

Cross-dependency data set
The Cross-dependency data set contains information used by the Where-used function.

Control
The control data set contains information about SCLM administrators, member locking, and the backup of unpromoted members.

Most projects start out with one VSAM data set, the primary accounting data set. Additional data sets can be added as the project evolves and more advanced SCLM capabilities are needed. Additional VSAM data sets are required for Import, Export, Auditing, automatic backup of accounting data and multiple control data set support. In some cases, it is desirable to use multiple VSAM data sets instead of one or two. If this is the case, see "Splitting project VSAM data sets" on page 69 for additional information.

SCLM uses VSAM Record Level Sharing (RLS) to support sharing the VSAM data sets across systems in a sysplex environment. This support requires:

- The Coupling Facility
- a VSAM cluster allocated with the proper characteristics for VSAM RLS
- VSAMRLS=YES specified on the FLMCNTRL macro in the SCLM project definition.
- Share options specified as 3,3 (recommended) on the VSAM file allocation.

See *z/OS DFSMStvs Planning and Operating Guide* for information about the hardware and software requirements to support VSAM RLS.

The VSAM data sets cannot be shared between systems in a sysplex under any other condition. Accessing any of the VSAM data sets from multiple systems when VSAM RLS is not available can result in the corruption of data, system errors, or other integrity problems. To avoid these problems, the project manager must allocate VSAM data sets so that they cannot be accessed from multiple systems. What this means is you should ensure that the VSAM data sets are allocated on DASD, or catalogs, not available to other systems in the sysplex.

Except when using the RLS, the share options required on the VSAM file allocations are 4,3. This means that the data set can be fully shared by any number of users in a single system, and VSAM refreshes the data and index components buffer pools for direct processing, to guarantee the coherency of the data in the buffer pool. Coherency, in this case, means that SCLM gets the most updated contents of the requested record.

All VSAM data sets should be REPROed periodically using the IDCAMS reproduction utility. This will reduce fragmentation and optimize the performance of your VSAM data sets.

Create the accounting data sets

The accounting data sets contain information about the application programs in the hierarchy, including statistics, dependency information, and build maps. SCLM functions use the accounting information to control and track members in the project partitioned data sets. Each project must have at least one primary accounting data set.

An optional secondary accounting data set can be created. The secondary accounting data set is a backup for the primary accounting data set. It allows for the restoration of accounting information if the primary data set becomes corrupted, for example due to a disk failure. This data set name must be unique. The secondary accounting data set should be stored on a different volume than the primary accounting data set. If a secondary data set is used, the performance of SCLM will be degraded, because updates are made to both the primary and secondary data sets. The information in both data sets should be compared periodically to ensure the integrity of the accounting information.

Both the primary and secondary accounting data sets are created the same way. Each accounting data set for the project must be a VSAM cluster. Use the IDCAMS utility to define accounting data sets. If accounting information for different groups is to be kept in separate accounting data sets, additional accounting data sets must be created. An example of the JCL used to define an accounting data set follows:

Note: This example is called FLM02ACT and is in the data set ISP.SISPSAMP that is included with ISPF. ISP.SISPSAMP also contains a sample for the allocation of the data set for Record Level Sharing. It is called FLM02RLS.

```
//jobname  JOB (wkpkg,dpt,bin),'name'
//* code additional JOBCARD statements here
//*********************************************************************
//*
//*  THIS JCL EXAMPLE DEFINES A VSAM CLUSTER TO BE USED AS THE SCLM
//*    ACCOUNTING FILE FOR A GIVEN PROJECT.
//*  THE HIGH-LEVEL QUALIFIER MUST BE AN ENTRY IN A VSAM USER CATALOG
//*    IN ORDER TO CREATE THIS CLUSTER.
//*  TO SPECIFY THE FILE, CHANGE THE DEFINE CLUSTER STATEMENT BELOW
//*    AS FOLLOWS:
//*
//*    1) ADD THE FOLLOWING LINE OF JCL TO DELETE THE VSAM CLUSTER
//*       BEFORE THE ALLOCATION IF THE DATA SET ALREADY EXISTS
//*       AND IT NEEDS TO BE DELETED:
//*       DELETE 'project.account.file' CLUSTER
//*       ADD THIS STATEMENT BETWEEN THE //SYSIN ALLOCATION AND THE
//*       DEFINE CLUSTER LINE OF JCL.
//*    2) CHANGE ALL project.account.file TO THE DESIRED FILE NAME.
//*        THIS VALUE IS SPECIFIED ON THE FLMCNTRL AND FLMALTC
//*        MACROS.  IF MORE THAN ONE VSAM ACCOUNTING DATA SET IS
//*        SPECIFIED ON THE FLMCNTRL AND FLMALTC MACROS, MULTIPLE
//*        IDCAMS DEFINE STEPS ARE REQUIRED.
//*        ACCOUNTING DATASET NAMES ARE USUALLY CHOSEN IN THE FOLLOWING
//*        MANNER - "PROJECT.ACCOUNT.FILE" (WHICH IS THE DEFAULT
//*        USED IN THE PROJECT DEFINITION IF NONE IS SPECIFIED).
//*    3) MODIFY CYLINDERS (PRIMARY SECONDARY)
//*    4) SPECIFY THE VOLUME VVVVVV ON WHICH IT WILL BE ALLOCATED
//*
//*  A JOB STEP IS THEN EXECUTED TO INITIALIZE THE FILE.
//*
//*********************************************************************
//STEP1   EXEC PGM=IDCAMS
//*
//SYSPRINT  DD SYSOUT=H
//*
//SYSIN     DD *
     DEFINE CLUSTER +
        (NAME('project.account.file') +
         CYLINDERS(4 1) +
         VOLUMES(VVVVVV) +
         KEYS(26 0) +
         RECORDSIZE(264 32000) +
         SHAREOPTIONS(4,3) +
         SPEED +
         SPANNED +
         UNIQUE) +
         INDEX(NAME('project.account.file.INDEX') -
         ) +
         DATA(NAME('project.account.file.DATA') -
         CISZ(2048) +
         FREESPACE(50 50) +
         )
/*
```

Figure 9. Accounting File Example (Part 1 of 2)

```
//*********************************************************************
//*
//*  INITIALIZE THE ACCOUNTING FILE
//*
//*********************************************************************
//STEP2    EXEC PGM=IDCAMS
//INPUT  DD *
                        SCLM ACCOUNTING FILE INITIALIZATION RECORD
/*
//OUTPUT DD DSN=project.account.file,DISP=SHR
//SYSPRINT  DD SYSOUT=H
//SYSIN     DD *
     REPRO INFILE(INPUT) OUTFILE(OUTPUT)
/*
//*
)CM 5665-402 (C) COPYRIGHT IBM CORP 1980, 1989
```

Figure 10. Accounting File Example (Part 2 of 2)

Space considerations for the accounting data sets

Each accounting data set requires approximately three cylinders of 3390 DASD for every 1000 partitioned data set members that SCLM controls. The space required varies depending on how much information SCLM will control. If additional space in the data set is desired, modify the space parameter (shown as CYLINDERS in the example JCL).

Create the export data sets

The export control data sets are optional unless the export and import functions are used.

Before using the EXPORT service, you must allocate and define an export accounting data set.

To prepare for the export operation:

1. Define the export accounting data sets to the project using the FLMCNTRL and FLMALTC macros. Do not use data set names that have already been specified for any ACCT or ACCT2 parameters in the FLMCNTRL and FLMALTC macros.

   Note: SCLM variables, including @@FLMPRJ, @@FLMGRP, and @@FLMUID, can be used when you specify the name of the accounting VSAM data sets.
2. Use the EXPACCT parameter on the FLMCNTRL and FLMALTC macros to specify the name of the export accounting data sets. This example illustrates how to use this parameter:

```
         FLMCNTRL ACCT=SCLM.ACCOUNT.DATABASE,                        C
                  EXPACCT=SCLM.EXPORT.ACCOUNT.DATABASE

SAMPLE   FLMALTC  ACCT=SCLM.ACCOUNT.SAMPLE,                          C
                  EXPACCT=SCLM.EXPORT.ACCOUNT.SAMPLE
```

3. VSAM attributes should match those used for the Accounting files, except for the SHAREOPTIONS, which must be SHAREOPTIONS(2,3).

Create the audit control data sets

The audit control data sets contain information about changes to SCLM-controlled members that are located in groups being audited. The audit control data sets are only required if the audit function is used. You must create the audit control data sets before the audit function is enabled. If auditing is used, each project must have at least one primary audit control data set.

You can create an optional secondary audit control data set. The secondary audit control data set is a backup for the primary audit control data set. It allows you to restore audit control information if the primary audit control data set is corrupted. Choose a unique name for this data set and put it on a different volume than the primary audit control data set. If a secondary data set is used, SCLM's performance will be degraded because updates are made to both the primary and secondary audit control data sets. The information in both data sets should be compared periodically to ensure the integrity of the audit control information.

Use the IDCAMS utility to define audit control data sets. Each audit control data set for the project must be a VSAM cluster. If audit control information for different groups will be kept in separate audit control data sets, you must create additional audit control data sets. The following JCL example defines audit control data sets.

Note: This example JCL is called FLM02VER and is in data set ISP.SISPSAMP that is included with SCLM.

```
//jobname  JOB (wkpkg,dpt,bin),'name'
//* code additional JOBCARD statements here
//*********************************************************************
//*
//*  THIS JCL EXAMPLE DEFINES A VSAM CLUSTER TO BE USED AS THE
//*    AUDIT CONTROL DATA SET FOR A GIVEN PROJECT.
//*  THE HIGH LEVEL QUALIFIER MUST BE AN ENTRY IN A VSAM CATALOG
//*    IN ORDER TO CREATE THIS CLUSTER.
//*  TO SPECIFY THE FILE, CHANGE THE DEFINE CLUSTER STATEMENT BELOW
//*    AS FOLLOWS:
//*
//*    1) ADD THE FOLLOWING LINE OF JCL TO DELETE THE VSAM CLUSTER
//*       BEFORE THE ALLOCATION IF THE DATA SET ALREADY EXISTS
//*       AND IT NEEDS TO BE DELETED:
//*       DELETE 'project.version.file' CLUSTER
//*       ADD THIS STATEMENT BETWEEN THE //SYSIN ALLOCATION AND THE
//*       DEFINE CLUSTER LINE OF JCL.
//*    2) CHANGE ALL project.version.file TO THE DESIRED FILE NAME.
//*        THIS VALUE IS SPECIFIED ON THE FLMCNTRL AND FLMALTC
//*        MACROS.  IF MORE THAN ONE VSAM ACCOUNTING DATA SET IS
//*        SPECIFIED ON THE FLMCNTRL AND FLMALTC MACROS, MULTIPLE
//*        IDCAMS DEFINE STEPS ARE REQUIRED.
//*    3) MODIFY CYLINDERS (PRIMARY SECONDARY)
//*    4) SPECIFY THE VOLUME VVVVVV ON WHICH IT WILL BE ALLOCATED
//*
//*  A JOB STEP IS THEN EXECUTED TO INITIALIZE THE FILE.
//*
//*********************************************************************
```

Figure 11. Audit Control Data Set Example (Part 1 of 2)

```
//STEP1   EXEC PGM=IDCAMS
//*
//SYSPRINT  DD SYSOUT=H
//*
//SYSIN     DD *
     DEFINE CLUSTER +
        (NAME('project.version.file') +
         CYLINDERS(4 1) +
         VOLUMES(VVVVVV) +
         KEYS(40 0) +
         RECORDSIZE(264 32000) +
         SHAREOPTIONS(4,3) +
         SPEED +
         SPANNED +
         UNIQUE) +
         INDEX(NAME('project.version.file.INDEX') -
         ) +
         DATA(NAME('project.version.file.DATA') -
         CISZ(2048) +
         FREESPACE(50 50) +
         )
/*//*******************************************************************
//*
//*  INITIALIZE THE AUDIT CONTROL FILE
//*
//*******************************************************************
//STEP2    EXEC PGM=IDCAMS
//INPUT  DD *
                        SCLM AUDIT CONTROL FILE INITIALIZATION RECORD
/*
//OUTPUT DD DSN=project.version.file,DISP=SHR
//SYSPRINT  DD SYSOUT=H
//SYSIN     DD *
     REPRO INFILE(INPUT) OUTFILE(OUTPUT)
/*
//*
)CM 5665-402 (C) COPYRIGHT IBM CORP 1980, 1989
```

Figure 12. Audit Control Data Set Example (Part 2 of 2)

Space considerations for the audit data sets

Each audit data set requires approximately one cylinder of 3390 DASD for every 100 partitioned data set members that SCLM controls. The space required varies depending on how much information SCLM will

control. If you require additional space in the data set, modify the space parameter (shown as CYLINDERS in the example JCL).

Create the Cross-dependency data set

The Cross-dependency data set is optional. It is only required if the Where-used function is to be used. Each project must have at least one cross-dependency data set for the Where-used function to be enabled. The VSAM attributes and space parameters should be the same as those used to define the accounting data set. Sample JCL for allocating and initialising the Cross-dependency data set can be found in member FLM02XDP of the ISP.SISPSAMP install library.

Create the SCLM control data set

The SCLM VSAM control data set is optional. It contains control information such as the SCLM administrators that have been defined to the SCLM project.

Use the IDCAMS utility to define the control data set. The control data set for the project must be a VSAM cluster. The JCL example shownhere defines the control data set.

Note: This example JCL is called FLM02CNT. It is stored in the ISPF sample library ISP.SISPSAMP.

```
//jobname  JOB (wkpkg,dpt,bin),'name'
//* code additional JOBCARD statements here
//*********************************************************************
//*
//*  THIS JCL EXAMPLE DEFINES A VSAM CLUSTER TO BE USED AS THE
//*    SCLM CONTROL DATA SET FOR A GIVEN PROJECT.
//*  THE HIGH LEVEL QUALIFIER MUST BE AN ENTRY IN A VSAM CATALOG
//*    IN ORDER TO CREATE THIS CLUSTER.
//*  TO SPECIFY THE FILE, CHANGE THE DEFINE CLUSTER STATEMENT BELOW
//*    AS FOLLOWS:
//*
//*    1) ADD THE FOLLOWING LINE OF JCL TO DELETE THE VSAM CLUSTER
//*       BEFORE THE ALLOCATION IF THE DATA SET ALREADY EXISTS
//*       AND IT NEEDS TO BE DELETED:
//*       DELETE 'project.control.file' CLUSTER
//*       ADD THIS STATEMENT BETWEEN THE //SYSIN ALLOCATION AND THE
//*       DEFINE CLUSTER LINE OF JCL.
//*    2) CHANGE ALL project.control.file TO THE DESIRED FILE NAME.
//*        THIS VALUE IS SPECIFIED ON THE FLMCNTRL MACRO
//*    3) MODIFY CYLINDERS (PRIMARY SECONDARY)
//*    4) SPECIFY THE VOLUME VVVVVV ON WHICH IT WILL BE ALLOCATED
//*
//*  A JOB STEP IS THEN EXECUTED TO INITIALIZE THE FILE.
//*
//*********************************************************************

//STEP1   EXEC PGM=IDCAMS
//*
//SYSPRINT  DD SYSOUT=H
//*
//SYSIN     DD *
     DEFINE CLUSTER +
        (NAME('project.control.file') +
         CYLINDERS(4 1) +
         VOLUMES(VVVVVV) +
         KEYS(26 0) +
         RECORDSIZE(264 32000) +
         SHAREOPTIONS(4,3) +
         SPEED +
         SPANNED +
         UNIQUE) +
         INDEX(NAME('project.control.file.INDEX') -
         ) +
         DATA(NAME('project.control.file.DATA') -
         CISZ(2048) +
         FREESPACE(50 50) +
         )
  //*
)CM 5665-402 (C) COPYRIGHT IBM CORP 2005
```

Figure 13. SCLM Control Data Set Example

Step 7: Protect the project environment

SCLM provides a controlled environment to maintain and track all software components. However, SCLM is not a security system. You must rely on RACF or an equivalent security system to provide complete environment security. Consider limiting authority to data sets in the hierarchy above the development layer.

The following sections describe the security requirements for the different types of data in the SCLM environment. Use this information to set up the security for the project environment. When this step is complete, the security requirements for the project environment are complete.

PROJDEFS data sets

The project definition LOAD data set should be restricted so that only the project manager has UPDATE authority to it. All other developers need READ access to this data set. Developers have no need to update the remaining PROJDEFS data sets and should not have UPDATE access to those data sets. READ access can be given to the other PROJDEFS data sets if this is reasonable for the project.

Project partitioned data sets

- Each developer needs READ authority to all the project partitioned data sets.
- Each developer needs UPDATE authority to the development groups that the individual uses to change SCLM-controlled members. UPDATE authority is also required for any groups the developer is allowed to promote into.
- If the SCLM versioning capability is used, each developer needs UPDATE authority to the versioning partitioned data sets.
- If the import/export capability is enabled, each developer needs UPDATE authority to the export data sets.
- We suggest that the project manager have ALTER authority to all the project partitioned data sets.

Control data sets

- Each developer in the project needs UPDATE authority to the control data sets that are updated by the developers.
- Each developer needs READ access to the primary and secondary (if used) accounting data sets for all groups in the hierarchy. This authorization is required for SCLM to perform its verification.
- If cross-reference data sets are used in the project, each developer needs READ access to the cross-reference data sets for all groups.
- If the auditing capability is used, each developer needs UPDATE authority to the audit control data sets.

For more information about RACF, refer to *z/OS Security Server RACF Command Language Reference*.

Step 8: Create the project definition

The project definition defines the development environment for an individual project. The project definition is organized into three parts: the hierarchy definition, project controls, and language definitions.

- The hierarchy definition determines the structure of the hierarchy and how data moves through the hierarchy.
- Project controls define how SCLM operates for the project.
- The language definitions define the languages for the project.

When creating a project definition, it is usually easier to copy a sample project definition and make the necessary project-specific modifications. The following project definitions are supplied in the ISPF sample library ISP.SISPSAMP:

- FLM@EXM1 uses several languages such as COBOL, PL/I, and Script.
- FLM@EXM2 shows several languages using Cross System Product, DB2, and IMS support.
- FLMWBPRJ includes languages that are used to build an application on your workstation using SCLM's workstation build capability.
- FLM01PRJ is used for the example scenario in “Project manager scenario” on page 39, and by the SCLM Sample Project dialog (see “Sample Project Utility (option 7)” on page 254).

Copy the project definition that is appropriate for your project, FLM@EXM1, FLM@EXM2 or FLMWBPRJ into your project.PROJDEFS.SOURCE data set. All project definitions and language definitions for your project should reside in your project.PROJDEFS.SOURCE data set.

Each part of the project definition uses SCLM macros to define the data so that SCLM understands it. The flexibility of these macros allows you to customize each project definition for specific purposes. Chapter 21, “SCLM macros,” on page 491 describes the use of these macros in detail.

Note: Because these are S/370 Assembler language macros, all rules pertaining to macros apply. In addition, there are some SCLM rules involving the use of the macros.

Alternate project definitions

You can generate more than one project definition for a project. Each project definition defines the relationships between groups in the project database and the processes that you can perform on the data in the project database. Each project definition can define a different database structure, specify different control options, or support different languages for the project.

Limit the use of alternate project definitions to satisfying a temporary need for a capability that the default (primary) project definition does not provide. You can use alternate project definitions successfully if they are never used to introduce or update members controlled under the primary project definition. Thus, you could use an alternate project definition to export data from the database definition or reference data in the primary database definition. However, if you use an alternate project definition to restrict an SCLM verification capability for data that is intended for the primary project definition, you can introduce integrity problems.

You can have an unlimited number of alternate project definitions for a project.

Figure 14 on page 25 shows an alternate project definition with a primary non-key integration group (DEPT) defined for the project database structure shown in Figure 4 on page 7.

```
PROJ1    FLMABEG
*
*
*    TYPE SPECIFICATION
*
ARCHDEF  FLMTYPE
DESIGN   FLMTYPE
LIST     FLMTYPE
LOAD     FLMTYPE
OBJ      FLMTYPE
SOURCE   FLMTYPE
*
*
*    GROUP SPECIFICATION, DEFINE THE AUTHORIZATION CODES
*
RELEASE  FLMGROUP AC=(REL),KEY=Y
TEST     FLMGROUP AC=(REL),KEY=Y,PROMOTE=RELEASE
INT      FLMGROUP AC=(REL),KEY=Y,PROMOTE=TEST
DEPT     FLMGROUP AC=(REL),KEY=N,PROMOTE=INT
USER1    FLMGROUP AC=(REL),KEY=Y,PROMOTE=DEPT
USER2    FLMGROUP AC=(REL),KEY=Y,PROMOTE=DEPT
USER3    FLMGROUP AC=(REL),KEY=Y,PROMOTE=DEPT
*
*
*    PROJECT CONTROLS
*
         FLMCNTRL ACCT=PROJ1.ACCOUNT.FILE,                       C
               MAXLINE=75
*
*
*    LANGUAGE DEFINITIONS
*
         COPY  FLM@ARCD     -- ARCHITECTURE     LANGUAGE    --
         COPY  FLM@TEXT     -- TEXT             LANGUAGE    --
         COPY  FLM@SCRP     -- SCRIPT 3         LANGUAGE    --
         COPY  FLM@ASM      -- 370 ASSEMBLER    LANGUAGE    --
         COPY  FLM@COBE     -- Enterprise COBOL LANGUAGE    --
         COPY  FLM@FORT     -- FORTRAN IV       LANGUAGE    --
         COPY  FLM@PSCL     -- PASCAL           LANGUAGE    --
         COPY  FLM@PLIE     -- Enterprise PL/I  LANGUAGE    --
         COPY  FLM@L370     -- 370 LINKAGE EDITOR           --
*
 FLMAEND
```

Figure 14. Sample Alternate Project Definition

Create the hierarchy definition

The hierarchy definition defines the project's hierarchy using groups and types. The rules for moving data within the hierarchy are defined with authorization codes. This information was created in Steps 1, 2, and 3. Modify the example project definition using the following macros and the information from Steps 1, 2, and 3 to define the hierarchy.

The macros that are used in the hierarchy definition are shown in the order that they are usually used in the project definition.

Specify the project name with FLMABEG

This macro defines the project name. It is required and must be the first macro in the project definition. You can use it only once. The project name must match the first qualifier of the PROJDEFS.LOAD data set.

If you want more than one project definition for a project, keep the project name in the alternate project definitions the same. See “Alternate project definitions” on page 24 for more information. In the example Figure 15 on page 30, the FLMABEG macro defines project PROJ1.

Define authorization groups with FLMAGRP

Use this macro to define a set (or group) of authorization codes. This macro is optional and needed only if you are defining a large number of authorization codes. You can use it multiple times.

The FLMAGRP provides a way of using an identifier to represent a list of authorization codes. If you decide to use multiple authorization codes for any of the groups in your hierarchy, it might be easier to associate an identifier with the list. If the list needs to be changed at a later date, the changes can be made on the FLMAGRP macros rather than changing the authorization code lists on all the FLMGROUP macros. The FLMAGRP macro must appear before any reference to the authorization group that it defines. The example Figure 15 on page 30 uses only one authorization code and therefore does not need to use FLMAGRP macros.

Define types with FLMTYPE

Use this macro to define one type in the project hierarchy. At least one occurrence of this macro is required. You can use it multiple times.

Define the types identified in “Step 2: Identify the types of data to support” on page 8 using the FLMTYPE macro. For example, in the sample project definition depicted in Figure 15 on page 30, type ARCHDEF is defined to contain architecture members.

Define groups with FLMGROUP

Use this macro to define one group in the project hierarchy. At least one occurrence of this macro is required. You can use it multiple times.

Define the groups identified in “Step 1: Determine the project's hierarchy” on page 4 using the FLMGROUP macro. Each group in the hierarchy requires an FLMGROUP statement.

The authorization codes defined in “Step 3: Establish authorization codes” on page 8 must also be defined now. Use the AC parameter on the FLMGROUP macro to define the authorization codes listed in “Step 3: Establish authorization codes” on page 8. The example Figure 15 on page 30 shows a project definition with only one authorization code defined.

End the definition with FLMAEND

This signifies the end of the project definition. It must be the last macro in the project definition and is required. You can use it only one time.

Set the project control options

The project control options dictate SCLM processing for an individual project. When this step is complete, the project controls of the project definition will be set up for the new project. Use project control options to specify:

- Primary accounting data set
- Secondary accounting data set
- Export accounting data set
- Audit control data set
- Cross-dependency data set
- Cross-dependency dynamic update
- VSAM Record Level Sharing
- Versioning partitioned data set
- Project partitioned data set naming conventions
- Maximum lines per page
- Number of versions to keep
- Translator option override
- Member level locking
- SCLM temporary data set allocation
- User exit routines

The following macros that can be used in the control section of the project definition are shown in the order that they are usually used in the project definition:

FLMCNTRL
Use this macro to specify project-specific control options. The options on FLMCNTRL apply to the entire project. This macro is optional unless you change any of SCLM's default control options. You can use it one time.

FLMALTC
Use this macro to provide alternate control for individual groups. This macro is used to override certain options on the FLMCNTRL macro for specific groups. The options on the FLMALTC macro apply only to the groups using it. This macro is optional. You can use it multiple times.

FLMATVER
Use this macro to enable the audit and version capability and to define the type of data, (audit or audit and versioning, to capture with the capability. If a project is using the versioning capability, it must also use the audit capability. This macro is optional. You can use it multiple times.

Primary accounting data set specification

The ACCT control option specifies the name of the primary accounting data set. The data set you specify must be the name of the VSAM cluster you want to use. The default accounting cluster name is `project.ACCOUNT.FILE`, where `project` is the 8-character name for the project.

In the example of a project definition, Figure 15 on page 30, the primary accounting data set name is `PROJ1.ACCT.FILE`.

Secondary accounting data set specification

The ACCT2 control option specifies the name of a backup VSAM accounting data set for the project. If a severe problem occurs with the primary accounting data set, you could use this backup data set to restore the primary accounting information.

If you use this option, additional VSAM updates to the secondary accounting data set take place and can affect SCLM's performance.

Export accounting data set specification

The EXPACCT control option specifies the name of the export accounting data set. The data set you specify must be the name of the VSAM cluster you want to use. The following variables can be used in specifying the name of the export accounting data set name:

- @@FLMPRJ
- @@FLMGRP
- @@FLMUID

The EXPACCT control option must have a data set name that is different from the ACCT or ACCT2 control option specified in FLMCNTRL or any FLMALTC macro.

The example project definition, Figure 15 on page 30 does not specify an export accounting data set.

Audit control data sets specification

The primary and secondary audit control data sets are optional. They only need to be specified if SCLM's auditing capability will be used. The VERS and VERS2 control options are used to specify the audit control data sets created in “Step 6: Allocate and create the control data sets” on page 17. The VERS control option specifies the primary audit control data set. The VERS2 control option specifies the secondary audit control data set that is a backup for the primary audit control data set. The FLMALTC macro can be used to specify different audit control data sets on specific groups.

Cross-dependency data set specification

The Cross-dependency data set is optional. It is only required if the Where-used function is to be used. The XDEP control option is used to specify the name of the Cross-dependency data set. The FLMALTC macro can be used to specify different Cross-dependency data sets on specific groups.

Cross-dependency data set dynamic update specification

The XDEPDYN control option is used to control the dynamic updating of the Cross-dependency data set. If XDEPDYN is set to Y, then the Cross-dependency data set is kept in sync with changes to the accounting data set. XDEPDYN may be set to N if the extra I/O activity is causing performance problems. The XDEPUPDT service should then be used to synchronise the Cross-dependency data set with the accounting data set.

VSAM Record Level Sharing (RLS)

The VSAMRLS control option indicates whether VSAM Record Level Sharing should be used. The default value is NO. The example found in this chapter does not use VSAM Record Level Sharing.

Versioning partitioned data sets specification

Specifying the names of versioning partitioned data sets is optional. The VERPDS control option allows you to specify the names of partitioned data sets that will contain the versioned data for a project. If the names of the versioning partitioned data sets will be different for specific groups, the FLMALTC macro must be used to associate the names of the versioning partitioned data sets with the specific groups. The following variables can be used in specifying the name of the versioning partitioned data set name:

- @@FLMPRJ
- @@FLMGRP
- @@FLMTYP
- @@FLMDSN

Project partitioned data set naming conventions

The DSNAME control option is used to specify a naming convention other than the SCLM default for the project partitioned data sets. The DSNAME option allows the project manager to specify the naming convention for all the data sets in the hierarchy. If the naming convention of the project partitioned data sets will be different for specific groups then the FLMALTC macro must be used so the naming convention for the data sets associated with the specific groups will be changed. For more information about modifying the naming convention for project partitioned data sets see “Flexible naming of project partitioned data sets” on page 12.

Maximum lines per page

Use the MAXLINE control option to specify the maximum lines per page for all SCLM-generated reports. The default is 60. The minimum number of lines per page is 35. In the example project definition Figure 15 on page 30, the maximum number of lines per page defaults to 60.

Number of versions to keep

Use the VERCOUNT parameter to specify how many versions of a member to keep. The default value of zero, used in the example found in this chapter, indicates that all versions are kept. The number of versions specified using this parameter applies to all types that are versioned. The VERCOUNT parameter on the FLMATVER macro can be used to override this value for specific types.

Valid values are 0 and any integer value greater than or equal to 2. Because that is what is already in the hierarchy, 1 is not a valid value. If you specify a value other than the default and you intend to version multiple groups in the hierarchy, either use the FLMALTC macro to specify different VERPDS data sets for each group or use the @@FLMGRP variable in the VERPDS name on the FLMALTC macro. Failure to allocate and specify unique VERPDS data sets can result in difficulty retrieving versions.

Translator option override

The OPTOVER control option allows you to keep developers from overriding project-defined translator options. If you specify Y, developers can override the translator options for any of the languages by using the PARM statement in the architecture members. For more details on specifying translator options in architecture members, see Chapter 11, “Architecture definition,” on page 271.

If you specify N, SCLM uses only translator options you specify in the language definition for the translators. Specifying N also overrides the OPTFLAG parameter, which allows option override by the translator. The default for the OPTOVER control option is Y. In the example project definition Figure 15 on page 30, the OPTOVER option defaults to Y.

Member level locking

Member level locking allows SCLM administrators to stop users from modifying members that belong to other users. To implement member level locking, perform the following steps:

- Change the FLMCNTRL macro in the project definition to specify the parameters MEMLOCK=Y, CONTROL, and ADMINID.
- Reassemble the project definition.

The user you specify in the ADMINID parameter will be the default SCLM administrator and will be able to add other SCLM administrators using the SCLM Admin option. See “Maintaining SCLM administrators (option A)” on page 255 for more information.

When member level locking is enabled, you will be able to edit a member if any of the following conditions is true:

- You are the default SCLM administrator (ADMINID parameter).
- You are defined as an SCLM administrator (option A).
- The accounting record doesn't exist at the development level.
- The accounting record exists at the development level and your user ID matches the change user ID on the accounting level.

If another user needs to modify the member, either the SCLM administrator or the user who last updated the member (Change User ID on the accounting record) can transfer ownership using the Transfer option in option 3.1. See “Transfer ownership” on page 174.

SCLM temporary data set allocations

Many installations specify one or more I/O unit names as Virtual Input Output (VIO) devices at system generation time. Use of these devices typically improves system performance by eliminating much of the overhead and time required to move data physically between main storage and an I/O device.

To take advantage of this facility, specify the name of the VIO unit in your project definition as the VIOUNIT parameter on the FLMCNTRL macro. This unit will be used for all temporary data sets under the following conditions:

- IOTYPE = O, P, S, or W
- CATLG = N
- RECNUM <= the MAXVIO parameter.

Some of the temporary data sets used by versioning will use the VIO unit as well as long as the size of the temporary data set to be allocated is less than or equal to the MAXVIO value.

Temporary data set allocations that fail to meet any of the preceding conditions will be allocated using the unit specified via the DASDUNIT parameter on the FLMCNTRL macro.

The default value for MAXVIO is 5000, and the maximum allowable value is 2147483647. A relatively large value should be specified in order to ensure that SCLM temporary data sets are allocated using the VIO unit. If SCLM functions fail for lack of memory (S80A ABEND or S878 ABENDs), try reducing this value.

The size of the temporary data sets allocated for translators is determined by the attributes specified on the FLMALLOC macros in the language definition. The size of the temporary data sets used by versioning is based on the attributes of the source data set being versioned.

User exit routine specification

SCLM provides a number of exit points that you can use to customize SCLM processing or to integrate SCLM with other products. You can specify your own user exit routines in the project definition using the user exit parameters on the FLMCNTRL macro. See Chapter 2, “User exits,” on page 53 for more information.

SCLM includes a sample user exit for use with Tivoli® Information Management. See Chapter 6, “Using SCLM and Tivoli Information Management for z/OS,” on page 127 for more information.

Example project definition

Figure 15 on page 30 shows an example of a project definition. The source for this example can be found in the ISPF sample library, ISP.SISPSAMP, member FLM@EXM1.

```
         TITLE '*** PROJECT DEFINITION FOR PROJECT=PROJ1 ***'
PROJ1    FLMABEG
*
*        ************************************************************
*        *  DEFINE THE AUTHORIZATION CODES                          *
*        ************************************************************
GRP1     FLMAGRP AC=(A1,B1,C1)
GRP2     FLMAGRP AC=(A2,B2,C2)
GRPALL   FLMAGRP AC=(GRP1,GRP2)
*
*        ************************************************************
*        *  DEFINE THE TYPES                                        *
*        ************************************************************
*
ARCHDEF  FLMTYPE EXTEND=SOURCE
COMP     FLMTYPE
DICT     FLMTYPE
DOCS     FLMTYPE
IDILANGX FLMTYPE
LINKLIST FLMTYPE
LIST     FLMTYPE
LMAP     FLMTYPE
LOAD     FLMTYPE
OBJ      FLMTYPE
OBJ1     FLMTYPE
OBJ2     FLMTYPE
SCRIPT   FLMTYPE EXTEND=SOURCE
SOURCE   FLMTYPE
*
*        ************************************************************
*        *  DEFINE THE GROUPS                                       *
*        ************************************************************
*
DEV1     FLMGROUP AC=(GRP1),KEY=Y,PROMOTE=TEST
DEV2     FLMGROUP AC=(GRP2),KEY=Y,PROMOTE=TEST
TEST     FLMGROUP AC=(GRP1),KEY=Y,PROMOTE=RELEASE
RELEASE  FLMGROUP AC=(GRPALL),KEY=Y,ALTC=RELDB
*
```

Figure 15. Example Project Definition (Part 1 of 3)

```
*********************************************************************
*                PROJECT CONTROLS
*********************************************************************
*
        FLMCNTRL ACCT=PROJ1.ACCT.FILE,                                C
               VERS=PROJ1.VER1.FILE,                                  C
               VERS2=PROJ1.VER2.FILE,                                 C
               MAXVIO=999999,                                         C
               VIOUNIT=VIO
*
RELDB   FLMALTC ACCT=PROJ1.ACCT.FILEX,                                C
               VERS=PROJ1.VER1.FILEX,                                 C
               VERS2=PROJ1.VER2.FILEX
*
*********************************************************************
*                VERSIONING AND AUDITABILITY                        *
*********************************************************************
*
*
        FLMATVER GROUP=TEST,                                          C
               TYPE=SOURCE,                                           C
               VERSION=YES
*
        FLMATVER GROUP=RELEASE,                                       C
               TYPE=SOURCE,                                           C
               VERSION=YES

*********************************************************************
*                LANGUAGE DEFINITION TABLES
*********************************************************************
*
*
*********************************************************************
* NON-COMPILERS
*********************************************************************
*
         COPY  FLM@ARCD           -- ARCHITECTURE DEF. LANGUAGE --
         COPY  FLM@CLST           -- CLIST             LANGUAGE --
         COPY  FLM@REXX           -- REXX              LANGUAGE --
         COPY  FLM@REXC           -- REXX PARSER AND COMPILER   --
         COPY  FLM@TEXT           -- TEXT              LANGUAGE --
         COPY  FLM@SCRP           -- SCRIPT 3          LANGUAGE --
         COPY  FLM@BOOK           -- SCRIPT/BOOKMASTER LANGUAGE --
*
*********************************************************************
* REXX PARSERS WITH STANDARD COMPILERS
*********************************************************************
*
         COPY  FLM@RASM           -- 370 ASSEMBLER H   LANGUAGE --
         COPY  FLM@RC37           -- 370 C             LANGUAGE --
         COPY  FLM@RCBL           -- COBOL II          LANGUAGE --
*
```

Figure 16. Example Project Definition (Part 2 of 3)

```
************************************************************************
* STANDARD COMPILERS USING SYSTEM MACRO LIBRARIES
************************************************************************
*
COBOL    FLMSYSLB SYS1.EXAMPLE.MACROS
         COPY  FLM@ASM            -- 370 ASSEMBLER     LANGUAGE --
         COPY  FLM@ASMH           -- 370 ASSEMBLER H   LANGUAGE --
         COPY  FLM@C370           -- 370 C             LANGUAGE --
         COPY  FLM@CPLK           -- 370 C + PRE-LINK  LANGUAGE --
         COPY  FLM@CLNK           -- 370 C PRE-LINK/LINK-EDIT   --
         COPY  FLM@COBL           -- COBOL             LANGUAGE --
         COPY  FLM@COB2           -- COBOL II          LANGUAGE --
         COPY  FLM@COBE           -- Enterprise COBOL   LANGUAGE --
         COPY  FLM@FORT           -- FORTRAN IV        LANGUAGE --
         COPY  FLM0HLAF           -- HIGH LEVEL ASSEM. LANGUAGE --
*                                 -- WITH FAULT ANALYSER        --
         COPY  FLM@HLAS           -- HIGH LEVEL ASSEM. LANGUAGE --
         COPY  FLM@PSCL           -- PASCAL            LANGUAGE --
         COPY  FLM@PLIC           -- PL/I CHECKOUT     LANGUAGE --
         COPY  FLM@PLIO           -- PL/I OPTIMIZER    LANGUAGE --
         COPY  FLM@PLIE           -- Enterprise PL/I   LANGUAGE --
*
************************************************************************
* LANGUAGE DEFINITIONS TO SUPPORT OBJ AND LOAD WITHOUT SOURCE
************************************************************************
*
         COPY  FLM@OBJ            -- DUMMY LANG DEF TO MIGRATE OBJ --
         COPY  FLM@COPY           -- COPY OBJ TO OUTPUT TYPE    --
*
************************************************************************
* LINKAGE EDITORS                                                     *
************************************************************************
*
         COPY  FLM@L370           -- 370 LINKAGE EDITOR         --
*
************************************************************************
*
         FLMAEND
*
* 5694-A01     COPYRIGHT IBM CORP 1992, 2007
```

Figure 17. Example Project Definition (Part 3 of 3)

Define the language definitions

Language Definitions define the languages and translators that a project uses. SCLM functions invoke translators (such as compilers, parsers, and linkage editors) based on a member's language. The language definition defines the translators used by each language. Each language can have multiple translators defined for it. The translators can be IBM program products, independent program products, or user-written translators.

IBM provides examples of language definitions for many commonly used languages such as COBOL and PL/I.

Table 5. Language Definitions Supplied with SCLM

Compilers and Linkage Editors	Language Definitions
Architecture definition	FLM@ARCD (noncompiler)
BookMaster®	FLM@BOOK (noncompiler)
CICS® map groups	FLM@BMS
CLIST	FLM@CLST (noncompiler)
COBOL OS/VS	FLM@COBL
COBOL OS with CICS preprocessing	FLM@CCOB

Table 5. Language Definitions Supplied with SCLM (continued)

Compilers and Linkage Editors	Language Definitions
COBOL OS with DB2 preprocessing	FLM@2COB
COBOL OS with DB2 and CICS preprocessing	FLM@ECOB
COBOL II	FLM@COB2
COBOL II with CICS preprocessing	FLM@CICS
COBOL II with DB2 preprocessing	FLM@2CO2
COBOL II with DB2 and CICS preprocessing	FLM@ECO2
COBOL II with member expansion and CICS preprocessing	FLM@ICO2
Enterprise COBOL	FLM@COBE
COBOL with integral CICS preprocessing	FLM@CCBE
Enterprise COBOL with integral DB2 preprocessing	FLM@2CBE
Enterprise COBOL with integral DB2 and CICS preprocessing	FLM@2CCE
COBOL	FLM@RCBL (COBOL parser written in REXX)
C/C++ for MVS	FLM@RCIS (C/C++ parser written in REXX)
C/370	FLM@C370, FLM@RC37 (C/370 parser written in REXX)
C/370 with CICS preprocessing	FLM@CC
C/370 with DB2 preprocessing	FLM@2C
C/370 with DB2 and CICS preprocessing	FLM@EC
C/370 with member expansion and CICS preprocessing	FLM@IC
C/370 with pre-link	FLM@CPLK
C/370 pre-link with link-edit	FLM@CLNK
DB2	See Table 24 on page 298
Enterprise COBOL compiler with integral DB2 preprocessing and Fault Analyzer side file generation.	FLM@2CBF
DB2 and PL/I enterprise compiler and NCAL linkedit to a sub-module library with Fault Analyzer side file generation.	FLM@2PLF
FORTRAN IV	FLM@FORT
FORTRAN IV with DB2 preprocessing	FLM@2FRT
JOVIAL	FLM@JOV FLM@JOVC
Object language definition to migrate object modules into SCLM as outputs (non-editable)	FLM@COPY

Table 5. Language Definitions Supplied with SCLM (continued)

Compilers and Linkage Editors	Language Definitions
Object/Load dummy language definition to migrate object and load into SCLM as inputs (editable)	FLM@OBJ
Pascal	FLM@PSCL
PL/I Checkout Compiler	FLM@PLIC
PL/I Optimizer with DB2 preprocessing	FLM@2PLO
PL/I Optimizing Compiler	FLM@PLIO
PL/I Optimizer with CICS preprocessing	FLM@CPLO
PL/I Optimizer with DB2 and CICS preprocessing	FLM@EPLO
PL/I Optimizer with member expansion and CICS preprocessing	FLM@IPLO
Enterprise PL/I	FLM@PLIE
Enterprise PL/I with integral CICS preprocessing	FLM@CPLE
Enterprise PL/I with integral DB2 preprocessing	FLM@2PLE
REXX	FLM@REXX (noncompiler) FLM@REXC (compiler)
Language Parsers written in REXX	FLM@RASM (Assembler), FLM@RCBL (COBOL), FLM@RC37 (C/370), FLM@RCIS (C/C++ for MVS)
SCRIPT 3	FLM@SCRP (noncompiler)
S/370 Assembler F	FLM@ASM
S/370 Assembler with DB2 preprocessing	FLM@2ASM
S/370 Assembler with CICS preprocessing	FLM@ASMC
S/370 Assembler with DB2 and CICS preprocessing	FLM@EASM
S/370 Assembler with member and CICS preprocessing	FLM@IASM
S/370 Assembler H	FLM@ASMH
High Level Assembler for MVS	FLM@HLAS, FLM@RASM (Assembler parser written in REXX)
S/370 Linkage Editor	FLM@L370
TEXT	FLM@TEXT (noncompiler)

All the example language definitions are located in the data set ISP.SISPMACS.

The ISPF Sample and Macro libraries contain a number of other files to support SCLM workstation builds. See “ISPF Sample and Macro libraries” on page 311.

This step describes how to define language definitions to the project definition. When this step is complete, all the languages your project will use will be defined.

To define the language definitions:

1. Determine what languages are used in your project.
2. Copy the appropriate example language definitions to the project.PROJDEFS.SOURCE data set allocated in "Step 4: Allocate the PROJDEFS data sets" on page 11.
3. Modify the language definitions.

   If you do not find an example language definition that meets your project requirements, you can write a new language definition. For instructions on defining a new language to SCLM, see "Defining a new language to SCLM" on page 101.

See Chapter 21, "SCLM macros," on page 491 for details on the use of each SCLM macro.

Modifying example language definitions

Use the following macros to modify language definitions for specific project requirements.

Table 6. SCLM Macros for Language Definition

Macro	Purpose
FLMSYSLB	Use this macro to define data sets that contain system, project, or language dependencies that are referenced by SCLM members but are not in the SCLM hierarchy themselves. Examples are system macros for Assembler programs and compiler-supplied include files for C programs.
FLMLANGL	Use this macro to define the language to SCLM.
FLMINCLS	Use this macro to associate sets of includes found during the parse of a member with the types in the project definition that contain those includes. FLMALLOC macros then reference this macro to allocate the include libraries for build translators. The FLMINCLS macro can be used multiple times for each language, but each FLMINCLS macro must have a unique name within the language and be associated with at least one FLMALLOC macro. This helps ensure that the includes that are found by build are the same ones found by the translators.
FLMLRBLD	Use this macro to tell SCLM to automatically rebuild members with this language after they are promoted into the listed groups.
FLMTRNSL	Use this macro to define a translator for a language. It can be used multiple times for a language.
FLMTOPTS	Use this macro to vary the options passed to a build translator based on the group where the build is taking place. Options can be appended to the existing options or replace the options completely. FLMTOPTS macros must follow an FLMTRNSL macro with FUNCTN=BUILD.
FLMTCOND	Use this macro to specify conditional execution of a BUILD translator. Part of the specification can include examination of return codes from previous BUILD translators in the language definition.
FLMALLOC	Use this macro for each data set allocation required by a translator. If you are using a ddname substitution list, specify an FLMALLOC macro for each ddname in the correct order. If not, determine the ddnames that are needed by the translator and specify an FLMALLOC macro for each ddname.
FLMCPYLB	Use this macro to identify data sets to be concatenated to a ddname. The data sets must be preallocated. The FLMCPYLB data sets are used as input to the Parse and other translators.

For each language, take the following actions as necessary:

- Specify data sets containing dependencies that are not to be tracked, such as assembler system macros (macro FLMSYSLB).
- Specify the maximum number of includes, change codes, user data records, compilation units, and external dependencies expected in a source member (macro FLMLANGL; keyword BUFSIZE).
- Determine if ddname substitution is needed for the translator. This information can be found in the translator documentation. Adjust the PORDER parameter on the FLMTRNSL macro as needed.
- Verify translator load module names and load data sets for accuracy (macro FLMTRNSL; keywords COMPILE, DSNAME, and TASKLIB).
- Adjust translator return codes to project requirements if nonzero return codes are acceptable (macro FLMTRNSL; keyword GOODRC).
- Update default translator options (macro FLMTRNSL; keyword OPTIONS).
- Verify translator version information (macro FLMTRNSL; keyword VERSION).
- Specify output listings (macro FLMALLOC; keyword PRINT).
- Specify output default types (macro FLMALLOC; keyword DFLTTYP) to match the FLMTYPE type specified in the project definition.
- Verify that system libraries are being allocated for build translators. Either specify ALCSYSLB=Y on the FLMLANGL macro or ensure that the data sets from FLMSYSLB macros are specified on FLMCPYLB macros following IOTYPE=I allocations.
- Specify the include sets for the language to use. You must specify all the include-sets returned by the parser for the language. If you add a new FLMINCLS macro, ensure that it is referenced by at least one FLMALLOC of a build translator. If you remove an FLMINCLS macro, update any FLMALLOC macros that reference it, ensuring that no member's accounting data contains references to that include set.

Figure 18 on page 37 shows an example of an Enterprise COBOL language definition.

```
*********************************************************************
*                                                                   *
*            Enterprise COBOL LANGUAGE DEFINITION FOR SCLM          *
*********************************************************************
*
         FLMLANGL    LANG=COBE                                       C
               LANGDESC='ENTERPRISE COBOL',                          C


*********************************************************************
*         --PARSER TRANSLATOR--                                     *
*********************************************************************
*
          FLMTRNSL   CALLNAM='SCLM COBOL PARSE',                     C
               FUNCTN=PARSE,                                         C
               COMPILE=FLMLPCBL,                                     C
               PORDER=1,                                             C
               CALLMETH=LINK,                                        C
               OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,)
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
*********************************************************************
*         --ENTERPRISE COBOL INTERFACE--                            *
*********************************************************************
*
          FLMTRNSL   CALLNAM='ENTERPRISE COBOL COMPILER',            C
               FUNCTN=BUILD,                                         C
               COMPILE=IGYCRCTL,                                     C
               DSNAME=IGY,SIGYCOMP,                                  C
               VERSION=3.1,                                          C
               GOODRC=0,                                             C
               PORDER=1,                                             C
               OPTIONS=(XREF,LIB,APOST,NODYNAM,LIST,NONUMBER,NOSEQ)
*
*********************************************************************
*         --DDNAME ALLOCATION--                                     *
*********************************************************************
*
 FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,                        C
     RECNUM=5000,DFLTTYP=OBJ
*
 FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
*
 FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000
*
 FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,RECNUM=5000
*
 FLMALLOC  IOTYPE=W,DDNAME=SYSUT2,RECNUM=5000
*
 FLMALLOC  IOTYPE=W,DDNAME=SYSUT3,RECNUM=5000
*
 FLMALLOC  IOTYPE=W,DDNAME=SYSUT4,RECNUM=5000
*
 FLMALLOC  IOTYPE=A,DDNAME=SYSUT5,RECNUM=5000
*
 FLMALLOC  IOTYPE=A,DDNAME=SYSUT6,RECNUM=5000
*
 FLMALLOC  IOTYPE=A,DDNAME=SYSUT7,RECNUM=5000
```

Figure 18. Enterprise COBOL Language Definition Example (Part 1 of 2)

```
*
 FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
  FLMCPYLB NULLFILE
*
 FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
  FLMCPYLB NULLFILE
*
 FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,KEYREF=LIST,                       C
     RECFM=FBA,LREC=133,                                               C
     RECNUM=50000,PRINT=Y,DFLTTYP=LIST
```

Figure 19. Enterprise COBOL Language Definition Example (Part 2 of 2)

In the example in Figure 18 on page 37, the COBOL language is defined to SCLM by the FLMLANGL macro. The FLMTRNSL parameters specify particular information about the compiler:

- The name of the compiler: ENTERPRISE COBOL.
- The name of the compiler load module: IGYCRCTL.
- The version of the compiler: 3.3.1.
- The compiler options: XREF, LIB, APOST, NODYNAM, LIST, NONUMBER, NOSEQ.

The FLMALLOC macros following the build FLMTRNSL macro specify each ddname needed by the COBOL compiler. SCLM allocates the ddnames specified on the FLMALLOC macro before invoking the translator (in this example, the COBOL IKFCBL00 load module). The FLMALLOC parameters allow specification of the record format (RECFM), the logical record length (LRECL), the number of records (RECNUM), and other options. An FLMCPYLB macro specifies that a ddname be associated with a null data set.

The language definitions must be defined to the project definition, either by placing the language definitions directly into the project definition or having the language definitions copied into the project definition when the project definition is assembled. It is easier to maintain the project definition if each language definition is kept in a separate member and copied into the project definition when the project definition is assembled. The example project definition Figure 15 on page 30 uses this method of including the language definitions.

Step 9: Assemble and link the project definition

Assemble all project definitions with the SCLM macro set using the standard IBM S/370 Assembler. Once assembled, link the object code using the standard IBM S/370 linkage editor and store the load module into the project.PROJDEFS.LOAD data set. All project definitions must reside in the project.PROJDEFS.LOAD data set to allow SCLM to be invoked correctly. SCLM accesses the project definition's load module when SCLM is invoked. If the project definition is updated, reassembled, and relinked while the current load module is being used, the active invocation of SCLM will not be affected.

Make sure all project definition load modules are reentrant. Nonreentrant project definition load modules can cause error conditions. Specify the RENT option during link-edit. The load module name of the default project definition for a project must match the project identifier specified on the FLMABEG macro. Alternate project definitions can have any load module name, but all alternate project definitions must have the same project identifier, specified on the FLMABEG macro, as the default project definition.

The SCLM macro set performs some verification of the project definition during assembly. When warning or error conditions are detected, the macros issue MNOTES, which are SCLM-specific diagnostic comments. The MNOTES produced by SCLM are listed in *z/OS ISPF Messages and Codes*. If the text of an MNOTE is missing, verify that the FLMABEG macro appears at the top of the project definition and is referenced correctly. Here are the return codes from the assembler:

0
The SCLM macros detected no errors.

4
The SCLM macros detected a potential error. The project definition might be valid, but might not reflect the desired options. Review the assembler listing for details.

8
The SCLM macros detected errors. Do not use the project definition until you correct the errors identified in the assembler listing.

Other
The assembler detected errors. Examine the assembler listing for the error messages and consult the assembler's user guide for additional information. Do not use the project definition until you correct the errors identified in the assembler listing.

Assemble and link example

The following example illustrates JCL that assembles and links a project definition. This example can be found in member FLM02PRJ in the data set ISP.SISPSAMP.

```
//jobname  JOB (wkpkg,dpt,bin),'name'
//* code additional JOBCARD statements here
//*
//ASMPROJ  PROC  PROJID=,PROJDEF=
//*--------------------------------------------------------------------*
//*   ASSEMBLE AND LINK A PROJECT DEFINITION                           *
//*                                                                    *
//*   PROC PARAMETERS:                                                 *
//*                                                                    *
//*      PROJID  -  HIGH-LEVEL QUALIFIER FOR PROJECT                   *
//*      PROJDEF -  PROJECT DEFINITION MEMBER NAME                     *
//*                                                                    *
//*   NOTE: MODIFY SYSLIB DSNAMES TO GET THE SCLM RELEASE MACROS       *
//*   AND ANY LANGUAGE DEFINITIONS YOU NEED.                           *
//*--------------------------------------------------------------------*
//ASM      EXEC PGM=ASMA90,REGION=4000K,PARM=OBJECT
//SYSLIB   DD DSN=&PROJID..PROJDEFS.SOURCE,DISP=SHR
//         DD DSN=ISP.SISPMACS,DISP=SHR
//SYSPRINT DD SYSOUT=H
//SYSPUNCH DD DUMMY
//SYSIN    DD DSN=&PROJID..PROJDEFS.SOURCE(&PROJDEF),DISP=SHR
//SYSUT1   DD UNIT=SYSALLDA,SPACE=(CYL,(2,2))
//SYSLIN   DD DSN=&&INT,DISP=(,PASS),UNIT=SYSDA,SPACE=(CYL,(5,5,0)),
//             DCB=(BLKSIZE=400)
//*--------------------------------------------------------------------*
//LINK     EXEC PGM=IEWL,PARM='RENT,LIST,MAP',REGION=512K
//SYSPRINT DD SYSOUT=H
//SYSLIN   DD DSN=&&INT,DISP=(OLD,DELETE)
//SYSUT1   DD UNIT=SYSALLIDA,SPACE=(CYL,(2,2)),DISP=NEW
//SYSLMOD  DD DISP=SHR,DSN=&PROJID..PROJDEFS.LOAD(&AMP;PROJDEF)
//         PEND
//*--------------------------------------------------------------------*
//ASMLINK  EXEC  PROC=ASMPROJ,PROJID=SCLM,PROJDEF=SCLM
//
```

Project manager scenario

This section describes the steps required to define and install an SCLM project. By completing the steps outlined in the following sections, the project manager can create a project that is under SCLM control. The sample project can also be defined using the SCLM sample project utility (Option 10.7). Once the project has been created, it can be used as a model for building other SCLM projects.

The project manager must perform all the steps described in this chapter before developers can follow the programmer scenario described in Chapter 10, “Development scenario,” on page 257.

Prerequisites for defining an SCLM project

Before beginning the project definition phase of this activity, you must have the following software, space, and tools available:

- ISPF with SCLM installed on a z/OS system.
- Enterprise PL/I for z/OS IBMZPLI Version 3.3.0 or equivalent.
- Enterprise COBOL for z/OS IGYCRCTL Version 3.3.1 or equivalent.

- Fault Analyzer. Version 6.1 or equivalent.
- Disk space to contain the data sets for the project. The project requires 265 tracks on 3390 DASD.
- Access to data set `ISP.SISPSAMP`.

  This data set is available as part of the ISPF product. It contains the project definition for this scenario and other examples. Check with the person at your site who installs ISPF to find out the name of this data set and how to allocate it.

  The member FLM01PRJ in this data set is the definition for the sample project definition used for this scenario.
- Access to data set `ISP.SISPMACS`.

  This macro library is included with ISPF and contains the macros used to assemble the project definition.
- ISPF knowledge at the user level (edit and utilities are used).
- VSAM installed.
- Basic VSAM knowledge. (Not required if defining the project with the SCLM Sample Project utility.

Example project overview

This SCLM project contains all the required components of SCLM projects in general and serves as a model for future projects. A description of the components of the project follows.

Figure 20 on page 40 shows three layers in the SCLM project hierarchy: development, test, and release.

- The development layer promotes to the test layer, and the test layer promotes to the release layer.
- The development layer is composed of the groups DEV1 and DEV2. You can think of these groups as being assigned to two separate developers. The SCLM hierarchy looks like Figure 20 on page 40.

Figure 20. Example Project Hierarchy

Figure 21 on page 42 shows nine modules in the hierarchy: FLM01AD9, FLM01CD7, FLM01CD8, FLM01MD1, FLM01MD2, FLM01MD3,FLM01MD4, FLM01MD5, and FLM01MD6. These are the programs that the developers edit in order to install fixes and new features.

- FLM01MD2 is written in PL/I and uses the Enterprise PL/I compiler. FLM01MD2 includes the FLM01IIN copybook.

  Note: Module FLM01MD2, copybook FLM01IIN, and the language definition for the Enterprise PL/I Compiler, are not included if the project is defined using the SCLM sample project utility and the first optional compiler field was not selected.
- Module FLM01AD9 is written in High Level Assembler. FLM01AD9 includes the FLM01INC copybook.

  Note: Module FLM01AD9 and copybook FLM01INC will be included if the project is defined using the SCLM sample project utility, but no side file will be generated if the Fault Analyzer field is not selected.
- FLM01CD7 and FLM01CD8 are written in COBOL and use the Enterprise COBOL compiler. FLM01CD7 includes the FLM01CIN copybook. FLM01CD8 includes the FLM01CDT copybook.

Note: Modules FLM01CD7, FLM01CD8, copybooks FLM01CIN, FLM01CDT and the language definition for the Enterprise COBOL Compiler, are not included if the project is defined using the SCLM sample project utility and the second optional compiler field was not selected

- The other modules are written in S/370 Assembler. They include a member named FLM01EQU that contains the register equates commonly used in assembly language programs.
- The modules are compiled or assembled by the BUILD function into an application named FLM01API. SCLM performs this operation using the architecture definitions contained in the ARCHDEF data sets.
- FLM01AP1 does not directly call any language translators. It references other architecture members. The Build process creates the load modules FLM01LD7, FLM01LD9, FLM01LD1, FLM01LD2, FLM01LD3, and FLM01LD4.

  Note:

  1. Load module FLM01LD2 is not created if the project is defined using the SCLM sample project utility and the first optional compiler field was not selected.
  2. Load module FLM01LD7 is not created if the project is defined using the SCLM sample project utility and the second optional compiler field was not selected.
- FLM01AP1,FLM01SB1 and FLM01SB2 are high-level architecture members. They do not call any language translators. FLM01LD7, FLM01LD9, FLM01LD1,FLM01LD2,FLM01LD3, and FLMO1LD4 are LEC architecture members. FLM01CMD and FLM01CM9 are CC architecture members, and FLM01ARH is an architecture member that is directly copied into FLM01LD3 and FLM01LD4.

  Note:

  1. Architecture member FLM01LD2 is not included if the project is defined using the SCLM sample project utility and the first optional computer field was not selected.
  2. Architecture member FLM01LD7 is not included if the project is defined using the SCLM sample project utility and the second optional compiler field was not selected.

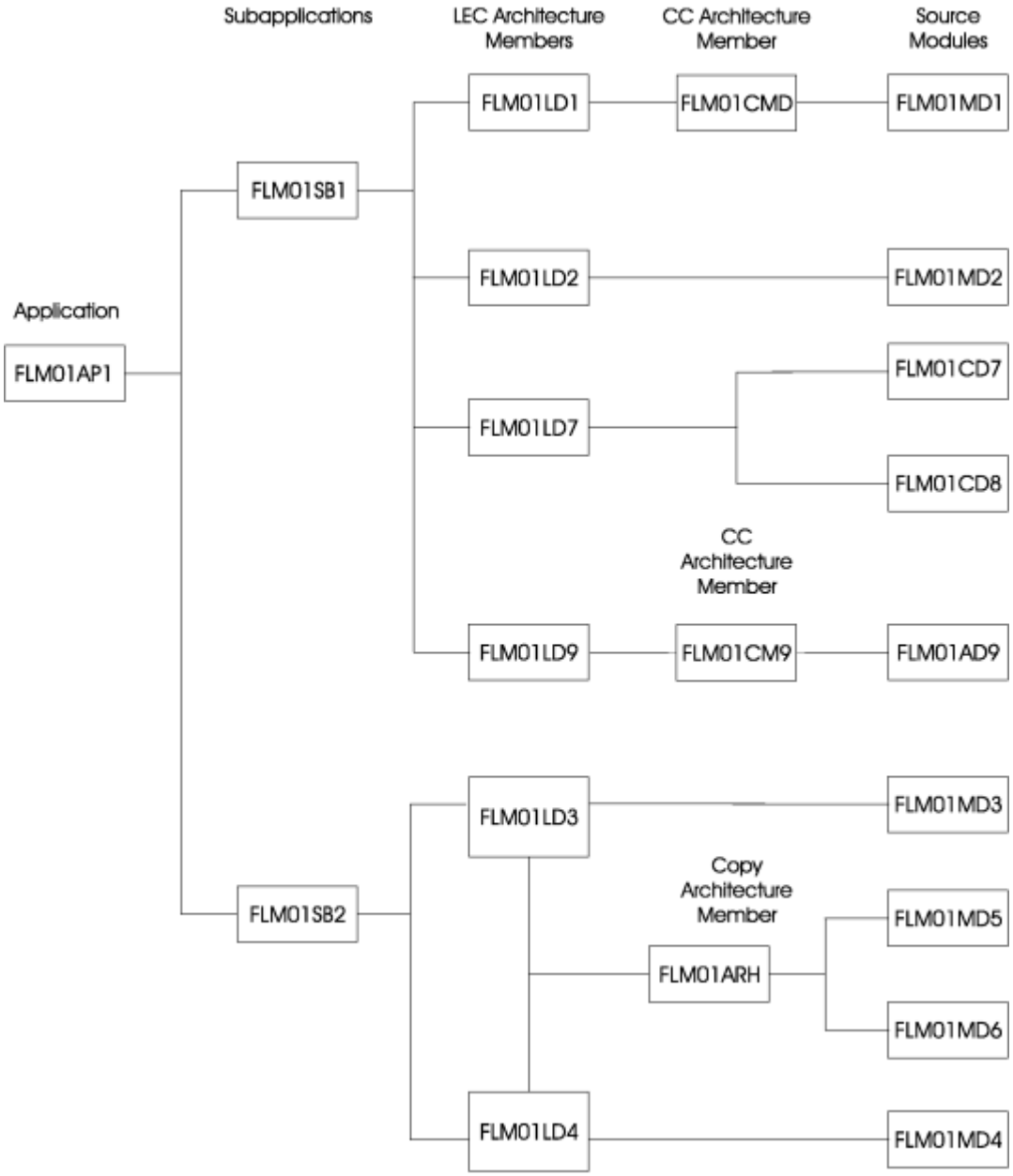


Figure 21. Example Project Architecture

Note:

1. Source module FLM01MD2 and architecture member and load module FLM01LD2 are not included if the project was defined using the SCLM sample project utility (Option 10.7) and the first optional compiler field was not selected.
2. Source module FLM01CD7, FLM01CD8, and architecture member and load module FLM01LD7, are not included if the project was defined using the SCLM sample project utility (Option 10.7) and the second optional compiler field was not selected.

Preparing the example project hierarchy

Use the following steps to install the example project data sets on your system. Follow the steps in the order listed and exactly as they are described. When you have completed all of the steps, you will have an SCLM project database with which you can experiment to better understand how SCLM works. If you encounter any errors during the following steps, use the TSO, ISPF, and SCLM messages to correct the problem. You can also define the sample project using the SCLM Sample Project utility (Option 10.7).

Note: This is the project that uses sample FLM01PRJ.

In the descriptions that follow, the default naming convention (PROJECT.GROUP.TYPE) is used. Assume for these examples that the project name is PROJ1. If you use a different name, be sure to inform those users who plan to complete the programmer scenario.

1. Sign on to TSO.
2. At the READY prompt, start ISPF.
3. Using the ISPF Data Set Utility, allocate the following partitioned data set with space in blocks (10,50), with 10 directory blocks, and with record format FB, LRECL 80:

   ```
   PROJ1.PROJDEFS.SOURCE
   ```

   This partitioned data set will contain the source code for the library structure as defined in the project definition.
4. Using the ISPF Data Set Utility, allocate the following partitioned data set with space in blocks (10,50), with 10 directory blocks, and with record format FB, LRECL 80:

   ```
   PROJ1.PROJDEFS.OBJ
   ```

   This partitioned data set will contain the object code for the library structure as defined in the project definition.
5. Using the ISPF Data Set Utility, allocate the following partitioned data set with space in blocks (10,50), with 10 directory blocks, and with record format U, LRECL 0, BLKSIZE 6144:

   ```
   PROJ1.PROJDEFS.LOAD
   ```

   This partitioned data set will contain the load module for the library structure as defined in the project definition. This member is named PROJ1.

   Note: Depending on the ISPF configuration for your site, you might receive warning or error messages when attempting to edit an SCLM project using the ISPF editor.
6. Use the ISPF Move/Copy Utility to copy the following members from ISP.SISPSAMP into PROJ1.PROJDEFS.SOURCE: FLM01ASF, FLM01ASM, FLM01CBE, FLM01PLE, FLM01PRJ, FLM01SCR, FLM01370, FLM02ALL, and FLM02ACT.

   Note:

   a. If you are not using Fault Analyzer, do not copy FLM01ASF.

   b. If you are not using Fault Analyzer, delete the following lines from FLM01PRJ:

      ```
      IDILANGX FLMTYPE
      COPY  FLM01ASF
      ```
7. Member FLM02ALL of PROJ1.PROJDEFS.SOURCE is a background job that allocates all of the data sets needed for this example application. You must provide a job card and change any other information that is specific to your location; for example, change all the occurrences of USERID to PROJ1 and alter the job card. After you have made these changes, submit the job.

   If this job allocates all the required data sets, you can skip to Step 9. Use the ISPF Data Set List Utility to determine whether the data sets were allocated.

   If the required data sets have not been allocated, you can allocate them by following Step 8.

   Note: If you are not using Fault Analyzer, remove the following step from FLM02ALL:

   ```
   //*
   //STEP3ID EXEC PGM=1EFBR14
   //DEV1     DD DSN=project.DEV1.IDILANGX,DISP=(NEW,CATLG),UNIT=SYSDA,
   //    DCB=(RECFM=VB,LRECL=1562,BLKSIZE=0),SPACE=(27998,(5,20,10))
   //DEV2     DD DSN=project.DEV2.IDILANGX,DISP=(NEW,CATLG),UNIT=SYSDA,
   //    DCB=(RECFM=VB,LRECL=1562,BLKSIZE=0),SPACE=(27998,(5,20,10))
   //TEST     DD DSN=project.TEST.IDILANGX,DISP=(NEW,CATLG),UNIT=SYSDA,
   //    DCB=(RECFM=VB,LRECL=1562,BLKSIZE=0),SPACE=(27998,(5,20,10))
   ```

```
//RELEASE  DD DSN=project.RELEASE.IDILANGX,DISP=(NEW,CATLG),UNIT=SYSDA,
//    DCB=(RECFM=VB,LRECL=1562,BLKSIZE=0),SPACE=(27998,(5,20,10))
```

8. If Step 7 fails, or if you choose not to use the FLM02ALL JCL member, follow these steps to allocate the required data sets.

   a. Using the ISPF Data Set Utility, allocate the following partitioned data sets with space in blocks (10,50), with 10 directory blocks, and with record format FB, LRECL 80:

   ```
   PROJ1.DEV1.SOURCE
   PROJ1.DEV2.SOURCE
   PROJ1.TEST.SOURCE
   PROJ1.RELEASE.SOURCE
   ```

   These partitioned data sets will contain the source code for the project.

   b. Using the ISPF Data Set Utility, allocate the following partitioned data sets with space in blocks (10,50), with 10 directory blocks, and with record format FB, LRECL 80:

   ```
   PROJ1.DEV1.ARCHDEF
   PROJ1.DEV2.ARCHDEF
   PROJ1.TEST.ARCHDEF
   PROJ1.RELEASE.ARCHDEF
   ```

   These partitioned data sets will contain the architecture definition for the project.

   c. Using the ISPF Data Set Utility, allocate the following partitioned data sets with space in blocks (30,100), with 10 directory blocks, and with record format VB, LRECL 137:

   ```
   PROJ1.DEV1.LIST
   PROJ1.DEV2.LIST
   PROJ1.TEST.LIST
   PROJ1.RELEASE.LIST
   ```

   These partitioned data sets will contain the listings from the compilations and assemblies of the modules.

   d. Using the ISPF Data Set Utility, allocate the following partitioned data sets with space in blocks (15,50), with 10 directory blocks, and with record format FB, LRECL 80:

   ```
   PROJ1.DEV1.OBJ
   PROJ1.DEV2.OBJ
   PROJ1.TEST.OBJ
   PROJ1.RELEASE.OBJ
   ```

   These partitioned data sets will contain the object code from the compilations and assemblies of the modules.

   e. Using the ISPF Data Set Utility, allocate the following partitioned data sets with space in blocks (7,13), with 10 directory blocks, and with record format U,LRECL 0, BLKSIZE 6144:

   ```
   PROJ1.DEV1.LOAD
   PROJ1.DEV2.LOAD
   PROJ1.TEST.LOAD
   PROJ1.RELEASE.LOAD
   ```

   These partitioned data sets will contain the load modules from the link-edits of the modules.

   f. Using the ISPF Data Set Utility, allocate the following partitioned data sets with space in blocks (5,20), with 10 directory blocks, and with record format FBA, LRECL 121:

   ```
   PROJ1.DEV1.LMAP
   PROJ1.DEV2.LMAP
   PROJ1.TEST.LMAP
   PROJ1.RELEASE.LMAP
   ```

   These partitioned data sets will contain the load maps from the link-edits of the modules.

g. Using the ISPF Data Set Utility, allocate the following partitioned data set with space in blocks (300,100) with 10 directory blocks, and with record format VB,LREC 1562:

```
PROJ1.DEV1.IDILANGX
PROJ1.DEV2.IDILANGX
PROJ1.TEST.IDILANGX
PROJ1.RELEASE.IDILANGX
```

These partitioned data sets will contain side files generated by Fault Analyzer.

Note: If you are not using Fault Analyzer, do not allocate PROJ1.*.IDILANGX data sets.

9. Using the ISPF Library Utility, rename member FLM01PRJ in PROJ1.PROJDEFS.SOURCE to PROJ1. This member contains the source code for the project definition for PROJ1.
10. Using ISPF Edit, edit PROJ1.PROJDEFS.SOURCE(PROJ1). Change all occurrences of USERID to PROJ1.
11. Using ISPF Edit, edit PROJ1.PROJDEFS.SOURCE(FLM01ASM). Change all system macro library references to the library of macros at your location.

    You must change the members FLM01ASF, FLM01CBE, FLM01PLE, FLM01SCR, and FLM01370 so that libraries, assemblers, and assembler options match the libraries and products in use at your location. The changes are specified in the samples delivered.

    Note: If you make changes to these members after Step 14 while installing this example project, reassemble and relink the data set PROJ1.PROJDEFS.SOURCE(PROJ1). If you are not sure this step is required, reassemble and relink.
12. Using ISPF Edit, edit PROJ1.PROJDEFS.SOURCE(FLM02ACT). Be sure that the job card contains valid accounting information. Change all occurrences of USERID to PROJ1.

    This member contains JCL that constructs the VSAM cluster used to contain the accounting information used by SCLM. You also need to alter the volumes for IDCAMS for your location, and you might need to make additional changes to conform to requirements at your location.
13. Submit the JCL in PROJ1.PROJDEFS.SOURCE(FLM02ACT). You know that your job has completed successfully when the PROJ1.ACCOUNT.FILE VSAM cluster is created.

    This is the VSAM data set that contains the SCLM accounting information for the project. This job deletes the cluster and then creates the cluster. Because the cluster does not exist the first time you submit the job, you receive a return code of 8 in the listing data set.
14. Assemble PROJ1.PROJDEFS.SOURCE(PROJ1) using either ISPF Foreground Assembler (option 4.1) or the sample JCL in “Assemble and link example” on page 39.

    Be sure that the SCLM macro library used at your location is in the concatenation sequence for the libraries used by the assembler. Specify the macro library in the Additional Input Libraries field on the Foreground Assembly panel.

    Look at the listing and confirm that no statements were flagged.
15. If you used the sample JCL in the “Assemble and link example” on page 39, make sure that member PROJ1.PROJDEFS.LOAD(PROJ1) exists.

    Otherwise, use the ISPF Foreground Linkage Editor to link-edit PROJ1.PROJDEFS.OBJ(PROJ1). This constructs the load module PROJ1.PROJDEFS.LOAD(PROJ1) that is executed by SCLM to control the library.

    Verify that the link occurred without errors.
16. Use the ISPF Move/Copy Utility to copy the following members from ISP.SISPSAMP into PROJ1.DEV1.SOURCE (these are the source members for the application and are moved into PROJ1.RELEASE.SOURCE later): FLM01AD9, FLM01CDT, FLM01CD7, FLM01CD8, FLM01EQU, FLM01CIN, FLM01IIN, FLM01INC, FLM01MD1, FLM01MD2, FLM01MD3, FLM01MD4, FLM01MD5, and FLM01MD6.
17. Use the ISPF Move/Copy Utility to copy the following members from ISP.SISPSAMP into PROJ1.DEV1.ARCHDEF (these are the architecture definition members and are moved into

PROJ1.RELEASE.ARCHDEF later): FLM01AP1, FLM01ARH, FLM01CMD, FLM01CM9, FLM01LD1, FLM01LD2, FLM01LD3, FLM01LD4, FLM01LD7, FLM01LD9, FLM01SB1, and FLM01SB2.

Note: If you are not using Fault Analyzer, edit FLM01CM9 as follows:

```
         Delete line OUT2  FLM1AD9 IDILANGX
         Change PARM1 to PARM
```

Understanding the sample project definition

This section examines the project definition used for the library in the sample project. Typically, the project manager is responsible for developing and maintaining the project definition.

1. Select the View option from the SCLM Main Menu and type:

   PROJ1
   in the **Project** field

   DEV1
   in the **Group** field

   Press Enter.

   Type `'PROJ1.PROJDEFS.SOURCE(PROJ1)'` in the **Data Set Name** field, and press Enter to examine the member that contains the project definition for PROJ1. The macros are:

   FLMABEG
   FLMABEG initializes the project definition by defining the project name as PROJ1.

   FLMTYPE
   FLMTYPE defines each type. The type values are:

   ARCHDEF
   architecture definitions

   SOURCE
   source code

   LIST
   listings from compilers and assemblers

   IDILANGX
   Fault Analyzer data sets

   Note: If you are not using Fault Analyzer, make sure that you have edited FLM01PRJ as specified in step 6 of "Preparing the example project hierarchy".

   OBJ
   object code

   LMAP
   load module maps

   LOAD
   executable load modules

   The type names were chosen arbitrarily for this sample project.

   FLMGROUP
   FLMGROUP defines each group. The PROMOTE keyword defines the library structure. Note that DEV1 and DEV2 are promoted to TEST and TEST is promoted to RELEASE.

   FLMCNTRL
   FLMCNTRL identifies the default VSAM data sets for the project. The VSAM data sets store library control information about the members in the project hierarchy.

   COPY
   COPY identifies members to be copied into the project definition. The members identified are the architecture definition language, assembler language, PL/I language, link-edit language, and SCRIPT language definitions.

FLMAEND
FLMAEND ends the project definition.

An additional developer, DEV3, can be added with another FLMGROUP macro, as shown in the following example:

```
DEV3     FLMGROUP AC=(P),KEY=Y,PROMOTE=TEST
```

The project definition specifies the names of the partitioned data sets used by the project (for example, PROJ1.DEV1.SOURCE), the library structure for the groups (for example, DEV1 members are promoted to TEST), and the languages to be used (for example, architecture definition, ASM, COBOL, PL/I, and link-edit).

2. View the PROJ1.PROJDEFS.SOURCE members:

FLM01ASF
ASMF language definition

FLM01ASM
ASM language definition

FLM01CBE
COBE language definition

FLM01PLE
PLIE language definition

FLM01370
linkage editor language definition

Note: FLM01ASF will be displayed only if you are using Fault Analyzer.

Note the following points about these members:

FLMSYSLB
This macro can be used to define a set of libraries that contain project and/or system macros or includes.

FLMLANGL
This macro specifies the language identifier.

FLMTRNSL
This macro is used once for each translator to be invoked for a language.

The SCLM parser is invoked when the keyword FUNCTN specifies PARSE. The SCLM parser stores statistics (for example, lines-of-code counts) and dependency information (for example, includes and copy statements).

The build translator is invoked when the keyword FUNCTN specifies BUILD. In FLM01370, the linkage editor IEWL is invoked. The build fails unless the return code is equal to, or less than, the value specified by the keyword GOODRC (0 in this example).

FLMALLOC
This macro is used to allocate data sets and ddnames required by translators.

Preparing the example project data

The following steps prepare the example project data. Perform the steps in the order listed and exactly as they are described. When you have completed all of the steps, all necessary data will reside at the RELEASE group. At this point, you or other SCLM users can use the data to experiment with and understand SCLM.

1. Select the SCLM option from the ISPF Primary Option panel.
2. Select the Utilities option from the SCLM Main Menu. Type:

PROJ1
in the **Project** field

DEV1
in the **Group** field

Leave the **Alternate** field blank.

3. Skip this step if you are not using Fault Analyzer. Otherwise:
   a. From the Utilities panel, select the Migration option. Type:

   SOURCE
   in the **Type** field

   FLM01AD9 (the ASM module)
   in the **Member** field

   1
   in the **Mode** field

   HLAF
   in the **Language** field

   1
   in the **Process** field

   1
   in the **Messages** field

   4
   in the **Report** field

   4
   in the **Listings** field

   Press Enter to begin processing. The migration utility registers new modules (in this case, FLM01AD9) into an SCLM library by creating accounting records for them.

   b. Skip this step if you are not using Fault Analyzer. Otherwise, from the Utilities panel, select the Migration option. Type:

   SOURCE
   in the **Type** field

   FLM01INC (the ASM copybook)
   in the **Member** field

   1
   in the **Mode** field

   HLAF
   in the **Language** field

   1
   in the **Process** field

   1
   in the **Messages** field

   4
   in the **Report** field

   4
   in the **Listings** field

   Press Enter to begin processing. The migration utility registers new modules (in this case, FLM01INC) into an SCLM library by creating accounting records for them.

   c. From the Utilities panel, select the Migration option. Type:

   SOURCE
   in the **Type** field

   FLM01C* (the COBOL copybook)
   in the **Member** field

1
in the **Mode** field

COBE
in the **Language** field

1
in the **Process** field

1
in the **Messages** field

4
in the **Report** field

4
in the **Listings** field

Press Enter to begin processing. The migration utility registers new modules (in this case, FLM01CIN,FLM01CDT,FLM01CD7,FLM01CD8) into an SCLM library by creating accounting records for them.

d. From the Utilities panel, select the Migration option. Type:

SOURCE
in the **Type** field

FLM01IIN (the PL/I copybook)
in the **Member** field

1
in the **Mode** field

PLIE
in the **Language** field

1
in the **Process** field

1
in the **Messages** field

4
in the **Report** field

4
in the **Listings** field

Press Enter to begin processing. The migration utility registers new modules (in this case, FLM01IIN)into an SCLM library by creating accounting records for them.

e. From the Utilities panel, select the Migration option. Type:

SOURCE
in the **Type** field

FLM01MD2 (the PL/I copybook)
in the **Member** field

1
in the **Mode** field

PLIE
in the **Language** field

1
in the **Process** field

1
in the **Messages** field

4
in the **Report** field

4
in the **Listings** field

Press Enter to begin processing. The migration utility registers new modules (in this case, FLM01MD2)into an SCLM library by creating accounting records for them.

4. When the migration is complete, you receive the message MIGRATION UTILITY COMPLETED with RETURN CODE = 0. The Migration Utility panel reappears. Type:

   *****
   in the **Member** field

   ASM
   in the **Language** field

   Press Enter to begin processing.

   Notice that you did not have to type **EX** on the command line or re-enter a value in the Process field. The value is carried from panel to panel and is maintained as is until you change it.

   The Migration Utility registers the SCLM accounting information for the remaining new modules (in this example, all are assembler language modules). Each time you use the Migration Utility, you can only migrate modules written in the same language. This example migrates FLM01MD2 first. After its migration, the other modules can be referenced as a group by using the asterisk (*). Because FLM01CDT, FLM01CIN, FLM01IIN, FLM01CD7, FLM01CD8, FLM01MD2, FLM01INC, and FLM01AD9 were migrated first, SCLM does not migrate them again when an * is specified.

   Note: FLM01INC and FLM01AD9 would be migrated already only if you are using Fault Analyzer.

5. When the migration is complete, you receive the message MIGRATION UTILITY COMPLETED with RETURN CODE = 0. The Migration Utility panel reappears. Type:

   ARCHDEF
   in the **Type** field

   *****
   in the **Member** field

   ARCHDEF
   in the **Language** field

   Press Enter to begin processing.

6. Return to the SCLM Main Menu. Select the Build option and press Enter.
7. On the Build panel, type:

   DEV1
   in the **Group** field

   ARCHDEF
   in the **Type** field

   FLM01AP1
   in the **Member** field

   / (slash)
   in the **Error Listings only** field

   1
   in the **Mode** field

   2
   in the **Scope** field

   1
   in the **Messages** field

   1
   in the **Report** field

3
in the **Listings** field

Press Enter. All modules in the project are assembled or compiled. SCLM updates the accounting information to indicate that a build operation was performed on each module. The Build Messages and Build Report reappears. The build should complete with a RETURN CODE = 0. The Build panel reappears.

If all of the site-dependent changes to the system macro library references were not made in "10" on page 45, build errors can occur during this step. If this happens, correct the macros, reassemble and link-edit the project definition, and repeat this step.

8. Return to the SCLM Main Menu. Select the Promote option and press Enter.
9. On the Promote panel, type:

DEV1
in the **From Group** field

ARCHDEF
in the **Type** field

FLM01AP1
in the **Member** field

1
in the **Mode** field

1
in the **Scope** field

1
in the **Messages** field

1
in the **Report** field

Press Enter. SCLM copies all members for all types at the DEV1 group to the TEST group and then purges all members from the DEV1 group. The Promote Messages and Promote Report appears. The Promote should complete with a RETURN CODE = 0. The Promote panel reappears.

10. On the Promote panel, type:

TEST
in the **From Group** field

ARCHDEF
in the **Type** field

FLM01AP1
in the **Member** field

1
in the **Mode** field

1
in the **Scope** field

1
in the **Messages** field

1
in the **Report** field

EX
on the command line

Press Enter. SCLM copies all members for all types at the TEST group to the RELEASE group and then purges all members from the TEST group. The Promote Messages and Promote Report appears. The Promote should complete with a RETURN CODE = 0. The Promote panel reappears.

All of the modules are located in the RELEASE group, and the SCLM example project, PROJ1, is now ready to use. This scenario illustrates the status of a current release of a product that does not have any maintenance, test, or development activities underway.

Chapter 2. User exits

SCLM provides a number of exit points that you can use to customize SCLM processing or to integrate SCLM with other products. SCLM does not provide the user exit routines to be invoked at these exit points. You can specify your own user exit routines in the project definition using the user exit parameters on the FLMCNTRL macro.

There can be performance implications associated with the specification of an exit routine depending on the processing performed by the exit routine. You can write a user exit routine in any language, including REXX. The exit routine can use any of the SCLM services to retrieve additional information that is not returned by the exit.

Writing and compiling a program to be reentrant, then specifying RENT and REUS on the link-edit makes the invocation of the routine more efficient.

Table 7 on page 53 lists the exits supplied by SCLM, along with the FLMCNTRL parameter used to specify an associated user exit routine. The "Initial" and "Verify" exits are invoked before any real processing (change to data) occurs, and can be used to perform tasks such as verifying a user's authority to perform a given function.

The Promote Copy, Promote Purge, and *all* "Notify" exits are invoked after processing has completed, and can be used to perform tasks such as putting an entry into a log file, generating a report, or sending notification to a specified set of users.

All of these exit points can be used to integrate SCLM with other products as well as to enable customized processing. For example, a Verify Change Code Exit routine might be used to query an external change management product to ensure that an open problem request exists for a change being made, and that the user making the change is authorized to do so. The SCLM sample bridge to Tivoli Information Management is an example of this type of exit routine.

Here are the available exits, along with the FLMCNTRL parameters used to specify an associated user exit routine.

Table 7. Exits and Exit Routine Specifications

Exit	Exit Routine Specification	When Invoked
Verify Change Code Exit	CCVFY	• At the start of an SCLM Edit session: – In SCLM Edit (option 2) before the member list is displayed (note that in this case, no member name is passed to the exit) – In SCLM Edit (option 2), on entry to edit of a member if the member name is specified explicitly – In the Library utility (3.1), on entry to edit of a member • When Change Code or Language is changed in SPROF • By the EDIT service.
Save Change Code Exit	CCSAVE	• After a member has been saved, but before SCLM accounting information is updated for the member • By the Migrate (3.3) utility • By the EDIT, MIGRATE, SAVE, and STORE services

© Copyright IBM Corp. 1990, 2019

Table 7. Exits and Exit Routine Specifications (continued)

Exit	Exit Routine Specification	When Invoked
Change Code Verification Exit (superseded)	VERCC	• At the start of an SCLM Edit session: – in SCLM Edit (option 2) before the member list is displayed (note that in this case, no member name is passed to the exit) – in SCLM Edit (option 2), on entry to edit of a member if the member name is specified explicitly – in the Library utility (3.1), on entry to edit of a member • When Change Code is changed in SPROF • By the Migrate (3.3) and Import (3.7) utilities • By the EDIT, IMPORT, MIGRATE, SAVE, and STORE services **Note:** 1. If VERCC is present in PROJDEFS, the Change Code cannot be blank when a member is saved. 2. VERCC has been superseded by CCVFY.
Build Initial Exit	BLDINIT	At the beginning of Build before any verification or processing occurs
Build Notify Exit	BLDNTF or BLDEXT1	After Build processing completes
Promote Initial Exit	PRMINIT	At the beginning of Promote before any verification or processing occurs
Promote Verify Exit	PRMVFY or PRMEXT1	At the end of the Verification phase of Promote, but before the Copy and Purge steps are processed
Promote Copy Exit	PRMCOPY or PRMEXT2	At the end of the Copy phase of Promote, but before the Purge step is processed
Promote Purge Exit	PRMPURGE or PRMEXT3	At the end of Promote after the Verification, Copy, and Purge phases have all been completed
Audit/Version Delete Verify Exit	AVDVFY	After the input parameters have been verified for an audit record and version, but before the record is deleted
Audit/Version Delete Notify Exit	AVDNTF	After the audit record has been deleted
Delete Initial Exit	DELINIT	• By the Delete from Group utility, before delete processing begins • By the DELGROUP service, before delete processing begins
Delete Verify Exit	DELVFY	• By the Library utility, after the input parameters have been verified but before the member is deleted • By the DELETE service, after the input parameters have been verified but before the member is deleted
Delete Notify Exit	DELNTF	• After delete processing has completed for the Delete from Group utility or DELGROUP service • After delete processing has completed for the Library Utility Delete option, or the DELETE service

Specify the change code verification routine

SCLM provides three exits you can use for verifying change codes, integrating with change management systems, or practically any other Edit, Migrate, Save, or Store processing you might want to perform:

- The **Verify Change Code** exit (CCVFY) enables you to verify a change code, a language, a user id, or other values. The exit routine is invoked at Edit verification and SPROF processing. It is invoked during SPROF processing when either the language or the change code has changed. A blank change code is acceptable. A nonzero return code from the exit routine stops processing immediately.
- The **Save Change Code** exit (CCSAVE) occurs before SCLM writes accounting data to the accounting data set for Edit, Migrate, Save, or Store processing. The routine is invoked during Save. This includes Edit save processing, the Migrate Utility, and the EDIT, STORE, SAVE, and MIGRATE services. A blank change code is acceptable. A nonzero return code from the exit routine stops processing immediately.
- The **Change Code Verification** exit (VERCC) was superseded by CCVFY. Like CCVFY it can be used to verify change records. A nonblank change code is required. If you supply this routine to SCLM, it is used by the SCLM Editor, Migration, and Import utilities, as well as the EDIT, IMPORT, MIGRATE, SAVE, and STORE services.

  When the VERCC routine is invoked just before the edit, SCLM stores the return code and allows the edit to begin. If the VERCC routine has set a nonzero return code, the VERCC routine will be invoked again when the member is saved. When a VERCC routine fails during a save, you have two options:

  – You can use the CREATE edit command to make a non-SCLM-controlled copy of the editing session and then use the migrate utility to bring the member back under SCLM control.
  – You can use SPROF from SCLM Edit to change or add the change code.

You can specify any or all of these routines for your project. If you specify a VERCC exit and a CCVFY or CCSAVE exit routine, the VERCC exit routine is invoked first. The CCVFY or CCSAVE exit routine is only invoked if the VERCC exit completes successfully. The exception is during SPROF processing where the CCVFY exit routine is called without first invoking the VERCC exit routine when only the language has changed.

All three of these exit routines are invoked in the same way.

SCLM passes a string of up to eight parameters separated by commas. The parameter list can include one list of user-specified options followed by up to seven SCLM parameters (see Table 8 on page 55). Register 1 contains the address of the input data. The first halfword of the input data is the length of the input string. Immediately following the halfword length is the input parameter string. The return code from the routine is the only parameter passed back. The return code is returned in Register 15. SCLM allows a member to be saved only if it receives a return code of 0 from the exit routine. SCLM informs you if it detects a nonzero return code.

A project can use any combination of the parameters to determine whether an update should be permitted. Table 8 on page 55 explains the format and description of the parameters passed from SCLM to all change code verification routines.

Table 8. Initial and Save Change Code Exit Routine Parameters

Parameter	Description
OPTION LIST	Up to 255-character (including delimiters) parameters specified on the FLMCNTRL macro using the CCVFYOP for options to the verify change code exit routine and CCSAVOP for those passed to the save change code exit routine. Delimit this string so that the SCLM parameters that follow can be identified by the exit routine.
GROUP	The 8-character name of the group in which the member is being created or modified (capitalized, left-aligned, blank-padded).

Table 8. Initial and Save Change Code Exit Routine Parameters (continued)

Parameter	Description
TYPE	The 8-character name of the member type being created or modified (capitalized, left-aligned, blank-padded).
MEMBER	The 8-character name of the member that is being created or modified (capitalized, left-aligned, blank-padded).
LANGUAGE	The 8-character name of the language specified for the member (capitalized, left-aligned, blank-padded).
USERID	The 8-character user ID of the developer performing the modification (capitalized, left-aligned, blank-padded).
AUTHCODE	The 8-character authorization code for the member (capitalized, left-aligned, blank-padded).
CHANGE CODE	The 8-character change code that has been entered (capitalized, left-aligned, blank-padded).

Change code verification routine example

The following example shows a simple program written in REXX to perform minimal verification. This routine verifies that the change code entered on the edit panel, or on the SPROF screen exists in a change code verification file. A return code of 0 indicates that the change code is valid. A return code of 8 indicates that the change code failed verification. The example assumes that the option list is empty.

Figure 22 on page 57 calls the REXX Parse function to separate the string of input parameters. The example then allocates the verification file and loops through the lines in the file until a matching change code is found. If one is found the program is left immediately, otherwise a return code of 8 tells SCLM to fail verification.

```
/* REXX ***************************************************************/
/* CCVERIFY - CHANGE CODE VERIFICATION USER EXIT                      */
/**********************************************************************/
/* INPUTS:                                                            */
/* PARMS -                                                            */
/* OPTION LIST - OPTIONS LIST (IF SPECIFIED ON FLMCNTRL).             */
/*  GROUP       - GROUP WHERE THE CHANGE IS BEING MADE.               */
/*  TYPE        - TYPE CONTAINING THE MEMBER BEING CHANGED.           */
/*  MEMBER      - MEMBER BEING CHANGED.                               */
/*  LANGUAGE    - LANGUAGE OF MEMBER BEING CHANGED.                   */
/*  USERID      - USER ID PERFORMING THE CHANGE.                      */
/*  AUTHCODE    - AUTHORIZATION CODE OF THE MEMBER.                   */
/*  CHANGE CODE - CHANGE CODE BEING USED FOR THE CHANGE.              */
/**********************************************************************/
/* OUTPUTS:                                                           */
/* RETURN_CODE - RETURN CODE                                          */
/* 0  - CHANGE CODE IS VALID.                                         */
/* 8  - CHANGE CODE IS INVALID.                                       */
/* 16 - CHANGE CODE FILE OPEN ERROR                                   */
/**********************************************************************/
/* PROCESS:                                                           */
/* THIS PROGRAM VERIFIES THAT THE CHANGE CODE ENTERED FOR THE         */
/* MEMBER MATCHES ONE ON A VALID CHANGE CODE FILE                     */
/**********************************************************************/

   ARG parm                  /* Parse arguments into variable parm */
   PARSE UPPER VAR parm group ',' type ',' member ',' lang ',',
                       userid ',' authcode ',' ccode

   group    = Strip(group,'T')
   type     = Strip(type,'T')
   member   = Strip(member,'T')
   lang     = Strip(lang,'T')
   userid   = Strip(userid,'T')
   authcode = Strip(authcode,'T')
   ccode    = Strip(ccode,'T')

   Address TSO "ALLOC FI(CCODEDS) DA('SSP.SCLM.CCIDVAL') SHR"

   "EXECIO * DISKR "CCODEDS" (STEM ccline. FINIS)"

   If rc <> 0 Then do
      Say 'Error reading change code file'
      Exit (16)
   End

   Address TSO "FREE FI(CCODEDS)"

   Do I = 1 To ccline.0
      If SUBSTR(ccline.I,1,8) = ccode then Exit (0)
   End
   Say "Invalid change code"

Exit (8)
```

Figure 22. Change Code Verification User Exit

This exit will be executed by specifying the following FLMCNTRL macro in the project definition:

```
FLMCNTRL ACCT=SSP.ACCOUNT.FILE,                                    C
      MAXVIO=50000,                                                C
      CCVFY=CCVERIFY,                                              C
      CCVFYDS=SSP.PROJDEFS.REXX,                                   C
      CCVFYCM=TSOLNK
```

Specify the Build and Promote User Exit routines

Two user exits can be specified for build. SCLM invokes the Build Initial user exit before any build processing begins. The Build Notify user exit is invoked at the end of a build.

Four user exits can be specified for promote. SCLM invokes the Promote Initial user exit before any promote processing begins. SCLM invokes the Promote Verification user exit, the Promote Copy user exit, and the Promote Purge user exit routines at the end of the promote verification, copy, and purge phases, respectively.

Build and promote user exits are defined to the project definition using the following parameters on the FLMCNTRL macro.

Build Initial User Exit
BLDINIT

Build Notify User Exit
BLDNTF or BLDEXT1 (old format)

Promote Initial User Exit
PRMINIT

Promote Verify User Exit
PRMVFY or PRMEXT1 (old format)

Promote Copy User Exit
PRMCOPY or PRMEXT2 (old format)

Promote Purge User Exit
PRMPRURGE or PRMEXT3 (old format)

Build and Promote User Exit routine requirements

If you specify a user exit option parameter, SCLM passes it to the user exit routine, followed by a string of up to eleven parameters separated by commas. The parameter list can include one list of user-specified options followed by up to ten SCLM parameters (see Table 9 on page 59). The address of this input data is contained at the address stored in register 1. The first halfword of the input data is the number of characters comprising the input data string. Immediately following this halfword length is the input parameter string itself.

The user exit routine must pass back a return code value to SCLM in register 15. A return code of zero is considered to be successful and processing continues. In most situations a nonzero return code from the user exit routine causes build or promote to end with a return code 8. Whether or not processing continues after the user exit depends on the return code value passed back by the user exit routine and the exit routine being invoked. Nonzero return code values from user exit routines are handled in the following ways:

- Both the Build Notify user exit (BLDNTF) and the promote purge phase user exit (PRMPURGE) can return any value as processing has already been completed at the time the exit is invoked. SCLM will, however, set a return code of 4 (in the case of BLDNTF) or 8 (in the case of PRMPURGE) for the final SCLM return code if a nonzero return code is set in the user exit.
- Any nonzero value returned by the Build Initial user exit (BLDINIT) or the Promote Initial user exit (PRMINIT) causes processing to stop.
- The processing that occurs after the promote verification phase user exit (PRMVFY) has been invoked depends on the promote mode in effect. In conditional mode, a return code greater than 4 causes promote processing to stop. In unconditional mode, any return code other than 20 allows promote processing to continue.
- The processing that occurs after the Promote Copy user exit (PRMCOPY) has been invoked depends only on the return code value returned. Any return code other than 20 allows normal promote processing to continue.

Table 9 on page 59 explains the format and description of the parameters passed from SCLM to all build and promote user exits.

Table 9. User Exit Parameters

Parameter	Description
OPTION LIST	Up to 255 characters, including delimiters (blank padding is not performed for this parameter). Parameter is specified in the FLMCNTRL macro using macro parameters BLDINIOP, BLDNTFOP, PRMINIOP, PRMVFYOP, PRMCPYOP, and PRMPRGOP. Delimit this string so that the SCLM parameters that follow can be identified by the user exit routine.
'xxxxxxxx'	An 8-character literal value indicating the exit type (capitalized, left-aligned, blank-padded). Valid types are: **BINITIAL** Build Initial (BLDINIT) **BUILD** Build Notify (BLDNTF) **PINITIAL** Promote Initial (PRMINIT) **PVERIFY** Promote Verify (PRMVFY) **PCOPY** Promote Copy (PRMCOPY) **PPURGE** Promote Purge (PRMPURGE).
PROJECT	The 8-character name of the project (capitalized, left-aligned, blank-padded).
LIBDEF	The 8-character name of the project definition (capitalized, left-aligned, blank-padded).
USERID	The 8-character value of the user's logon ID (capitalized, left-aligned, blank-padded).
FROM GROUP	The 8-character name of the group (capitalized, left-aligned, blank-padded). The group is the "from group" for the promote and the "build group" for the build.
TYPE	The 8-character name of the type (capitalized, left-aligned, blank-padded).
MEMBER	The 8-character name of the member (capitalized, left-aligned, blank-padded).
SCOPE	The 8-character name of the scope (capitalized, left-aligned, blank-padded). Valid scopes are as follows: **Build scope** Limited, normal, subunit, extended. **Promote scope** Normal, subunit, extended.
MODE	The 13-character name of the mode (capitalized, left-aligned, blank-padded). Valid modes are as follows: **Build mode** Forced, conditional, unconditional, and report only. **Promote mode** Conditional, unconditional, and report.
TO GROUP	The 8-character name of the group (capitalized, left-aligned, blank-padded). The group is the "to-group" for the promote exit routines. This parameter is blank for the build exit routine.

Build allocates the following ddnames for internal use: `BLDEXIT; BLDLIST; BLDMSGS; BLDREPT`

Promote allocates the following ddnames for internal use: `COPYERR; PROMEXIT; PROMMSGS; PROMREPT`

Use of these names in user exit routines can cause conflicts. At the end of an exit routine, free only those ddnames explicitly allocated by the exit routine.

Build and Promote User Exit output data sets

If you specify a Build Notify or Promote Verify, Promote Copy, or Promote Purge user exit routine, SCLM generates a sequential data set containing a record for each member changed or verified by build or promote. This data set is not generated for the Build Initial or Promote Initial user exits. Verified members are those eligible for promotion during the promote verification phase. Changed members for build are those members produced due to translator calls. Changed members for promote are those members copied or purged. SCLM puts new data in the data set for the invocation of each exit. User exit routines can use the output data set when called, but the data set is rewritten for later exits and is deleted when the SCLM processor ends.

The data definition names (ddnames) for build and promote exit output data sets are BLDEXIT and PROMEXIT respectively. The attributes of the output data sets are the same for all the exit routines:

RECFM
FB

BLOCK SIZE
3200

LRECL
160

The format of the data set is the same for every exit. The data set contains three 8-character fields and one 16-character status field. A blank separates all fields. The following list defines the fields generated for every build and promote exit routine:

Table 10. User Exit Output Data Set Format

Field	Description
GROUP	Specifies the 8-character name of the group beginning in column 1.
TYPE	Specifies the 8-character name of the type beginning in column 10.
MEMBER	Specifies the 8-character name of the member beginning in column 19.
STATUS	Specifies the status beginning in column 28. **BUILT/DELETED** Indicates if the member was built or if it was an obsolete output that was deleted. This field is written by BLDNTF. **PROMOTABLE/NOT PROMOTABLE** Indicates if the member is eligible for promotion. This field is written by PRMVFY. **COPY SUCCESSFUL/COPY FAILED/COPY NOT ATTEMPTED** Indicates if the member was copied. This field is written by PRMCOPY. COPY NOT ATTEMPTED can be issued when a promote to a non-key group is performed of a NOT PROMOTABLE member. **PURGE SUCCESSFUL/PURGE FAILED** Indicates if the member was purged. This field is written by PRMPURGE.

The following example shows build user exit output:

```
USER1   TYPE1   MEMBER1  BUILT
USER1   TYPE    MEM1     BUILT
USER1   TYPE2   MEMBER5  BUILT
```

Specify the Audit Version Delete User Exit routine

There are two audit version delete exit points in SCLM: audit version delete verify (AVDVFY) and audit version delete notify (ADVNTF). These exits are invoked when an audit record or an audit record and its associated version are deleted using either the SCLM Audit and Version Utility, Version Selection dialog (ISPF Option 10.3.8), or the VERDEL service interface.

The use of the audit version delete exits is optional. SCLM does not provide the user exit routines to be invoked by these exit points.

The audit version delete verify exit is invoked after the initial verification of the inputs is done, but before the actual deletion of the audit and version data takes place.

The audit version notify exit is invoked after the deletion of the audit and version data has been attempted (in the case of a failure) or performed (when the deletion is successful).

These exits can be used to perform logging functions or additional verification, send notifications or coordinate processing with non-SCLM tools.

Audit Version Delete User Exit routine requirements

If you specify a user exit option parameter, SCLM passes it to the user exit routine, followed by a string of up to eleven parameters separated by commas. The parameter list can include one list of user-specified options followed by up to ten SCLM parameters (see Table 11 on page 61). The address of this input data is contained at the address stored in register 1. The first halfword of the input data is the number of characters comprising the input data string. Immediately following this halfword length is the input parameter string itself.

The user exit routine must pass back a return code value to SCLM in register 15. A return code of zero is considered to be successful and processing continues. A nonzero return code from the first audit version delete exit verify routine (AVDVFY) causes processing to end and the requested audit and version information is not deleted. The second audit version delete notify user exit routine (AVDNTF) can pass back any value in register 15 and processing continues because the delete has already been performed.

Table 11 on page 61 explains the format and description of the parameters passed from SCLM to all audit version delete user exits.

Table 11. User Exit Parameters

Parameter	Description
OPTION LIST	Up to 255 characters, including delimiters (blank padding is not performed for this parameter). Parameter is specified in the FLMCNTRL macro using macro parameters AVDVFYOP and AVDNTFOP. Delimit this string so that the SCLM parameters that follow can be identified by the user exit routine.
‘xxxxxxxx’	An 8-character literal value indicating the exit type (capitalized, left-aligned, blank-padded). Valid types are: **ADVERIFY** Audit Version Delete Verify **ADNOTIFY** Audit Version Delete Notify
PROJECT	The 8-character name of the project (capitalized, left-aligned, blank-padded).

Table 11. User Exit Parameters (continued)

Parameter	Description
LIBDEF	The 8-character name of the project definition (capitalized, left-aligned, blank-padded).
USERID	The 8-character value of the user's logon ID (capitalized, left-aligned, blank-padded).
GROUP	The 8-character name of the group (capitalized, left-aligned, blank-padded) for the audit record or audit record and version.
TYPE	The 8-character name of the type (capitalized, left-aligned, blank-padded) for the audit record or audit record and version.
MEMBER	The 8-character name of the member (capitalized, left-aligned, blank-padded)for the audit record or audit record and version.
DATE	The 10-character, multicultural support, formatted date with 4-character year for the audit record or audit record and version.
TIME	The 11-character time for the audit record or audit record and version. The format for the time is HH:MM:SS.hh or HH:MM:SS,hh. In the format, HH is the hour from a 24-hour clock, MM is the minutes, SS is the seconds, and hh is the hundredths of a second.
VERSION MEMBER NAME	The 8-character version member name (capitalized, left-aligned, blank-padded) indicates whether the requested audit record has an associated version. When an associated version exists, this value is the same as the member name. This value is blank when the requested audit record does not have an associated version.

Specify the Delete User Exit routine

There are three delete exit points in SCLM: an initial delete exit (DELINIT), a Delete Verify exit (DELVFY), and a Delete Notify exit (DELNTF).

The initial delete exit is invoked only for the DELGROUP service or Delete from Group dialog (ISPF Option 10.3.9). It is invoked during initialization and before any processing is done. The "group" (for the DELGROUP service *only*), "type", and "member name" values can contain pattern symbols. The purpose of this exit is to enable verification for a certain level, for example, to ensure that a user is authorized to use Delete from Group.

The Delete Verify exit is invoked for Library Utility Delete (ISPF Option 10.3.1) and the DELETE service. It is invoked after the input parameters have been verified, but before any processing is performed.

The Delete Notify exit is invoked for Library Utility Delete, the DELETE service, and the DELGROUP service and Delete from Group dialog. The exit is invoked after the delete has been attempted (in the case of failure) or performed (when the deletion succeeds).

Delete User Exit Routine requirements

If you specify a user exit option parameter, SCLM passes it to the user exit routine, followed by a string of up to ten parameters separated by commas. The parameter list can include one list of user-specified options followed by up to nine SCLM parameters (see Table 12 on page 63). The address of this input data is contained at the address stored in register 1. The first halfword of the input data is the number of characters comprising the input data string. Immediately following this halfword length is the input parameter string itself.

The user exit routine must pass back a return code value to SCLM in register 15. A return code of zero is considered to be successful and processing continues. For the Delete Verify and Delete Initial exit routines, any return code other than zero indicates failure and processing ends. In the case of the Delete

Notify exit, the delete has already been performed. SCLM will, however, set a return code of 4 for the final SCLM return code if a nonzero return code is set in the user exit.

Table 12 on page 63 explains the format and description of the parameters passed from SCLM to all delete user exits.

Table 12. User Exit Parameters

Parameter	Description
OPTION LIST	Up to 255 characters, including delimiters (blank padding is not performed for this parameter). Parameter is specified in the FLMCNTRL macro using macro parameters DELINTOP, DELVFYOP, and DELNTFOP. Delimit this string so that the SCLM parameters that follow can be identified by the user exit routine.
'xxxxxxxx'	An 8-character literal value indicating the exit type (capitalized, left-aligned, blank-padded). Valid types are: **DGINIT** Initial Delete **DVERIFY** Verify delete exit invoked for the DELETE service or Library Utility Delete **DNOTIFY** Notify delete exit invoked for the DELETE service or Library Utility Delete **DGNOTIFY** Notify delete exit invoked for the DELGROUP service or Delete from Group dialog
PROJECT	The 8-character name of the project (capitalized, left-aligned, blank-padded).
LIBDEF	The 8-character name of the project definition (capitalized, left-aligned, blank-padded).
USERID	The 8-character value of the user's logon ID (capitalized, left-aligned, blank-padded).
GROUP	The 17-character name of the group (capitalized, left-aligned, blank-padded).
TYPE	The 17-character name of the type (capitalized, left-aligned, blank-padded).
MEMBER	The 17-character name of the member (capitalized, left-aligned, blank-padded).
FLAG	The 8-character delete flag (capitalized, left-aligned, blank-padded). Valid delete flags are ACCT, BMAP, TEXT, and OUTPUT. This value is always TEXT for a Library Utility Delete. OUTPUT is valid only for Delete from Group.
MODE	The 8-character name of the mode (capitalized, left-aligned, blank-padded). Valid modes are EXECUTE and REPORT. This value is valid only for Delete from Group. A blank value is passed for the DELETE service and Library Utility Delete.

Delete from Group allocates the following ddnames for internal use: `DGEXIT; DGLIST; DGMSGS; DGREPT`

Use of these names in a delete user exit routine can cause conflicts. At the end of an exit routine, free only those ddnames explicitly allocated by the exit routine.

Delete User Exit output data set

When a Delete from Group is performed and you specify a delete notify user exit routine, SCLM generates a sequential data set containing a record for each member for which a delete is requested. SCLM puts new data in the data set for the invocation of each exit. The Delete Notify user exit routine can use the

output data set when called, but the data set is rewritten for later exits and is deleted when the SCLM processor ends.

The default data definition name (ddname) for the delete exit output data set is DGEXIT. The attributes of the output data set are:

RECFM
FB

BLOCK SIZE
3200

LRECL
160

The data set contains the following fields. A blank separates all fields.

Table 13. User Exit Output Data Set Format

Field	Description
DATA TYPE	Specifies the 8-character name of the type of data. This is equivalent to the section headings in the Delete from Group report. Valid types are MEMBER or BUILDMAP. MEMBER is used when an accounting record or an accounting record and PDS member are deleted.
GROUP	Specifies the 8-character name of the group beginning in column 9.
TYPE	Specifies the 8-character name of the type beginning in column 18.
MEMBER	Specifies the 8-character name of the member beginning in column 27.
STATUS	Specifies the 19-character status beginning in column 36. Valid values are: **DELETE SUCCESSFUL** Indicates the requested data was successfully deleted. **DELETE FAILED** Indicates an error occurred and the delete failed. **DELETE WARNING** Indicates a warning was issued. The requested data either did not exist or was successfully deleted. **NOT ATTEMPTED** Indicates that Delete from Group was done in report mode. The delete was not attempted.
OUTPUT	Specifies the 1-character OUTPUT indicator beginning in column 56. If the requested data was a build output, then this column contains an asterisk (*).

The following example shows the delete user exit output that is generated when a Delete from Group is requested:

```
MEMBER  USER1   TYPE1   MEMBER1  DELETE SUCCESSFUL    *
```

User exit routine example

Figure 23 on page 65 is an example program written in REXX that performs simple Promote Copy user exit activity. This routine reads the promote exit file, and based on the types of the members being promoted, copies the member to a library outside of SCLM's control. The exit then passes a return code of zero (0) to SCLM.

```
/* REXX */
/* PROMCPY1 - PROMOTE COPY USER EXIT                                  */
/*********************************************************************/
/* INPUTS:                                                           */
/* PARMS  -                                                          */
/* EXTYP  -  An 8-character literal value indicating the exit type   */
/*           Valid types are:                                        */
/*           BINITIAL Build Initial (BLDINIT)                        */
/*           BUILD    Build Notify (BLDNTF)                          */
/*           PINITIAL Promote Initial (PRMINIT)                      */
/*           PVERIFY  Promote Verify (PRMVFY)                        */
/*           PCOPY    Promote Copy (PRMCOPY)                         */
/*           PPURGE   Promote Purge (PRMPURGE).                      */
/* PROJ    - The 8-character name of the project                     */
/* PRJDF   - The 8-character name of the project definition          */
/* TSOUID  - The 8-character value of the user's logon ID            */
/* FROMGRP - From Group or Build Group                               */
/* TYPE    - Type containing the member being promoted.              */
/* MEMBER  - Member being promoted.                                  */
/* SCOPE   - The 8-character name of the scope                       */
/*           Valid scopes are as follows:                            */
/*           Build scope Limited, normal, subunit, extended.         */
/*           Promote scope Normal, subunit, extended.                */
/* MODE    - The 13-character name of the mode                       */
/*           Valid modes are as follows:                             */
/*           Build mode Forced, conditional, unconditional,          */
/*           and report only.                                        */
/*           Promote mode Conditional, unconditional, and report.    */
/* TOGRP   - The 8-character name of the group;                      */
/*           blank for build exit                                    */
/*                                                                   */
/*********************************************************************/
/* OUTPUTS:                                                          */
/* RETURN_CODE - RETURN CODE                                         */
/* 0  - All copies performed successfully.                           */
/* 16 - All or some copies not performed successfully                */
/* 32 - Input or Output files can not be initialized                 */
/*********************************************************************/
/* PROCESS:                                                          */
/* THIS PROGRAM COPIES LOAD MODULES TO THEIR EXECUTION DATASET       */
/*                                                                   */
/*********************************************************************/

ARG PARM

/* Initialize passed parameters                                      */
Call INIT

/* Only process when to group is production                          */
If togrp <> 'PROD' then exit 0
```

Figure 23. Promote User Exit (Part 1 of 3)

```
/* read exit file                                                    */
"execio * diskr PROMEXIT (stem extline. finis)"

/* Process each line of the exit file                               */
Do i = 1 to extline.0  /* For all lines in stem variable             */

   /* Extract variables from a line out of the exit file            */
   parse upper var extline.i eogroup 10 eotype 19 eomember 28 eostatus

   eogroup = STRIP(eogroup)
   eotype = STRIP(eotype)
   eomember= STRIP(eomember)
   eostatus= STRIP(eostatus)
   /* If member ok continue */
   If eostatus = 'COPY SUCCESSFUL' then
      Call Process_Member
End

EXIT max_rc

INIT:
/* Parse out variables passed to the exit routine and strip blanks   */
PARSE UPPER VAR parm extyp ',' proj ',' prjdf ',' tsouid ',',
fromgrp ',' type ',' member ',' scope ',' mode ',' togrp

extyp   = strip(extyp)
proj    = strip(proj)
prjdf   = strip(prjdf)
tsouid  = strip(tsouid)
fromgrp = strip(fromgrp)
type    = strip(type)
member  = strip(member)
scope   = strip(scope)
mode    = strip(mode)
togrp   = strip(togrp)

max_rc   = 0

return

Process_Member:
/* Process each member in the exit file                              */
/* If the member type is to be processed setup 'TO' dataset          */
/* 'TO' dataset for the copy is a preallocated library              */

Select
   When eotype = "LOADLIB" then Do
      outdsn = "'SYS2.LOADLIB'"
      Call Perform_Copy
   End

   When eotype = "LOADCICS" then Do
      outdsn = "'SYS2.CICSLOAD'"
      Call Perform_Copy
   End

   Otherwise
      nop
End

Return
```

Figure 24. Promote User Exit (Part 2 of 3)

```
Perform_copy:
/* Initialize the FROM and TO datasets and perform copy             */

indsn = "'"proj"."togrp"."eotype"'"

Address ISPEXEC "LMINIT DATAID(FROMDSN) DATASET("indsn")"

If rc <> 0 then do
   Say "Error on LMINIT for FROM dataset  indsn  return code" rc
   exit 32
End

Address ISPEXEC "LMINIT DATAID(TODSN) DATASET("outdsn")"

If rc <> 0 then do
   Say "Error on LMINIT for TO dataset  indsn  return code" rc
   exit 32
End

/* Copy member from SCLM prod into live dataset                      */
Address ISPEXEC "LMCOPY FROMID("fromdsn") FROMMEM("eomember")
 TODATAID("todsn") TOMEM("eomember") REPLACE"

If rc <> 0 then do               /* If error on the Copy             */
   Say "Member" eomember "can not be copied to" outdsn
   max_rc = 16
End
Else                             /* Member was copied successfully   */
   Say eomember "has been copied to" outdsn

Return
```

Figure 25. Promote User Exit (Part 3 of 3)

The program uses the ISPF library management services to perform the copy and as such must be invoked in SCLM in one of two ways:

1. Using the ISPLNK call method as shown below:

```
PRMCOPY=SELECT,                                          C
PRMCPYCM=ISPLNK,                                         C
PRMCPYOP='CMD(PROMCPY1,',                                C
```

2. From a driver exit that uses a call method of TSOLNK as follows:

```
Address ISPEXEC 'SELECT CMD(PROMCPY1' parm ')'
```

Chapter 3. Additional project manager tasks

In addition to the tasks described in Chapter 1, “Defining the project environment,” on page 3, project managers can perform other tasks associated with defining and maintaining SCLM projects. This chapter describes other areas of responsibility in which project managers are involved. These include:

- Splitting VSAM data sets
- Backing up and recovering the project environment
- Synchronizing and maintaining accounting data sets
- Modifying the Delete from Group dialog interface
- Implementing Package Backout

Splitting project VSAM data sets

You might need to split the project VSAM data sets into multiple data sets because of security requirements, data set size, performance or changes in the way the project is being developed. By using multiple VSAM data sets in conjunction with flexible data set naming, cross-project support (for example, sharing common code) can be achieved.

The following steps make up the basic process for splitting project VSAM data sets:

1. Decide how you want to split the data sets. SCLM allows the VSAM data sets to be split on group boundaries.
2. Back up the data from the existing VSAM data sets for those groups using the new VSAM data sets. There are two ways to back up the data:
   a. You can use the SCLM export utility to export the contents of each group to the new data set. Because the Import utility deletes the contents of the export data set upon a successful completion of the import, you should make a backup of the export VSAM data sets using the IDCAMS reproduction utility (REPRO). By using this method, you do not need to update the contents of the PDS data sets. You only need to copy members from those groups that will be using the new VSAM data set. This method does not copy the audit records.

      Note: Using the REPRO function of the IDCAMS utility, you can split the audit data base at any point to create any number of smaller audit data bases. In order to use these smaller audit data bases, create alternate project definitions that specify the newly created audit data bases.
   b. You can use the IDCAMS REPRO utility to make a copy of each of the VSAM data sets used by the project. This method has the advantage of creating a backup of the project VSAM data sets. All records are copied to the new VSAM data set. While having the copies for all groups in the new VSAM data set is not a problem for SCLM, it does increase the size of the data set. These records can be deleted by setting up an alternate project definition that points only to the new VSAM data set and using the DELGROUP service to delete the groups that are not needed in that data set.
3. Make a backup copy of the project definition. This backup copy is needed to delete the data from the original VSAM data sets.
4. Update the project definition to add an FLMALTC macro for the new data sets and ALTC parameters on the groups that will be using those data sets.
5. Allocate the new VSAM data sets.
6. Assemble the new project definition.
7. Restore the data for the new VSAM data set from backup. How you do this depends on what method you used to back up the data:
   a. If you used the Export utility, use the Import utility to restore the data to the new VSAM data sets.

© Copyright IBM Corp. 1990, 2019

b. If you used the IDCAMS REPRO utility, use the REPRO utility to restore the data. You can do this before assembling the new project definition because it does not use any SCLM services.

8. Test the new project definition. Here are some suggestions for testing the new project definition:
   - Edit a member at the modified group. Create a new member, and also edit an existing member.
   - Run a build from the modified group.
   - Promote from the modified group.
9. Delete data from the existing VSAM data set for those groups that reference the new VSAM data set. You can do this by using a backup copy of the old project definition and the Delete from Group utility for each group that was moved.

   If you used the method of promoting to a new group, this step is not needed.

Backing up and recovering the project environment

The important point in backing up and recovering the project environment is that all the data remains synchronized. The project partitioned data sets contain related data, and the control data sets contain the control information for the PDS members. Thus, backing up and restoring the project environment means that the project partitioned data sets and the control data sets must be backed up and restored together.

The recommended procedure for backing up the project environment is to run a background job when no one is working within the hierarchy. You should determine how often to run this job. Remember that the topmost group of the hierarchy (the production group) usually contains most of the software and is usually frozen. You should back up the topmost groups whenever new data is promoted into the topmost groups. The lower groups in the hierarchy are subject to change much more often, and the code in the development groups usually changes daily. Perform backups for the lower groups based on your project's requirements. Again, remember that you must back up an entire group as a unit; this includes the project partitioned data sets and the control data sets.

Be careful when recovering a project environment. When you restore a group, it returns to the version that was in effect when you backed it up. This change can affect code below the restored group. Also the control data sets reflect the status of the group when it was backed up.

Synchronizing accounting data sets

The SCLM FLMCNTRL and FLMALTC macros allow you to select dual accounting data sets to be maintained using the ACCT and ACCT2 parameters. If an unrecoverable problem occurs with one of the primary accounting data sets, use the following JCL to restore the primary accounting data set.

```
//jobname  JOB (wkpkg,dpt,bin),'name'
//*************************************************************
//*                                                           *
//*  JCL TO RESTORE THE PRIMARY ACCOUNTING DATA SET FROM THE    *
//*  SECONDARY ACCOUNTING DATA SET.                           *
//*                                                           *
//*  SPECIFY THE UNCORRUPTED DATA SET AS YOUR INPUT DATA SET    *
//*                                                           *
//*************************************************************
//STEP1    EXEC PGM=IDCAMS
//INPUT    DD DISP=OLD,DSN=PROJ1.ACCOUNT2.FILE
//OUTPUT   DD DISP=OLD,DSN=PROJ1.ACCOUNT.FILE
//SYSPRINT DD SYSOUT=H
//SYSIN    DD *
 REPRO INFILE(INPUT) OUTFILE(OUTPUT)
/*
//
```

Figure 26. JCL to Restore the Primary Accounting Data Set

You can also use this JCL to initialize a backup data set for a project that is currently running under SCLM. If problems occur with the backup data set, SCLM issues warning messages. You must restore the backup data set when problems occur.

Maintaining accounting data sets

When groups or types are removed from the project definition, some accounting information from those groups or types can remain in the VSAM data sets for that project. In order to avoid having records that are no longer useful in the VSAM data sets, you should use the DELGROUP service to remove the VSAM records for any groups or types that are being removed from the project definition. This step should be performed before the groups and types are removed from the project definition.

If groups or types have been previously removed from the project definition, you can create an alternate project definition that includes a definition for the removed groups and types. This project definition can be used with the DELGROUP service to delete any remaining VSAM records.

Modifying the Delete from Group dialog interface

Given the power of Delete from Group, there are some restrictions in the dialog interface. Explanations for the restrictions and instructions for modifying the dialog to remove such restrictions follow.

The Group field is restricted to disallow patterns. To remove this restriction:

1. Edit the panel FLMDDG#P. It is recommended that you update the DTL version instead of the generated panel to avoid losing the changes if the panel is regenerated. See *z/OS ISPF Dialog Tag Language Guide and Reference* for more information.
2. Replace the line:

   ```
   <dtafld datavar=DGLEVEL usage=both
          entwidth=8 pmtwidth=12 >&lib_prompt;
   ```

   with the lines:

   ```
   <dtafld datavar=DGLEVEL usage=both
          deswidth=41 entwidth=9 pmtwidth=12 >&lib_prompt;
     <dtafldd>(Pattern can be used)
   ```

   or with the lines:

   ```
   <dtafld datavar=DGLEVEL usage=both
          deswidth=41 entwidth=17 pmtwidth=12 >&lib_prompt;
     <dtafldd>(Pattern can be used)
   ```

   depending upon how you resolve the next restriction. They should be consistent if patterns are allowed.
3. Edit the imbed FLMZDG#P, and replace the line:

   ```
   VER(&DGLEVEL,NB,NAME)
   ```

   with the line:

   ```
   VER(&DGLEVEL,NONBLANK)
   ```

The Type and Member fields are restricted to 9 characters; FLMCMD and FLMLNK allow up to 17 characters. To remove this restriction:

1. Edit the panel FLMDDG#P. It is recommended that you update the DTL version instead of the generated panel to avoid losing the changes if the panel is regenerated. See *z/OS ISPF Dialog Tag Language Guide and Reference* for more information.
2. Replace the lines:

```
<dtacol entwidth=8 pmtwidth=12
        deswidth=49 fldspace=11 >
```

   with the lines:

```
<dtacol entwidth=17 pmtwidth=12
        deswidth=41 fldspace=11 >
```

The Delete mode always defaults to Report when the panel appears. To remove this restriction, remove the following lines from the FLMZDG#P panel imbed:

```
&DMODE = 'REPORT'
&DMODEV = '2'
```

Implementing package backout

This topic describes how to implement package backout.

1. Determine the TYPE (for example, ARCHPACK) to hold the package high-level architecture members. If required allocate the appropriate data sets.
2. Update the project definition for this type to have the parameter ISAPACK=Y on the FLMTYPE macro. When an architecture member using this type is promoted, the package backout is invoked.
3. Determine the types of files (such as Object, load libraries) that are to be backed up during the promotion of a package high-level architecture member. The Project definition for these file TYPES should be updated to specify the BACKUP=Y on the FLMTYPE macro.
4. Determine at which groups (for example, production) the package backout is to be implemented, and the group that the members will be backed up to.

   In the Project definition for these groups, use the BKGRP=*group_name* parameter on the FLMGROUP macro to specify the group to which the members will be backed up. These new backup groups must be added to the project definition, so add an FLMGROUP macro for them. Make sure the group is key. Use the group that is being backed up as the PROMOTE= group.

   For example, to back up RELEASE into a group called BACKGRP:

```
BACKGRP  FLMGROUP AC=(P),KEY=Y,PROMOTE=RELEASE
RELEASE  FLMGROUP AC=(P),KEY=Y,BKGRP=BACKGRP
```

5. Determine if member-level restore is to be implemented to allow individual members to be restored instead of an entire package. If it is required, update the FLMGROUP macro to have BKMBRLVL=Y.
6. Create the backup libraries for the TYPES you have specified with BACKUP=Y for the groups for which package backout has been specified. The data set names will have the format *project_name.group_name.ds_type*, where *group_name* is the value specified on the BKGRP= parameter for each group. Allocate the backup libraries with the same attributes as the libraries that are being backed up.
7. Determine the File type to contain the package backout details. Add the parameter PACKFILE=Y to the Project definition for this type. The PACKFILE flag must only be specified on one FLMTYPE in the project definition, for example `PACKFILE FLMTYPE PACKFILE=Y`

   Create a library of this type for the groups for which package backout has been specified. The data set names will have the format *project_name.group_name.ds_type*, where *group_name* is the value specified on the BKGRP= parameter for each group and *ds_type* is the type on the FLMTYPE macro with PACKFILE=Y. Allocate this data set with LRECL=130 and RECFM=FB.

8. Determine if package reuse is to be used. If so set 'REUSEDAY=*nnnn*' on the FLMTYPE macro that has the PACKFILE=Y specified.
9. Reassemble and link the project definitions.

Figure 27 on page 73 shows a sample project definition that allows for package backout.

```
ARCHDEF  FLMTYPE
SOURCE   FLMTYPE  EXTEND=MACROS
MACROS   FLMTYPE
LIST     FLMTYPE
ARCHPACK FLMTYPE  ISAPACK=Y
OBJ      FLMTYPE  BACKUP=Y
LMAP     FLMTYPE
LOAD     FLMTYPE  BACKUP=Y
PACKFILE FLMTYPE  PACKFILE=Y
⋮
DEV1     FLMGROUP AC=(P,A,LONGNAME),KEY=Y,PROMOTE=TEST
DEV2     FLMGROUP AC=(P,A),KEY=Y,PROMOTE=TEST
TEST     FLMGROUP AC=(P),KEY=Y,PROMOTE=RELEASE
BACKGRP  FLMGROUP AC=(P),KEY=Y,PROMOTE=RELEASE
RELEASE  FLMGROUP AC=(P),KEY=Y,BKGRP=BACKGRP
```

Figure 27. Sample project definition

Chapter 4. Converting projects to SCLM

To convert an existing project to an SCLM-controlled project, bring the project groups under control one at a time beginning with the top layer of the hierarchy, which is the production (frozen) group, and work downward. Most projects to be converted already exist in some kind of logical hierarchy. If all production source code is stored in one logical place and code under development is stored elsewhere, you have at least a two-layer hierarchy. Before migration can begin, you must place the source code to be converted into partitioned data sets.

There are many advantages to using the preceding method. First, you can bring a project under SCLM control in discrete steps, over a period of time. Second, SCLM can locate integrity problems in the existing hierarchy and fix them systematically during the conversion process. Third, SCLM performs the conversion using the same tools that developers use in the normal development process. Thus, you ensure consistency within the hierarchy, and you become familiar with SCLM. Finally, from the conversion process, you receive an indication of the performance that you can expect of SCLM during the development process.

Prerequisites for existing hierarchies

The best time to begin the conversion process is when the components to be controlled are concentrated in a small number of groups—for example, immediately following a software release. The following actions help you prepare a hierarchy for the conversion process.

- Create the project definition to be used with the converted hierarchy. See Chapter 1, “Defining the project environment,” on page 3, for details.
- Verify that all partitioned data sets to be controlled are available online. If the data is not in partitioned data sets, allocate partitioned data sets by following “Step 5: Allocate the project partitioned data sets” on page 12, and copy data from the existing data sets to the partitioned data sets.
- Delete all unnecessary data from the libraries being converted.
- If you intend to use non-key groups in the converted hierarchy, ensure that they do not contain any data before conversion.

Create alternate project definitions

You need to create several alternate project definitions to complete the conversion process. Because the SCLM migration utility can only run against development libraries, which are in the lowest layer of the hierarchy, you need an alternate project definition for each layer of the proposed hierarchy. The first alternate project definition you use defines only the topmost group. That group becomes a development group. The second project definition defines the topmost group and those groups that promote into it, and so on. You do not need to define non-key groups in the alternate project definitions you use for the conversion process because they should not contain any members.

Create architecture definitions for the project

Although you can perform the conversion process without architecture definitions, their creation can greatly simplify the conversion process as well as support future development needs. Define a set of architecture members first for the code in the topmost group of the hierarchy. These architecture members must reference only members that are present in the topmost group because only those members are visible during the first group conversion.

To determine which architecture members you need, perform the following steps:

© Copyright IBM Corp. 1990, 2019

1. Determine whether all the build translators can use the default translator options in the language definitions. If they can, you do not need compilation control architecture members.
2. Determine the contents of every load module to be controlled. The IEHLIST utility prints the names of all objects in a load module.
3. Produce a linkage edit control architecture member for every load module, and reference each object (actually compilable source members) with an INCLD statement. Use the INCL statement in place of INCLD to reference compilation control architecture members if they are created above.
4. Produce high-level architecture members as needed to control any non-translatable data or data that is not included in load modules.
5. Produce a high-level architecture member and reference each linkage edit control architecture member and high-level architecture member defined above with an INCL statement.

The high-level architecture member created in Step 5 now defines, through its dependencies, the entire application architecture.

After you create the architecture members for the topmost group, you might need to add modifications in the lower groups of the hierarchy. Members that were added during the development process and were not moved to the topmost group may require additional architecture members. You must introduce architecture modifications in the group requiring the change. This action allows the architecture for the hierarchy to match the members controlled in the hierarchy. See Chapter 11, “Architecture definition,” on page 271 for a description of the process and syntax for defining architecture members.

Register existing PDS members with SCLM

Editable members and noneditable members are processed in separate and unique ways by SCLM.

Editable members, such as source members, are not created by the SCLM build function. Editable members must be registered with SCLM through the migration utility. Both the language associated with the member and a change code (only if you have a change code verification routine) are required as input to the migration utility. TEXT can be used as the language of members that do not need to be compiled, assembled, or processed, such as panels and messages. Call the migration utility for each library containing editable members.

The SCLM Build function creates noneditable members. Object code, listings, and load modules are examples of noneditable members. The SCLM build function must be called to create all of the noneditable members to be tracked within the hierarchy. If all of the customization related to language translators is complete and has been tested, run the build processor in the unconditional mode using the topmost architecture member for your application. This unconditional build will identify all build errors that exist. If errors are anticipated and the application is large, use architecture members with smaller scopes. For example, use an LEC architecture member rather than an HL. Using the conditional mode of the build processor causes processing to stop when a member containing an error is encountered.

The normal process is to migrate source members into SCLM and then generate the outputs using the SCLM Build function. There may be occasions, however, where you would like to use SCLM to manage object and load modules for which the source code no longer exists. There are two ways of doing this.

The first method uses a 'dummy' language definition with an FLMLANGL macro, but no FLMTRNSL macros. An example of this is provided as member FLM@OBJ in the ISP.SISPMACS data set included with SCLM. This language definition allows you to migrate object and load modules into SCLM as editable members in the same manner that source modules are introduced.

Note: Special care must be taken when using versioning in a project that has stored object and load modules in this manner. SCLM will consider the members to be editable and will allow versioning to occur if specified. This may cause errors in SCLM version processing. The second method is a better choice when versioning is being used in the project.

The second method involves migrating the object and load modules into a temporary type and then using the SCLM Build function to copy them to the target type. The SCLM build process will mark the copied object and load modules as non-editable. This solution is a better choice for projects with versioning in

use. Member FLM@COPY in the ISP.SISPMACS data set can be used to store object modules into SCLM in this manner. It can be modified for use with load modules. This language definition will migrate the members into a temporary type as editable members. SCLM will allow the migrate because, like the FLM@OBJ language definition, there is no FLMTRNSL macro with FUNCTN=PARSE and therefore no parser will be invoked. The FLMTRNSL macro for the Build function calls IEBGENER to copy the modules from one SCLM type to the other as non-editable outputs.

Introducing fixes to the converted hierarchy

During the conversion process, SCLM might discover integrity errors existing in the current development hierarchy. If it encounters these errors in the topmost group of the hierarchy, the errors have an effect on the rest of the conversion process. You can encounter two kinds of errors:

- Dependency errors for editable members. Errors can be caused when an included member or macro cannot be found within the hierarchy. If you want the missing member tracked in the hierarchy, you must copy the correct version of the included member to the group being converted. If you do not want the missing member tracked in the hierarchy, define it to SCLM using the FLMSYSLB macro and the FLMCPYLB macro in the language definition of the member.
- Compile errors, or any similar translator errors in any group, located during the build process. The errors must be corrected before proceeding with the conversion. Use the listings produced by build to locate and correct the errors. After making the correction rebuild the members that contained the errors.

Chapter 5. Language definition considerations

SCLM can be tailored to support languages other than those listed in the examples provided with the product. By creating a *language definition* as part of the project definition, you specify to SCLM the languages that will be used for the project. Language definitions provide SCLM with language-specific control information such as the language name and the definition of the language translators.

The language definition describes language-specific processing in two ways:

- From a data-flow perspective, the language definition specifies all data sets used as input to or output from various SCLM processes such as Parse, Build, Promote, and Delete.
- From a procedural perspective, the language definition specifies the translators (for example, parsers or compilers) that are invoked to process your SCLM-controlled data. The order in which those translators are invoked and the options to be passed to the translators are defined in the language definition.

You must provide SCLM a language definition for each language (PL/I, COBOL, Link-Edit, and so on) that you want SCLM to support. In most cases, you can make minor changes to sample SCLM language definitions provided with ISPF.

A language definition consists of a collection of the following definitions:

- System library definitions
- Language identifier definition
- Include set definitions
- Translator definitions
- Allocation definitions
- Copy library definitions

Because a macro exists for each of these definitions and because each macro accepts a number of different parameters, you can specify a large variety of language definitions. The language definitions provided with the product are examples that can serve as a reference in the construction of language definitions for a specific application and environment.

To determine what modifications you can make to the language definition, become familiar with the parameters of the language definition macros as documented in Chapter 21, “SCLM macros,” on page 491. Typically, to write a new language definition, you would copy an old language definition and then modify it to meet your specific needs.

In the remainder of this chapter, several language definitions are examined more closely in order to describe some of the implementations of language definitions. Topics discussed in this chapter include:

- Using multiple translators in a language definition
- Invoking user-defined parsers
- Processing conditionally saved components
- Specifying the location of included members
- Tracking dynamic includes
- Using input list translators.

Using multiple translators in a language definition

You can use the FLMTRNSL macro to define translators for a language. The parameters of the FLMTRNSL macro define all the attributes needed to call a given translator. The FLMTRNSL FUNCTN parameter defines the function or purpose for which a translator is called. SCLM uses translators for the following functions:

© Copyright IBM Corp. 1990, 2019

- Parsing source code to determine statistics and dependency information. SCLM calls these translators when a member is saved in the editor or migrated (dialog function or MIGRATE service) or saved with the SAVE service.
- Translating one form of code into another, for example:
  - Source code to object code and listings
  - Script input to a formatted document
  - Object code to load modules

  SCLM calls these translators during the build process.
- Verifying data. A verify translator performs validation in addition to the default SCLM validation. The verify translator is invoked before the translation step (such as compiling and linking) of build, and before the copy phase of promote.
- Copying data. SCLM calls these translators during the promote process. The data can be either PDS members controlled directly by SCLM or non-PDS data that includes an intermediate form of compilation units and external data identified to SCLM via a build translator.
- Purging data. SCLM calls these translators during the promote process. The data can be either PDS members controlled directly by SCLM or non-PDS data that includes an intermediate form of compilation units and external data identified to SCLM via a build translator.

The translators required for a language are language-specific. Some languages require parse and build translators while others need parse, build, copy, and purge translators.

Most SCLM-supplied example language definitions have two translators defined. The first identifies the parser to be invoked, and the second identifies the translator to be invoked during a build. Language definitions can be created for the invocation of one or more translators during the parse, build, copy, verify, or purge functions. For each of these functions, the translators are invoked in the order in which they appear in the language definition. Within a function in the language definition, a translator can pass data on to the next translator invoked by that function within the language definition. This capability allows you to customize the SCLM product for unique processing requirements in your project.

When connecting SCLM translators in a language definition, make sure they are ordered so that they will execute in the correct sequence. If used for build, you should order the preprocessing and compile steps as you would in a CLIST or JCL.

If multiple-step language definitions specify more than one translator to be invoked during a build, make sure the DDNAMEs for outputs to be copied into the project hierarchy are unique. If the same DDNAME is used, only the outputs from the last translator will be copied to the hierarchy. For more information, refer to “Using DDnames and DDname substitution lists” on page 102.

Figure 28 on page 81 shows a language definition that uses multiple translators. The DB2 preprocessor (DSNHPC) creates a COBOL source data set using the SYSCIN ddname. The next translator, the COBOL II compiler IGYCRCTL, reads in the SYSCIN data set.

Note that the receiving translator defines SYSCIN as IOTYPE=U, meaning that SYSCIN has already been allocated in a previous translator step.

```
*********************************************************************
* COBOL II WITH DB2 PREPROCESSOR  - LANGUAGE DEFINITION FOR SCLM
*
* DB2 OUTPUT IS PASSED VIA THE 'SYSCIN' DD ALLOCATION TO COBOL II.
* POINT THE FLMSYSLB MACRO(S) AT ALL 'STATIC' COPY DATASETS.
* CUSTOMIZE THE 'OPTIONS' AND 'GOODRC' FIELDS TO YOUR STANDARDS.
* ADD THE 'DSNAME' FIELD IF THE TRANSLATOR IS IN A PRIVATE LIBRARY.
* WHEN A NEW TRANSLATOR VERSION REQUIRES TOTAL RECOMPILATION FOR THIS
* LANGUAGE, THE 'VERSION' FIELD ON FLMLANGL SHOULD BE CHANGED.
*********************************************************************
* CHANGE ACTIVITY:                                                  *
*                                                                   *
*********************************************************************
*
         FLMLANGL    LANG=DB2COB2,ALCSYSLB=Y
*
* PARSER TRANSLATOR
*
          FLMTRNSL   CALLNAM='SCLM COBOL PARSE',                     C
               FUNCTN=PARSE,                                         C
               COMPILE=FLMLPCBL,                                     C
               PORDER=1,                                             C
               OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,)
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
* BUILD TRANSLATORS
*
*         --DB2 PREPROCESSOR INTERFACE--
          FLMTRNSL   CALLNAM='DB2 PREPROCESS',                       C
               FUNCTN=BUILD,                                         C
               COMPILE=DSNHPC,                                       C
               VERSION=1.0,                                          C
               GOODRC=4,                                             C
               PORDER=3,                                             C
               OPTIONS=(HOST(COB2))
*  1       -- N/A --
           FLMALLOC  IOTYPE=N
*  2       -- N/A --
           FLMALLOC  IOTYPE=N
*  3       -- N/A --
           FLMALLOC  IOTYPE=N
*  4       -- SYSLIB --
           FLMALLOC  IOTYPE=I,KEYREF=SINC
*  5       -- SYSIN --
           FLMALLOC  IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,          C
               RECNUM=2000
*  6       -- SYSPRINT --
           FLMALLOC  IOTYPE=W,RECFM=FBA,LRECL=121,                    C
               RECNUM=9000,PRINT=I
*  7       -- N/A --
           FLMALLOC  IOTYPE=N
*  8       -- SYSUT1 --
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=800,RECNUM=9000
*  9       -- SYSUT2 --
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=800,RECNUM=9000
* 10       -- SYSUT3 --
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=800,RECNUM=9000
```

Figure 28. COBOL II with DB2 Preprocessor (Part 1 of 2)

```
* 11       -- N/A --
           FLMALLOC  IOTYPE=N
* 12       -- SYSTERM --
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
* 13       -- N/A --
           FLMALLOC  IOTYPE=N
* 14       -- SYSCIN --
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,                        C
               RECNUM=9000,DDNAME=SYSCIN
* 15       -- N/A --
           FLMALLOC  IOTYPE=N
* 16       -- DBRMLIB--
           FLMALLOC  IOTYPE=P,DDNAME=DBRMLIB,MEMBER=@@FLMONM,           C
               DFLTTYP=DBRM,KEYREF=OUT1,                                C
               RECFM=FB,LRECL=80,RECNUM=5000,DIRBLKS=1
*
*         --COBOL II INTERFACE--
*
          FLMTRNSL   CALLNAM='COBOL II COMPILER',                       C
               FUNCTN=BUILD,                                            C
               COMPILE=IGYCRCTL,                                        C
               VERSION=2.0,                                             C
               GOODRC=0,                                                C
               PORDER=3,                                                C
               OPTIONS=(XREF,LIB,APOST,NODYNAM,LIST,NONUMBER,NOSEQ)
*
* DDNAME ALLOCATION (USING DDNAMELIST SUBSTITUTION)
*
*  1       (* SYSLIN *)
           FLMALLOC  IOTYPE=O,KEYREF=OBJ,RECFM=FB,LRECL=80,             C
               RECNUM=5000,DFLTTYP=OBJ,DDNAME=SYSLIN
*  2       (* N/A *)
           FLMALLOC  IOTYPE=N
*  3       (* N/A *)
           FLMALLOC  IOTYPE=N
*  4       (* SYSLIB *)
           FLMALLOC  IOTYPE=I,KEYREF=SINC,DDNAME=SYSLIB
*  5       (* SYSIN *)
           FLMALLOC  IOTYPE=U,DDNAME=SYSCIN
*  6       (* SYSPRINT *)
           FLMALLOC  IOTYPE=O,KEYREF=OUT2,RECFM=FBA,LRECL=133,          C
               RECNUM=25000,PRINT=Y,DFLTTYP=LIST,DDNAME=SYSPRINT
*  7       (* SYSPUNCH *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
*  8       (* SYSUT1 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
*  9       (* SYSUT2 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 10       (* SYSUT3 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 11       (* SYSUT4 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 12       (* SYSTERM *)
           FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
            FLMCPYLB NULLFILE
* 13       (* SYSUT5 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 14       (* SYSUT6 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 15       (* SYSUT7 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
```

Figure 29. COBOL II with DB2 Preprocessor (Part 2 of 2)

Invoking user-defined parsers

SCLM allows you to replace an SCLM-supplied source parser with a user-defined source parser. This option is important when you are defining a new language for a project because such a language is likely to have a syntax unlike any of the languages that the SCLM-supplied parsers can recognize.

When you write a new parser for a language, you must:

1. Define the information tracked by SCLM in terms of the syntax of the language you want to support.
2. Write a program, based on the information you defined, that passes the statistical and dependency information for a member written in this new language to SCLM. This program is called a parser.
3. Tell SCLM how to invoke your parser.

Figure 31 on page 86 to Figure 40 on page 94 contain a parser, written in PL/I, for the ISPF skeleton (SKELS) language. This section works through the three preceding steps using the SKELS parser as an example.

Several user-modifiable parsers, written in REXX, are included with SCLM. FLMLRASM (Assembler), FLMLRCBL (COBOL), FLMRC2 (workstation C/C++ and resource files), FLMLRIPF (workstation help files), FLMLRC37 (C/370), and FLMLRCIS(C/C++ for MVS with include set support) are described in Chapter 22, "SCLM translators," on page 565. Chapter 7, "Understanding and using the customizable parsers," on page 131 contains information on modifying the REXX parsers.

Defining information tracked by SCLM

SCLM tracks four kinds of information for each module:

- Statistical information

  Statistical information includes such data as the total lines and the number of comments in the module. See Part 2, "Developer's Guide," on page 137 for a description of the 10 statistics kept by SCLM.

- Dependency information

  SCLM tracks two types of dependency information. The first is the name of the members that are included by a member. The second is the include set that is used to find the include. This information is used when a member is built or promoted. See "Specifying the locations of included members" on page 95 for more information on the include information kept by SCLM.

- Change code information

  The change code information is a list of change codes associated with members under SCLM control. These change codes are optional unless the project manager has defined a change code verification routine requiring them. Includes and change codes for a member can be viewed with the Library Utility.

- User-defined information

  User-defined information is a list of free-form records derived from the member via the parse translator and stored in the accounting record. When writing a new parser, define exactly how the parser derives this information from a module.

Writing the parser

Consider these things when you write your own parser:

- If any information is to be passed to the parser from SCLM, it is passed through a single parameter string as if your program had been invoked from TSO as:

```
CALL program 'parameter list'
```

- You can use the SCLM variables to pass information to the parser about the module to be parsed.
- You can allocate any files you need (including the module to be parsed) to ddnames or pass the data set names directly through the parameter list.
- SCLM allocates space for an array and a structure. It is up to the parser to place statistical and dependency information in the array and the structure as it parses the module. SCLM can pass the address of the structure and the array to the parser through the parameter list string. If the parser returns a successful return code, SCLM moves the parsed information into the accounting record of the module.

The SKELS parser example consists of four routines. Together, these routines perform the work needed to parse an ISPF skeleton as we have described.

GETPTRS
Takes the addresses from the parameter list and places them in the appropriate pointer variables.

INITIAL
Initializes the counter variables and the parse structure (STAT_INFO).

PARSE
Reads the lines of the skeleton one at a time, and saves any statistical or dependency information it finds.

WRAPUP
Prepares the parse structure and the parse array (LIST_INFO) to be passed back to SCLM.

Telling SCLM how to invoke your parser

You need to add a few SCLM macros to your project definition for SCLM to invoke your parser. The macros used to define the SKELS parser are shown in Figure 30 on page 85 For your parser, you need:

- An FLMLANGL to define your language (if it is not already there)
- An FLMTRNSL to define your parser
- An FLMALLOC for each ddname required by your parser
- An FLMCPYLB for each data set name you want to specify.

In Figure 30 on page 85 you can examine the keywords on the macros to see how they are used.

On the FLMLANGL macro, the LANG parameter indicates the string (in this case it is SKELS) that needs to be given to SCLM when you want SCLM to treat a module like a skeleton. The BUFSIZE parameter is the number of elements in the LIST_INFO array that SCLM passes to the parser.

On the FLMTRNSL macro, the COMPILE and DSNAME parameter tell SCLM that the parser can be found in SCLM.PROJECT.LOAD(FLM@SKLS). The OPTIONS parameter contains three SCLM variables: @@FLMSTP, @@FLMLIS, and @@FLMSIZ. When the parser converts the character string values of @@FLMLIS and @@FLMSTP to fullword binary integers, the result is the addresses of the LIST_INFO array and the STATS_INFO structure, respectively. The value of @@FLMSIZ is the number of bytes allocated for the LIST_INFO array.

The first FLMALLOC macro allocates the module to be parsed to ddname SSOURCE. The SKELS parser looks at this ddname for the skeleton source. The second FLMALLOC macro allocates an error listings file. If an error occurs during the parse, the SKELS parser writes an explanatory message and provides a recommended solution. If the SKELS parser passes back a return code greater than that specified on the GOODRC parameter of the FLMTRNSL macro, the contents of this listings file are written to the edit listings file for the parse. This is how you can pass messages and information about the parse to your users.

```
/*******************************************************************/
/* ISPF SKELETON LANGUAGE DEFINITION                               */
/*******************************************************************/
         FLMLANGL    LANG=SKEL,VERSION=V2.3,BUFSIZE=50

  PARSER TRANSLATOR

          FLMTRNSL   CALLNAM='SKEL PARSER',                          C
               COMPILE=FLM@SKLS,                                     C
               DSNAME=SCLM.PROJECT.LOAD,                             C
               FUNCTN=PARSE,                                         C
               PORDER=1,                                             C
               GOODRC=0,                                             C
               VERSION=V1R0M0,                                       C
               OPTIONS='/@@FLMSTP,@@FLMLIS,@@FLMSIZ,'
           (* SOURCE      *)
           FLMALLOC IOTYPE=A,DDNAME=SSOURCE
           FLMCPYLB @@FLMDSN(@@FLMMBR)
           (* LISTING     *)
           FLMALLOC IOTYPE=W,RECFM=VBA,LRECL=133,                    C
               RECNUM=6000,DDNAME=ERROR,PRINT=Y
```

Figure 30. SKELS Parser Definition

```
PROCESS;
/*********************************************************************/
/***                                                               ***/
/***  Program:   PSKELS                                            ***/
/***                                                               ***/
/***  Purpose:   Performs an SCLM parse of ISPF skeletons after    ***/
/***             SCLM edit and during migration of source to SCLM.***/
/***                                                               ***/
/***  Inputs:    A parameter list containing addresses of a        ***/
/***             structure and a variable-length array into which ***/
/***             parse information is placed.  The length of the  ***/
/***             array, in bytes, is also passed.                  ***/
/***                                                               ***/
/***             In addition, source from the member to be parsed ***/
/***             is read from ddname SSOURCE.                      ***/
/***                                                               ***/
/***  Outputs:   The structure and array are filled with parse     ***/
/***             information by this program.  Any error messages ***/
/***             are written to ddname ERROR.                      ***/
/***                                                               ***/
/***  Retcode:   A fullword integer value, indicating the overall ***/
/***             success of the parse, is returned in register 15.***/
/***                                                               ***/
/***             0 = Successful parse; parse information is        ***/
/***                 returned in the structure and array.          ***/
/***                                                               ***/
/***             4 = Variable-length array was too small to hold   ***/
/***                 all of the parsed information.  Not all       ***/
/***                 information was passed back to SCLM.  The     ***/
/***                 number of elements needed is shown in the     ***/
/***                 listings data set.                            ***/
/***                                                               ***/
/***                 To correct this problem, either:              ***/
/***                                                               ***/
/***                   * Increase the value of BUFSIZE in the      ***/
/***                     FLMLANGL macro for this parser, or        ***/
/***                                                               ***/
/***                   * Break the skeleton being parsed into      ***/
/***                     smaller skeletons and use )IM to join     ***/
/***                     them back together.                       ***/
/***                                                               ***/
/***  Logic:     1) Obtain addresses of structure and array from   ***/
/***                parameter list.                                ***/
/***             2) Initialize counters in structure.              ***/
/***             3) For each line of skeleton source:              ***/
/***                a) Increment appropriate counters.             ***/
/***                b) If record starts with )IM, find and save    ***/
/***                   imbedded skeleton name.                     ***/
/***                c) Scan the record for variable names and      ***/
/***                   save in a program array any new names.      ***/
/***                d) If record starts with )DEFAULT, get new     ***/
/***                   '&' and ')' characters.                     ***/
/***             4) Calculate summary statistics.                  ***/
/***             5) Write an 'END ' element to end of parse array.***/
/***             6) Return.                                        ***/
/***                                                               ***/
/*********************************************************************/
```

Figure 31. Parser for ISPF skeletons (Part 1 of 8)

```
PSKELS: PROC(PARMLIST) OPTIONS(MAIN);
   DCL PARMLIST     CHAR(255) VAR; /* Parameter list               */
   DCL PARMLISTx    CHAR(255) VAR; /* Copy of the parameter list   */
   DCL PAREN        CHAR(1),       /* Contains ')' special char    */
       NAME         CHAR(8),       /* Contains a referenced name   */
       NAMECHRS     CHAR(39),      /* Valid name characters        */
       RECORD       CHAR(80),      /* Output buffer for error list */
       STAT_PTR     POINTER,       /* Points to stats structure    */
       LIST_PTR     POINTER,       /* Points to parse array        */
       NON_COM_READ BIT(1),        /* Prolog flag                  */
       EOF          BIT(1),        /* End-of-file flag             */
       (I,J,K)      FIXED BIN(31), /* Simple counters              */
       USED_ELMTS   FIXED BIN(31), /* Number of parse array        */
                                   /*    elements used so far      */
       LISTLEN      FIXED BIN(31), /* Total number of available    */
                                   /*    parse array elements      */
       RETCODE      FIXED BIN(31); /* Return code                  */
   DCL ADDR         BUILTIN,
       INDEX        BUILTIN,
       LENGTH       BUILTIN,
       MIN          BUILTIN,
       REPEAT       BUILTIN,
       SUBSTR       BUILTIN,
       VERIFY       BUILTIN,
       PLIRETC      BUILTIN;
   DCL SSOURCE      FILE STREAM INPUT;
   DCL ERROR        FILE STREAM PRINT;
   DCL FXB_OV       FIXED BIN(31), /* Fullword integer             */
       PTR_OV       POINTER BASED(ADDR(FXB_OV));
                                   /* Pointer variable overlay on  */
                                   /*    top of a fullword integer */
                                   /*    variable                  */
   %INCLUDE(STATINFO);
   %INCLUDE(LISTINFO);
   RETCODE = 0;
   CALL GETPTRS;
   CALL INITIAL;
   CALL PARSE;
   CALL WRAPUP;
   CALL PLIRETC(RETCODE);
```

Figure 32. Parser for ISPF skeletons (Part 2 of 8)

```
GETPTRS: PROC;
/*********************************************************************/
/***                                                               ***/
/***  Routine:   GETPTRS                                           ***/
/***                                                               ***/
/***  Purpose:   Converts the information passed to this program   ***/
/***             into addresses and array length information.      ***/
/***                                                               ***/
/***  Inputs:    A varying length string containing parameters in  ***/
/***             the following format:                             ***/
/***                                                               ***/
/***               '<stat_ptr>,<list_ptr>,<length>,'               ***/
/***                                                               ***/
/***                  where stat_ptr is the EBCDIC representation  ***/
/***                                of the address of the static   ***/
/***                                portion of the account         ***/
/***                                record for this member,        ***/
/***                        list_ptr is the EBCDIC representation  ***/
/***                                of the address of the          ***/
/***                                dynamic portion of the         ***/
/***                                account record, and            ***/
/***                        length   is the number of bytes        ***/
/***                                allocated to the dynamic       ***/
/***                                portion of the account         ***/
/***                                record.  This value is equal   ***/
/***                                to 228 times the number of     ***/
/***                                elements in that array.        ***/
/***                                                               ***/
/***             Note that this format is consistent with the      ***/
/***             OPTIONS keyword on the FLMTRNSL macro that        ***/
/***             specifies how to invoke this parser.              ***/
/***                                                               ***/
/***  Outputs:   The three variables, STAT_PTR, LIST_PTR and       ***/
/***             LISTLEN are set from the values in the            ***/
/***             parameter list.                                   ***/
/***                                                               ***/
/***  Logic:     1) Find the first comma.                          ***/
/***             2) Convert the contents of the character string   ***/
/***                before that comma into integer format.  For    ***/
/***                example, the string '19,' would be converted   ***/
/***                into an integer (X'00000013')                  ***/
/***             3) Find the next comma.                           ***/
/***             4) Convert the contents of the character string   ***/
/***                before that comma into integer format.         ***/
/***             5) Find the last comma.                           ***/
/***             6) Convert the contents of the character string   ***/
/***                before that comma into integer format.         ***/
/***                                                               ***/
/***  Note:      We take advantage of PL/I's ability to convert    ***/
/***             a number in character string format into a        ***/
/***             fullword binary value.                            ***/
/***                                                               ***/
/*********************************************************************/
   PARMLISTX = PARMLIST;
   I = INDEX(PARMLIST,',');
   FXB_OV = SUBSTR(PARMLIST,1,I-1);
   STAT_PTR = PTR_OV;
   PARMLIST = SUBSTR(PARMLIST,I+1,LENGTH(PARMLIST)-I);
```

Figure 33. Parser for ISPF skeletons (Part 3 of 8)

```
   I = INDEX(PARMLIST,',');
   FXB_OV = SUBSTR(PARMLIST,1,I-1);
   LIST_PTR = PTR_OV;
   PARMLIST = SUBSTR(PARMLIST,I+1,LENGTH(PARMLIST)-I);

   I = INDEX(PARMLIST,',');
   LISTLEN = SUBSTR(PARMLIST,1,I-1);
   LISTLEN = LISTLEN / 228;
END GETPTRS;
INITIAL: PROC;
/*********************************************************************/
/***                                                               ***/
/***  Routine:   INITIAL                                           ***/
/***                                                               ***/
/***  Purpose:   Initializes the counters and variables to be      ***/
/***             used during the parse.                            ***/
/***                                                               ***/
/***  Inputs:    None.                                             ***/
/***                                                               ***/
/***  Outputs:   Initialized variables.                            ***/
/***                                                               ***/
/*********************************************************************/
   STATINFO.LINES.TOTAL      = 0; /* # of lines in the skeleton     */
   STATINFO.LINES.COMMENT    = 0; /* # of lines starting with )CM   */
   STATINFO.LINES.NON_COMMENT= 0; /* # lines not starting w/ )CM    */
   STATINFO.LINES.BLANK      = 0; /* # lines starting with )BLANK   */
   STATINFO.LINES.PROLOG     = 0; /* # lines before 1st noncomment  */
      /**/
   STATINFO.STMTS.TOTAL      = 0; /* = LINES.TOTAL                  */
   STATINFO.STMTS.COMMENT    = 0; /* = LINES.COMMENT                */
   STATINFO.STMTS.CONTROL    = 0; /* # of lines starting with )     */
   STATINFO.STMTS.ASSIGNMENT = 0; /* = 0                            */
   STATINFO.STMTS.NON_COMMENT= 0; /* = LINES.NON_COMMENT            */
      /**/
   USED_ELMTS = 0;
      /**/
   NAMECHRS   = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789@#$';
   PAREN = ')';
END INITIAL;
PARSE: PROC;
/*********************************************************************/
/***                                                               ***/
/***  Routine:   PARSE                                             ***/
/***                                                               ***/
/***  Purpose:   Parses the skeleton and places the result in the  ***/
/***             account record structures whose addresses were    ***/
/***             passed to the program.                            ***/
/***                                                               ***/
/***  Inputs:    Skeleton source from ddname SSOURCE.              ***/
/***                                                               ***/
/***  Outputs:   Parse results in structure STAT_INFO and array    ***/
/***             LIST_INFO.                                        ***/
/***                                                               ***/
/***  Logic:     1) Read each record of the skeleton.  For each    ***/
/***                line read, increment the appropriate           ***/
/***                counters.                                      ***/
/***                                                               ***/
/*********************************************************************/
```

Figure 34. Parser for ISPF skeletons (Part 4 of 8)

```
   OPEN FILE(SSOURCE);
   EOF = '0'B;
   NON_COM_READ = '0'B;
   ON ENDFILE(SSOURCE) EOF = '1'B;
   GET FILE(SSOURCE) EDIT(RECORD) (A(80));
   DO WHILE (¬EOF);
/*****************************************************************/
/***  Perform this loop for each record in the skeleton.       ***/
/*****************************************************************/
/***  Increment total line counter.                            ***/
/*****************************************************************/
      STATINFO.LINES.TOTAL = STATINFO.LINES.TOTAL + 1;
/*****************************************************************/
/***  If the line starts with )IM, save the name of the        ***/
/***  imbedded member in LIST_INFO in an 'INCL' array element.  ***/
/*****************************************************************/
      IF SUBSTR(RECORD,1,3) = PAREN || 'IM' THEN
        DO;
          CALL GETNAME;
          USED_ELMTS = USED_ELMTS + 1;
          IF USED_ELMTS < LISTLEN THEN
            DO;
              LISTINFO(USED_ELMTS).TYPE = 'INCL';
              LISTINFO(USED_ELMTS).DATA = NAME;
            END;
          ELSE;
        END;
      ELSE;
/*****************************************************************/
/***  If the line starts with )DOT, save the name of the       ***/
/***  referenced table in LIST_INFO in a 'USER' array element.  ***/
/*****************************************************************/
      IF SUBSTR(RECORD,1,4) = PAREN || 'DOT' THEN
        DO;
          CALL GETNAME;
          USED_ELMTS = USED_ELMTS + 1;
          IF USED_ELMTS < LISTLEN THEN
            DO;
              LISTINFO(USED_ELMTS).TYPE = 'USER';
              LISTINFO(USED_ELMTS).DATA = 'TABLE: ' || NAME;
            END;
          ELSE;
        END;
      ELSE;
/*****************************************************************/
/***  If the line starts with )CM, increment the comment       ***/
/***  counter.  Otherwise, increment the non-comment counter.  ***/
/*****************************************************************/
      IF SUBSTR(RECORD,1,3) = PAREN || 'CM' THEN
        STATINFO.LINES.COMMENT = STATINFO.LINES.COMMENT + 1;
      ELSE
        STATINFO.LINES.NON_COMMENT = STATINFO.LINES.NON_COMMENT + 1;
```

Figure 35. Parser for ISPF skeletons (Part 5 of 8)

```
/*********************************************************************/
/***  If the line starts with )BLANK, increment the blank line    ***/
/***  counter.                                                    ***/
/*********************************************************************/
      IF SUBSTR(RECORD,1,6) = PAREN || 'BLANK' THEN
        STATINFO.LINES.BLANK = STATINFO.LINES.BLANK + 1;
      ELSE;
/*********************************************************************/
/***  If the line starts with ), increment the control            ***/
/***  statement counter.                                          ***/
/***                                                              ***/
/***  If the line does not start with ), increment the data       ***/
/***  line counter.                                               ***/
/***                                                              ***/
/***  If this is the first data line, then we have reached the end***/
/***  of the prolog (defined here as the comment lines before the ***/
/***  first data line).  Set the prolog count to the number of    ***/
/***  comments read so far.                                       ***/
/*********************************************************************/
      IF SUBSTR(RECORD,1,1) = PAREN THEN
        STATINFO.STMTS.CONTROL = STATINFO.STMTS.CONTROL + 1;
      ELSE
        DO;
          IF ¬NON_COM_READ THEN
            DO;
              STATINFO.LINES.PROLOG = STATINFO.LINES.COMMENT;
              NON_COM_READ = '1'B;
            END;
          ELSE;
        END;
/*********************************************************************/
/***  If this line starts with )DEFAULT, then the special         ***/
/***  character (the left parenthesis) for control cards might    ***/
/***  have changed.  Get the new character.                       ***/
/*********************************************************************/
      IF SUBSTR(RECORD,1,8) = PAREN || 'DEFAULT' THEN
        DO;
          I = VERIFY(SUBSTR(RECORD,9,72),' ') + 8;
          PAREN = SUBSTR(RECORD,I,1);
        END;
      ELSE;
/*********************************************************************/
/***  End of parse-a-line loop.  If there's another line, read it ***/
/***  and go back through the loop.                               ***/
/*********************************************************************/
      GET FILE(SSOURCE) EDIT(RECORD) (A(80));
   END;
   CLOSE FILE(SSOURCE);
/*********************************************************************/
/***  If there were no non-comment lines, then set the number of  ***/
/***  prolog lines to the number of comment lines.                ***/
/*********************************************************************/
   IF ¬NON_COM_READ THEN
     STATINFO.LINES.PROLOG = STATINFO.LINES.COMMENT;
   ELSE;
END PARSE;
```

Figure 36. Parser for ISPF skeletons (Part 6 of 8)

```
GETNAME: PROC;
/*********************************************************************/
/***                                                               ***/
/***  Routine:   GETNAME                                           ***/
/***                                                               ***/
/***  Purpose:   Returns the name specified on an )IM or )DOT      ***/
/***             statement.                                        ***/
/***                                                               ***/
/***  Inputs:    An 80-byte record in variable RECORD.             ***/
/***                                                               ***/
/***  Outputs:   The 8-byte name in variable NAME.                 ***/
/***                                                               ***/
/***  Logic:     1) Find the first blank after the )IM or )DOT.    ***/
/***             2) Find the next nonblank after that blank.      ***/
/***             3) Move that nonblank and the next 7 bytes into ***/
/***                variable NAME.                                 ***/
/***                                                               ***/
/*********************************************************************/
   I = INDEX(RECORD,' ');
   I = VERIFY(SUBSTR(RECORD,I,81-I),' ') + I - 1;
   NAME = SUBSTR(RECORD,I,8);
END GETNAME;

WRAPUP: PROC;
/*********************************************************************/
/***                                                               ***/
/***  Routine:   WRAPUP                                            ***/
/***                                                               ***/
/***  Purpose:   Saves the last of the parse information in the    ***/
/***             SCLM structures and outputs error messages to     ***/
/***             the listing file if the LIST_INFO array was not   ***/
/***             large enough to hold all of the information.      ***/
/***                                                               ***/
/***  Inputs:    None.                                             ***/
/***                                                               ***/
/***  Outputs:   More data in LIST_INFO and STAT_INFO.             ***/
/***                                                               ***/
/***  Logic:     1) Calculate summary information.                 ***/
/***             2) Write an 'END ' element to LIST_INFO.          ***/
/***             3) If there was not enough room in LIST_INFO,     ***/
/***                write out messages that describe the error     ***/
/***                and that indicate how to solve the problem.    ***/
/***                                                               ***/
/*********************************************************************/
   STATINFO.STMTS.TOTAL       = STATINFO.LINES.TOTAL;
   STATINFO.STMTS.COMMENT     = STATINFO.LINES.COMMENT;
   STATINFO.STMTS.NON_COMMENT = STATINFO.LINES.NON_COMMENT;
```

Figure 37. Parser for ISPF skeletons (Part 7 of 8)

```
    /**/
    /*   WRITE AN END ELEMENT TO LIST ARRAY                          */
    /**/
    USED_ELMTS = USED_ELMTS + 1;
    IF USED_ELMTS < LISTLEN THEN
      DO;
        LISTINFO(USED_ELMTS).TYPE = 'END ';
        LISTINFO(USED_ELMTS).DATA = ' ';
      END;
    ELSE
      DO;
        OPEN FILE(ERROR);
            /**/
        PUT FILE(ERROR) SKIP LIST(
                  'ERROR: INFORMATION RESULTING FROM PARSE DOES NOT ' ||
                  'FIT IN PARSE ARRAYS.');
            /**/
        PUT FILE(ERROR) SKIP LIST(
                  '       PARSE ARRAY ELEMENTS:', LISTLEN);
            /**/
        PUT FILE(ERROR) SKIP LIST(
                  '       ELEMENTS NEEDED:     ', USED_ELMTS);
            /**/
        PUT FILE(ERROR) SKIP(2) LIST(
                  'FIX:   1) INCREASE BUFSIZE VALUE IN FLMLANGL MACRO,');
            /**/
        PUT FILE(ERROR) SKIP LIST(
                  '              - OR - ');
            /**/
        PUT FILE(ERROR) SKIP LIST(
                  '       2) BREAK THIS SKELETON UP INTO SMALLER ' ||
                  'SKELETONS AND IMBED THEM ');
            /**/
        PUT FILE(ERROR) SKIP LIST(
                  '          IN A NEW "TOP LEVEL" SKELETON ');
            /**/
        PUT FILE(ERROR) SKIP(2) LIST(
                  'PARAMETER LIST: ' || PARMLISTX);
            /**/
        LISTINFO(LISTLEN).TYPE = 'END ';
        LISTINFO(LISTLEN).DATA = ' ';
            /**/
        CLOSE FILE(ERROR);
            /**/
        RETCODE = 4;
      END;
  END WRAPUP;
  END PSKELS;
```

Figure 38. Parser for ISPF skeletons (Part 8 of 8)

```
/*****************************************************************/
/***                                                           ***/
/***  LISTINFO Structure                                       ***/
/***                                                           ***/
/***  Maps the static portion of the account record.           ***/
/***                                                           ***/
/***  The number of elements declared for this array should not ***/
/***  be greater than the value specified on the BUFSIZE keyword ***/
/***  on the FLMLANGL macro.                                   ***/
/***                                                           ***/
/*****************************************************************/
   DCL 1 LISTINFO(50)      BASED(LIST_PTR),
         2 TYPE            CHAR(4),
         2 DATA            CHAR(224);
```

Figure 39. LISTINFO Module

```
/*****************************************************************/
/***                                                           ***/
/***  STATINFO Structure                                       ***/
/***                                                           ***/
/***  Maps the static portion of the account record.           ***/
/***                                                           ***/
/*****************************************************************/
   DCL 1 STATINFO          BASED(STAT_PTR),
         2 LINES,
           3 TOTAL         FIXED BIN(31),
           3 COMMENT       FIXED BIN(31),
           3 NON_COMMENT   FIXED BIN(31),
           3 BLANK         FIXED BIN(31),
           3 PROLOG        FIXED BIN(31),
         2 STMTS,
           3 TOTAL         FIXED BIN(31),
           3 COMMENT       FIXED BIN(31),
           3 CONTROL       FIXED BIN(31),
           3 ASSIGNMENT    FIXED BIN(31),
           3 NON_COMMENT   FIXED BIN(31);
```

Figure 40. STATINFO Module

Processing conditionally saved components

SCLM provides a feature to handle translators that, by design, have missing or static outputs. Static outputs help SCLM in its work-avoidance algorithms. Note, however, that SCLM relies on translator return codes to determine which outputs are static.

Example of processing conditionally saved components

Suppose a translator can determine if a developer changed only comments in the source code, and signals that by a return code of 2. The translator creates a listing output to match the current source. However, creating object code for the source is unnecessary because comment changes to source do not alter object code. In this case, the object code is a static output because it did not change. Specifying a NOSAVRC=2 on the FLMALLOC macro corresponding to the object output instructs SCLM not to copy object modules back to the hierarchy when the translator returns a 2. SCLM copies the generated listing back to the hierarchy when the translator returns a 2, if the object modules already exist in the hierarchy.

Components that depend on the object do not need to be rebuilt when only the listing is regenerated. If you specify DEPPRCS=N on the FLMLANGL macro, SCLM rebuilds components dependent on a member only if all its outputs were saved.

```
          FLMLANGL    LANG=XYZ,VERSION=V1,DEPPRCS=N
* BUILD TRANSLATOR(S)
*
          FLMTRNSL   CALLNAM='TRANSLATOR XYZ',                      C
               FUNCTN=BUILD,                                        C
               COMPILE=XYZ,                                         C
               GOODRC=4
*
*         (* SYSIN *)
          FLMALLOC  IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,         C
             RECNUM=1000,DDNAME=SYSIN
*         (* SYSPRINT *)
          FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=VBA,LRECL=133,       C
             RECNUM=30000,PRINT=Y,DDNAME=SYSPRINT,DFLTTYP=LISTING
*         (* SYSLIN *)
          FLMALLOC  IOTYPE=O,KEYREF=OBJ,RECFM=FB,LRECL=80,          C
             RECNUM=5000,DDNAME=SYSLIN,DFLTTYP=OBJ,NOSAVRC=2
```

Figure 41. Sample Language Definition for Conditionally Saved Components

Setting up the project definition

To access this feature, use the FLMALLOC, FLMLANGL, and FLMTRNSL macros:

1. Identify the static outputs and their corresponding FLMALLOCs in the language definition.
2. For each static output:
   - List the translator return code that indicates that the output is not to be saved
   - Specify that return code as the NOSAVRC parameter of the FLMALLOC macro for that output.

   The NOSAVRC must have a nonzero positive value. It is only valid for IOTYPEs O and P.
3. Make sure that the GOODRC on the FLMTRNSL macro corresponding to that translator is greater than or equal to the highest NOSAVRC parameter you specified.
4. Determine whether you want SCLM to rebuild components that depend on a given member only if all its outputs (including the static outputs) were saved. If that is the case, specify DEPPRCS=N on the FLMLANGL macro. If you specify DEPPRCS=Y (or let it default to Y), SCLM rebuilds components that depend on that member whenever the build translator returns a good return code. In the preceding example, DEPPRCS=Y causes SCLM to rebuild components that depend on the given member even when only the listing has changed.

Likewise, the translator can directly store output in an external data set not under SCLM control. For example, the Ada translator controls output stored in Ada sublibraries. Under such circumstances, the build function requires a signal from the translator to detect whether some of the external outputs were saved to external data sets. SCLM uses NOSVEXT on the FLMTRNSL macro in the same fashion as the parameter NOSAVRC on the FLMALLOC macro to detect whether external outputs were saved.

Specifying the locations of included members

SCLM tracks two pieces of information for each include member that is found by a parser. The first piece of information is the member name of the include; the second is the include set that contains the included member. If no include set is returned by the parser for a member, SCLM assigns that member to the default include set. The name of the default include set is all blanks.

SCLM does not track an include member if it meets all of the following conditions:

- The language definition for the member specifies CHKSYSLB=PARSE. This is the default.
- An accounting record for the include is not found by searching the hierarchy for each type specified on the FLMINCLS for the include set.
- The include is found in one of the data sets specified on an FLMSYSLB macro for the include set.

Includes that meet these conditions are removed from the list of includes stored in the accounting record of the member. Because the include is not being tracked, build and promote do not detect if the include is removed from the FLMSYSLB data sets or added to the project database.

Build ignores an include if it meets all of the following conditions:

- The language definition for the member specifies CHKSYSLB=BUILD.
- An accounting record for the include is not found by searching the hierarchy for each type specified on the FLMINCLS for the include set.
- The include is found in one of the data sets specified on an FLMSYSLB macro for the include set.

Includes that meet these conditions are removed from the list of includes stored in the build map record of the member. Because the include is not being tracked, build and promote will not detect if the include has changed since the last build.

The include information is used by build and promote to determine whether the member is up-to-date. When you build, the includes for an up-to-date member have the same type, date, time, and version as the last time that member was built. When you promote, the includes for an up-to-date member have the same date, time, and version as the last time that member was built. Promote does not search the types listed on FLMINCLS macros for includes. It relies instead on the information in the build map to determine

the type name of the included member. If a member is not up-to-date, build attempts to rebuild the member and promote does not allow the member to be promoted to the next group in the hierarchy.

An include set is used to associate an included member name with the type or types in the project that are searched to find a member with that name. The FLMINCLS macro is used to associate an include set with one or more types in the project definition. Types are searched in the order listed on the FLMINCLS macro. Each type is searched from the current group to the top of the hierarchy before the next type in the list is searched.

The number of include sets used by a language is usually related to the number of include ddnames supported by the build translators for that language, where the includes are located in project data sets. If the build translator only supports one include ddname, a single include set is sufficient for that language. On the other hand, if there are multiple build translators, each supporting an include ddname and the includes are separated into different types for each build translator, multiple include sets would be needed.

If multiple include sets are needed, parsers must return the appropriate include set for each include.

Example

This example shows how pieces of a project might look if it were set up to use multiple include sets.

The following list shows the different types of includes in the project and the location of each include type in the project data sets.

Include Type
Project Types and SYSLIB Data sets to Search

Constants
CONSTANT

Messages
INCLENGL, INCLUDE, PRODX.MSGLIB (syslib data set)

SQL Declarations
DCLGEN, source member's type, source member's extended type

All other includes
INCLUDE, source member's type, source member's extended type, SYS1.SEDCHDRS (syslib data set)

Figure 42 on page 96 shows how the include section of a source member might be coded:

```
#include <stdio>               /* C standard i/o            */
EXEC SQL INCLUDE SQLDEF1;      /* SQL definitions           */
#include "DD:MESSAGE(prog1)"   /* prog1 specific messages   */
#include "DD:CONSTANT(common)" /* common constants          */
#include "DD:CONSTANT(prog1)"  /* prog1 specific constants  */
```

Figure 42. Source member with includes in different include sets

The parser must return the following values:

Member
include set

STDIO

SQLDEF1
SQL

PROG1
MESSAGE

COMMON
CONSTANT

PROG1
CONSTANT

You could then use the language definition in Figure 43 on page 97 for this member.

```
************************************************************************
*           C370 W/DB2  LANGUAGE DEFINITION FOR PROJECT X              *
*                                                                      *
************************************************************************
*
CDB2      FLMSYSLB    SYS1.SEDCHDRS
*
          FLMLANGL    LANG=CDB2,VERSION=V1,ALCSYSLB=Y
*
* CONSTANT INCLUDES
*
CONSTANT  FLMINCLS TYPES=(CONSTANT)
*
* MESSAGE INCLUDES
*
MESSAGE   FLMINCLS TYPES=(INCLENGL,INCLUDE)
*
* SQL INCLUDES
*
SQL       FLMINCLS TYPES=(DCLGEN,@@FLMTYP,@@FLMETP)
*
* ALL OTHER INCLUDES - DEFAULT INCLUDE SET
*
          FLMINCLS TYPES=(INCLUDE,@@FLMTYP,@@FLMETP)
*
* PARSER TRANSLATOR
*
          FLMTRNSL   CALLNAM='C370 REXX PARSER',                      C
               FUNCTN=PARSE,                                          C
               COMPILE=MYCPARSE,                                      C
               DSNAME=SOMEUSR.PARSER.LOAD,                            C
               CALLMETH=TSOLNK,                                       C
               PORDER=1,                                              C
               OPTIONS=(LISTSIZE=@@FLMSIZ,                            C
               LISTINFO=@@FLMLIS,                                     C
               STATINFO=@@FLMSTP)
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
* BUILD DB2 PREPROCESSOR TRANSLATOR
*
*         --DB2 PREPROCESSOR INTERFACE--
          FLMTRNSL   CALLNAM='DB2 C PREP',                            C
               FUNCTN=BUILD,                                          C
               COMPILE=DSNHPC,                                        C
               VERSION=D220,                                          C
               GOODRC=4,                                              C
               PORDER=3,                                              C
               OPTIONS=(HOST(C),APOST)
```

Figure 43. Language definition to support multiple include sets (Part 1 of 3)

```
*  1       -- N/A --
           FLMALLOC  IOTYPE=N
*  2       -- N/A --
           FLMALLOC  IOTYPE=N
*  3       -- N/A --
           FLMALLOC  IOTYPE=N
*  4       -- SYSLIB --
           FLMALLOC  IOTYPE=I,INCLS=SQL
*  5       -- SYSIN --
           FLMALLOC  IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,          C
               RECNUM=5000
*  6       -- SYSPRINT --
           FLMALLOC  IOTYPE=W,RECFM=FBA,LRECL=133,                    C
               RECNUM=35000,PRINT=Y
*  7       -- N/A --
           FLMALLOC  IOTYPE=N
*  8       -- SYSUT1 --
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=800,RECNUM=9000
*  9       -- SYSUT2 --
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=800,RECNUM=9000
* 10       -- SYSUT3 --
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=800,RECNUM=9000
* 11       -- N/A --
           FLMALLOC  IOTYPE=N
* 12       -- SYSTERM --
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
* 13       -- N/A --
           FLMALLOC  IOTYPE=N
* 14       -- SYSCIN --
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,                      C
               RECNUM=9000,DDNAME=DB2TRANS
* 15       -- N/A --
           FLMALLOC  IOTYPE=N
* 16       -- DBRMLIB--
           FLMALLOC  IOTYPE=P,DDNAME=DBRMLIB,MEMBER=@@FLMONM,         C
               DFLTTYP=DBRM,KEYREF=OUT1,                              C
               RECFM=FB,LRECL=80,RECNUM=5000,DIRBLKS=1
*
* BUILD C370 TRANSLATOR
*
         FLMTRNSL   CALLNAM='C 370',                                  C
               FUNCTN=BUILD,                                          C
               COMPILE=EDCCOMP,                                       C
               DSNAME=SYS1.SEDCCOMP,                                  C
               VERSION=C210,                                          C
               GOODRC=0,                                              C
               PORDER=3,                                              C
               OPTIONS=(XREF,LANGLVL(SAAL2),SOURCE,OPT,TEST(ALL),     C
               MARGINS(1,72),NOGONUM,NOTERMINAL,FLAG(I),SHOWINC)
```

Figure 44. Language definition to support multiple include sets (Part 2 of 3)

```
*
* 1         (* SYSIN *)
            FLMALLOC  IOTYPE=U,DDNAME=DB2TRANS
*
* 2         (* SYSLIN *)
            FLMALLOC  IOTYPE=O,KEYREF=OBJ,RECFM=FB,LRECL=80,            C
                RECNUM=5000,DFLTTYP=OBJ
*
* 3         (* SYSMSGS *)
            FLMALLOC  IOTYPE=A
              FLMCPYLB SYS1.SEDCMSGS(EDCMSGE)
*
* 4         (* SYSLIB *)
            FLMALLOC  IOTYPE=A
              FLMCPYLB SYS1.SEDCHDRS
*
* 5         (* USERLIB *)
            FLMALLOC  IOTYPE=I
*
* 6         (* SYSPRINT *)
            FLMALLOC  IOTYPE=A
             FLMCPYLB NULLFILE
*
* 7         (* SYSCPRT  *)
            FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=VBA,LRECL=137,         C
                RECNUM=20000,PRINT=Y,DFLTTYP=LIST
*
* 8         (* SYSPUNCH *)
            FLMALLOC  IOTYPE=A
             FLMCPYLB NULLFILE
*
* 9         (* SYSUT1 *)
            FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=2000
*
* 10        (* SYSUT4 *)
            FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=2000
*
* 11        (* SYSUT5 *)
            FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=2000*
* 12        (* SYSUT6 *)
            FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=2000
*
* 13        (* SYSUT7 *)
            FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=2000
*
* 14        (* SYSUT8 *)
            FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=2000
*
* 15        (* SYSUT9 *)
            FLMALLOC  IOTYPE=W,RECFM=VB,LRECL=137,RECNUM=2000
*
* 16        (* SYSUT10 *)
            FLMALLOC  IOTYPE=A
             FLMCPYLB NULLFILE
*
*           (* CONSTANT *)
            FLMALLOC  IOTYPE=I,DDNAME=CONSTANT,INCLS=CONSTANT
*
*           (* MESSAGE *)
            FLMALLOC  IOTYPE=I,DDNAME=MESSAGE,INCLS=MESSAGE
```

Figure 45. Language definition to support multiple include sets (Part 3 of 3)

Dynamic include tracking

The SCLM build processor attempts to resolve all include references to source members before it invokes any translator. However, for some translators, the include for a source member cannot be resolved until *after* the translator invocation. Such includes are referred to as *dynamic includes*. SCLM can track dynamic includes if the dynamic includes for a member can be altered only by modification of the member or one of the included members.

To support dynamic includes, SCLM invokes an additional build translator step (FLMTRNSL macro) following the translator that produces the output data set containing a list of dynamic includes. This additional translator should parse the output data set for dynamic includes and store them in memory supplied by the build processor. You pass the address of this memory to the translator by specifying the SCLM variable @@FLMINC in the translator options (OPTION parameter on FLMTRNSL macro). @@FLMINC is a pointer to a set of includes relating to a specified member. The value of @@FLMINC is a string of decimal characters that you must convert to a fullword binary value before using it as an address. The following record layout is used to store the dynamic includes:

```
COUNT    : 4 bytes
TYPE1    : 8 bytes
MEMBER1  : 8 bytes
TYPE2    : 8 bytes
MEMBER2  : 8 bytes
    .
    .
    .
TYPE#    : 8 bytes
MEMBER#  : 8 bytes
```

Figure 46. Record Layout Used to Store Dynamic Includes

You must specify the number of dynamic includes in the first 4 bytes as a fullword binary integer, followed by the list of dynamic include member and type names. The amount of memory that the SCLM build processor supplies limits the number of dynamic includes to 1000.

When using dynamic includes, consider the following points:

- Be sure to remove any duplicate include references before placing them in the structure pointed to by @@FLMINC.
- Processors need the ability to handle 31-bit addresses as specified by the @@FLMINC parameter.
- Do not return any include references that are actually to external (non-SCLM) libraries. The build step will receive an error (FLM01001) for any members not in the specified SCLM library.
- Deletion of members referenced through a dynamic include causes a build verification error (FLM43001). The build process does not proceed, even when using unconditional mode. If a referenced member is to be deleted, a build using the updated source should be performed before the deletion so that the build map can be updated to remove the reference.
- Dynamic include references to members that are outputs of other members do not cause a relationship to the member that created it, even when using extended mode. Builds and promotes for these must use a high-level architecture definition whose scope includes both source members.

Input list translators

SCLM provides support for Build translators that operate on more than one source member in a single invocation. This type of translator is known as an input list translator. SCLM users can use existing translators that support this feature or write new user-defined translators to take advantage of the feature. The IBM Ada/370 Compiler is the only SCLM-supported translator that can use input lists.

The SCLM Input List feature can increase the performance of an SCLM Build. Instead of SCLM calling a translator once for each member to be built, SCLM calls the translator passing a list of members to be built. SCLM attempts to place as many members as possible on each input list, thereby limiting the number of translator invocations. The project manager specifies the maximum number of members passed to a translator on an invocation in the language definition that includes the translator. This feature is most useful when using translators that have a high startup overhead to run. Fewer invocations mean increased speed for the SCLM Build process.

An input list translator receives a file that contains a list of data sets that a Build action is performed against. It returns a file that contains a return code for each data set in the input list and, optionally, a set of unique outputs for each data set in the input list.

Two translators, FLMTPRE and FLMTPST, serve as the interfaces between SCLM and the input list translator.

- The FLMTPRE translator generates a list of data sets that an input list translator can use as input.
- The FLMTPST translator passes the return code information that an input list translator provides for every data set on the input list back to SCLM.

For more information, refer to "FLMTPRE" on page 623 and "FLMTPST" on page 625.

Note: The input list feature of the Build function is designed to work with direct translations of source members only (source members referenced with an INCLD statement). Using the input list feature with source members controlled by CC or Generic architecture definitions will produce undefined results (source members referenced with a SINC statement).

Configuring the input list translators

Use the following macros to configure the input list translators to fit your needs:

- FLMLANGL

  Set the following parameters:

  - INPLIST=Y
  - MBRLMT to the maximum number of members that can be included in the same invocation of the translator.
  - SLOCLMT to the maximum number of source lines to be processed on a single invocation of the translator.
- FLMTRNSL

  Set the following parameters:

  - INPLIST=Y
  - MBRRC to the maximum good return code for each member in the input list. MBRRC defaults to 0 and is optional.
- FLMALLOC

  Set the following parameters:

  - MALLOC to designate which outputs of a translator have multiple unique instances.
  - IOTYPE to O or P.

    SCLM only saves outputs with IOTYPE=O in the hierarchy. For IOTYPE=O, you must also specify the FLMCPYLB macro and the data set name on FLMCPYLB must contain the @@FLMMBR variable somewhere in the variable string to enable SCLM to find the member-specific outputs. When IOTYPE=O is specified, the input list translator is expected to allocate the output data sets necessary for each member.

    Temporary data sets allocated with IOTYPE=P can be used as work data sets for the translators, but they cannot be stored in the hierarchy.
  - ALLCDEL to designate which output data sets were defined by the translator and should be deleted by SCLM.

Defining a new language to SCLM

This section describes the control structures used to manage SCLM processes and illustrates how to define a new language to SCLM. An example is included to show the statements needed to define the control structures and SCLM macros. The example refers to a fictitious compiler, the Finnoga 4, to show

how to gather the information you need and how to specify that information to SCLM in the form of language definition macros.

Using DDnames and DDname substitution lists

Many translators support a ddname substitution list; this contains ddnames, which are passed as a parameter to the translator. In Figure 50 on page 116, the ddname in position 5 is the ddname from which the compiler reads the source to be compiled. The ddname occupying that position in the ddname substitution list is usually called SYSIN. You can override the default ddname by placing another ddname in position 5 of the ddname substitution list. The compiler then reads from the other ddname. Table 14 on page 102 lists the various ddnames used by the Finnoga 4 compiler described in this example. The position number indicates the position of the ddname in a ddname substitution list. In addition, Table 14 on page 102 gives a brief description of the data sets allocated to the ddnames.

Note that some position numbers do not have a ddname associated with them.

SCLM allows a maximum of 512 characters for the ddname substitution list. Because every FLMALLOC for a given translator causes an 8-character ddname to be put into the ddname substitution list, when the PORDER > 1, a given translator may have a maximum of 64 FLMALLOCs.

Ddname substitution lists are usually documented in the programming guide for specific compilers and linkage editors. Note that it is rare for two different compilers to have the same ddname substitution list mappings.

Compilers are not required to support a ddname substitution list in order to be defined to SCLM. However, ddname substitution list support makes it easy to link or string two different compilers or preprocessors together. In “Defining a preprocessor to SCLM” on page 113, you will see how a ddname substitution list is used to pass the outputs of a preprocessor to a compiler.

Compiler options

Assume that there are four Finnoga 4 compiler options that you can use:

- SOURCE or NOSOURCE
- MACRO or NOMACRO
- OPTIMIZE or NOOPTIMIZE
- OBJ().

It is not critical at this point to understand what these options mean to the compiler, just which options are to be used for each compile. You should always specify SOURCE, NOMACRO, and OBJ(), but you must specify the OPTIMIZE parameter on a module-by-module basis.

Table 14. DDname Substitution List Example

Position Number	DDname	Description of data set(s) allocated
1	SYSLIN	A partitioned data set into which the Finnoga 4 compiler writes the object module. The OBJ keyword in the compiler's option string specifies the member name to use.
2	<none>	<none>
3	<none>	<none>
4	SYSLIB	One or more partitioned data sets through which the Finnoga 4 compiler searches for INCLUDE members.
5	SYSIN	A sequential data set that contains Finnoga 4 source to be compiled.

Table 14. DDname Substitution List Example (continued)

Position Number	DDname	Description of data set(s) allocated
6	SYSPRINT	A sequential listings data set. The Finnoga 4 compiler writes out a copy of the source that was compiled along with any error, warning, and informational messages.
7	<none>	<none>
8	FINLIB	A data set that contains information needed by the Finnoga 4 compiler. This data set comes with the compiler.
9	<none>	<none>
10	SYSUT1	A sequential work data set.
11	SYSUT2	A sequential work data set.

Defining a new language: step-by-step

The following list briefly describes the process required to write a new SCLM language definition:

1. Define the language name to SCLM.
2. Define include-sets for the language to identify the locations of included members.
3. List the various programs (parsers, compilers, and so on) used to parse and build your source.
4. For each program (or translator), look up the ddname substitution list (usually in the Programmer's Guide for the compiler), or list the ddnames used by the program.
5. For each program or translator, write an FLMTRNSL macro followed by FLMALLOC macros (one for each ddname to be allocated for the translator). Use the information in the program documentation to determine which IOTYPE value to specify as well as which other FLMALLOC keywords are appropriate.
6. Write a sample architecture definition and send it to your users. Describe to your users how to convert a JCL file of linkage editor control statements into architecture definitions.
7. Place the application under SCLM control.

This section is an illustration of the process for defining a language to SCLM. As you progress through the definition, you will code SCLM macros with the information SCLM needs to control Finnoga 4 modules. You will place this code into a member of the PROJDEFS.SOURCE data set called @FINNOGA. Language definitions such as @FINNOGA are usually referenced in the code for a project definition by means of the COPY statement.

Step 1.
Define the language.

The first step is to tell SCLM that you are defining a new language. To do so, code the following FLMLANGL macro:

```
FLMLANGL LANG=FINNOGA,VERSION=FINN4
```

In this example, values are specified for two parameters. The default values are used for the other parameters.

Parameter
Description

LANG=
Specifies the language name a user must enter on the SPROF panel or on the Migrate Utility panel to request that this language definition be used to drive build and parse operations of the Finnoga 4 modules.

VERSION=
Identifies the specific release of the current Finnoga 4 compiler. If you install a new release or version of the Finnoga 4 compiler, you can set this parameter to a different value so that SCLM can mark all Finnoga 4 modules needing to be rebuilt. You must then re-assemble and link your project definition.

Step 2.
Define include sets for the language to identify the locations of included members.

After the language is defined, you can specify where SCLM finds included members for the Finnoga 4 language. In the following example, the FLMINCLS macro is used to list the types that are searched for includes:

```
          FLMINCLS TYPES=(INCLUDE,@@FLMTYP)
```

In this example, the TYPES parameter of the FLMINCLS macro is used to tell SCLM where to look for includes. Because no name is specified, this definition applies to the default include set.

Parameter
Description

FLMINCLS name
Specifies the name of the include set that uses this definition. If no name is specified (as in this example), the definition is associated with the default include set. An include set defines a search path for all includes associated with that include set. Multiple include set s can be specified in a language definition if the parser and compiler support distinguishing one kind of include from another. For the parser, this means that the syntax of the language must support determining which include set an include belongs to. For the compiler, this means that a separate ddname must be used for each different include set (kind of include).

Two include sets are useful when the standard language includes are kept in one Type and the “EXEC SQL” includes are kept in another Type. A parser can be written to determine which include set each include is in. The language definition then associates a ddname from the build translators with the appropriate include set name.

TYPES=
Specifies the name(s) of the types which are searched to find includes. In this case, the “INCLUDE” type is searched first. The @@FLMTYP SCLM variable indicates that the type of the member that is processed by the Finnoga 4 compiler is to be searched next. For example, if 'EXAMPLE.USERX.SOURCE(PROGA)' is going to be compiled, SCLM looks for includes first in the data sets associated with the INCLUDE type and then the SOURCE type.

Step 3.
Specify the programs that process the modules.

Next, identify the programs that are used to parse and build the Finnoga 4 modules. There are usually two such programs: a parser and the compiler. For each of these programs, code an FLMTRNSL macro and the appropriate FLMALLOC macros and FLMCPYLB macros.

Assume that you have written your own parser and that it is in the data set SCLM.PROJDEFS.LOAD(FINPARSE). The parser requires an option string @@FLMSIZ,@@FLMSTP,@@FLMLIS, and reads the source from ddname SOURCE.

Add this to your language definition:

```
FLMTRNSL   CALLNAM='FINNOGA PARSER',                          C
      FUNCTN=PARSE,                                           C
      COMPILE=FINPARSE,                                       C
      DSNAME=SCLM.PROJDEFS.LOAD,                              C
      PORDER=1,                                               C
      OPTIONS=(@@FLMSIZ,@@FLMSTP,@@FLMLIS)
```

The parameters included in this example are described as follows:

Parameter
Description

CALLNAM=
A character string that appears in messages during the specified FUNCTN (in this case PARSE). This value will assist in recognizing which translator was executing during the specified FUNCTN.

FUNCTN=
The value PARSE tells SCLM that this program is to be invoked whenever you parse a module with language FINNOGA.

COMPILE=
Member name of the load module for the Finnoga 4 parser. Note that the keyword "COMPILE" actually identifies the load module name of a translator (which may or may not be a compiler).

DSNAME=
Names the partitioned data set that contains the Finnoga 4 parser load module. DSNAME is required when the data set containing the desired module is not in the system concatenation. DSNAME is similar to a STEPLIB.

When more than one data set is to be searched, the TASKLIB parameter can be used in conjunction with, or as a replacement for, the DSNAME parameter.

PORDER=
The value 1 tells SCLM that this program expects an options string but not a ddname substitution list.

OPTIONS=
Specifies the options string to be passed to the parser. Strings that start with @@FLM are SCLM variables, and they are replaced by their current values before the string is passed to the parser.

Since the parser reads its source from a ddname, you must tell SCLM how to allocate that ddname. To do this, use an FLMALLOC macro and an FLMCPYLB macro.

```
FLMALLOC  IOTYPE=A,DDNAME=SOURCE
FLMCPYLB  @@FLMDSN(@@FLMMBR)
```

A description of the parameters follows:

Parameter
Description

IOTYPE=A
Tells SCLM to allocate a ddname to one, or a concatenation of, specific data set(s). Each of those data sets are subsequently identified by using an FLMCPYLB macro.

DDNAME=
Identifies the ddname to be allocated.

@@FLMDSN(@@FLMMBR)
Identifies the member to be parsed. When the two SCLM variables are resolved, you get the member of the data set in which you are interested.

Now you can tell SCLM how to invoke the Finnoga 4 compiler. To do so, use an FLMTRNSL macro followed by one or more FLMALLOC and FLMCPYLB macros.

```
FLMTRNSL   CALLNAM='FINNOGA 4',                                C
      FUNCTN=BUILD,                                            C
      COMPILE=FNGAA40,                                         C
      PORDER=3,                                                C
      GOODRC=0,                                                C
      OPTIONS='SOURCE,NOMACRO,OBJ(@@FLMMBR),',                 C
      PARMKWD=PARM1
```

You can specify only a few of the parameters and let SCLM supply default values for the others:

Parameter
Description

CALLNAM=
Names the compiler. This name appears in build messages.

FUNCTN=
Tells SCLM that this program gets invoked whenever you want to build a member with language FINNOGA.

COMPILE=
Identifies the load module name for the Finnoga 4 compiler.

DSNAME=
If you do not specify a DSNAME value, SCLM assumes that the load module can be found in the system concatenation.

PORDER=
The value 3 tells SCLM to pass an options string and a ddname substitution list to the Finnoga 4 compiler.

GOODRC=
The value 0 indicates that SCLM is to consider this build unsuccessful if the compiler completes with any return code greater than 0.

OPTIONS=
Specifies the options string to be passed to the compiler. At compiler run time, the SCLM variable @@FLMMBR is resolved to the member name being built.

PARMKWD=
The value PARM1 specifies the concatenation of the contents of the PARM1 parameters in the architecture definition to the preceding options string. Use the PARM1 parameter to specify the OPTIMIZE/NOOPTIMIZE option for each member. An example of this is provided later in this section.

As discussed previously, the Finnoga 4 compiler uses 7 ddnames and also supports a ddname substitution list. The preceding parser invocation definition showed how to define a translator (the parser) that does not use a ddname substitution list. The following SCLM FLMALLOC macros are used by SCLM to construct the ddname substitution list shown in Table 14 on page 102.

When you use a ddname substitution list, you must define the ddnames in the order in which they are expected to appear in the ddname substitution list by the translator. The first ddname defined is placed by SCLM into position 1 in the ddname substitution list. The second ddname specified is placed into position 2 in the ddname substitution list, and so on.

Note that you do not have to specify any ddnames in the following example macros. SCLM will create temporary unique ddnames and place them into the ddname substitution list positions. Because of the way ddname substitution lists work, the compiler uses those temporary ddnames instead of the standard documented ddnames (like SYSIN).

The first ddname in the Finnoga 4's ddname substitution list is SYSLIN. It is allocated to a partitioned data set into which the compiler places the object module.

```
FLMALLOC IOTYPE=P,KEYREF=OBJ,DFLTTYP=OBJ,RECFM=FB,LRECL=80,   C
    RECNUM=5000
```

The parameters specified in this macro are described as follows:

Parameter
Description

IOTYPE=P
The compiler is written in such a way that a partitioned data set must be allocated to this ddname. The compiler will write to a member of this partitioned data set. SCLM creates a temporary PDS and allocates it to a temporary ddname (since no DDNAME keyword was specified).

This example illustrates two points. It shows how to define a temporary PDS for output from a translator and emphasizes that each compiler (or parser) that you define to SCLM may be slightly different from any other translator you have defined to SCLM.

Always refer to the translator documentation when defining a translator to SCLM.

KEYREF=OBJ
To save what is written to this ddname and keep it under SCLM control, SCLM must be able to determine the member name and the SCLM-controlled data set name in which it is to save this output module. If SCLM is building an architecture definition, it determines the project, group, type and member as follows:

- The high-level qualifier is the project identifier that was previously specified.
- The group is the level at which the build is taking place. The group name is the second qualifier.
- SCLM looks at the architecture definition being built and retrieves the member and type from the architecture statement associated with the keyword OBJ. The type name is the third qualifier.

DFLTTYP=OBJ
To save what is written to this ddname and keep it under SCLM control, SCLM must be able to determine the member name and the SCLM-controlled data set name in which it is to save this output module. If SCLM is building a source member, it determines the project, group, type and member as follows:

- The high-level qualifier is the project identifier that was previously specified.
- The group is the level at which the build is taking place.
- The type is the value of the DFLTTYP= keyword.
- The member name defaults to the name of the member being built.

If SCLM is building an architecture definition (and not a source member directly) then the DFLTTYP= value is ignored. Instead, SCLM uses the type associated with the KEYREF= value.

RECFM=FB
Specifies the record format of the temporary data set that SCLM creates. In this example, the record format is fixed block.

LRECL=80
Specifies the record length, in characters, of the temporary data set that SCLM creates.

RECNUM=5000
Tells SCLM to allocate enough space in this data set to hold 5000 records (records that are fixed block and 80 characters in length).

Positions 2 and 3 in the ddname substitution list are not used. Create two FLMALLOC macros with IOTYPE=N to tell SCLM to fill those name fields with hex zeros and to continue to the next ddname.

```
FLMALLOC IOTYPE=N
*
FLMALLOC IOTYPE=N
```

The ddname in position 4 of the ddname substitution list must be allocated to one or more partitioned data sets. This ddname is used by the Finnoga 4 compiler to find included members. The FLMINCLS macro described earlier needs to be referenced here to ensure that the compiler is picking up includes from the correct data sets. Since IOTYPE=I allocations default to the default include set shown earlier, this is automatically done. If another name was used on the FLMINCLS macro, that name needs to be referenced here using the INCLS parameter. IOTYPE=I allocates a ddname with a concatenation of all the PDS's in the hierarchy starting with the group specified for the BUILD and ending with the top, or production level, group. First the hierarchy for the INCLUDE type is allocated, followed by the type of the first SINCed member from the architecture definition, or, if no architecture definition is used, the type of the member being built.

```
FLMALLOC IOTYPE=I,KEYREF=SINC
```

The parameters used with this macro are as follows:

Parameter
Description

IOTYPE=I
Allocate this ddname to a concatenation of SCLM-controlled data sets. The types used in the concatenation are determined by the FLMINCLS macro referenced by the INCLS= parameter on the FLMALLOC macro. In this case, there is no INCLS= parameter so the default FLMINCLS (or include set) is used.

A hierarchy of data sets is concatenated for each type specified for the referenced FLMINCLS macro. The hierarchy begins at the group where the build is taking place and extends to the top of the project's hierarchy. In this case, the concatenation first contains all of the data sets for the INCLUDES type followed by the data sets for the value substituted into the @@FLMTYP variable. See the KEYREF= parameter to determine the value which is substituted into the @@FLMTYP and @@FLMETP variables.

KEYREF=SINC
If you are building an architecture definition, refer to the first SINC statement in that architecture definition for the type that is substituted into the @@FLMTYP macro. The value for @@FLMETP comes from the EXTEND= parameter of the FLMTYPE macro for that type. If you are not building an architecture definition, the type is the type of the member being built.

The next ddname in the ddname substitution list is allocated to the source to be compiled

```
FLMALLOC IOTYPE=S,KEYREF=SINC
```

The parameters used in the example are as follows:

Parameter
Description

IOTYPE=S
Tells SCLM to allocate a temporary sequential data set.

KEYREF=SINC
If you are building a source module directly, SCLM copies that member to this temporary data set. If you are building a CC architecture definition, SCLM copies the members listed on the SINC statement to this data set.

Next, define the SYSPRINT ddname to SCLM.

```
FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=VBA,LRECL=125,          C
     RECNUM=5000,PRINT=Y,DFLTTYP=FINLIST
```

This definition contains the following parameters:

Parameter
Description

IOTYPE=O
Specifies that the compiler writes to this ddname using a sequential data set. SCLM creates a temporary sequential data set and allocates it to a temporary ddname (since this is part of a ddname substitution list).

KEYREF=LIST
Refers SCLM to the LIST record in the architecture definition being built. That record contains the member name and type into which the listing is saved after a successful build. (SCLM copies the data from the temporary data sets into members of the PDS's controlled by SCLM after a successful build.)

DFLTTYP=FINLIST
Specifies the data set type into which this listing is written whenever a Finnoga 4 module is built directly or when using INCLD in an architecture definition.

PRINT=Y
Specifies that this is a listing that should be copied to the Build List data set after the build process completes.

Although the next position in the ddname substitution list is not used, you still need to tell SCLM what to put there. Create another FLMALLOC with IOTYPE=N:

```
         FLMALLOC IOTYPE=N
```

Next, specify the FINLIB data set allocation to SCLM. Specifically, indicate that the Finnoga 4 library resides in a data set named SYS1.FINNOGA.LIB:

```
         FLMALLOC IOTYPE=A
         FLMCPYLB SYS1.FINNOGA.LIB
```

Finally, note that position 9 in the ddname substitution list, like position 7, is not used:

```
         FLMALLOC IOTYPE=N
```

The last two ddnames in the ddname substitution list for the Finnoga 4 compiler are temporary work data sets. Use IOTYPE=W for temporary work data sets, such as SYSUT1, SYSUT2, and so on. In addition, specify the record format and length of the two files, as shown in the following example:

```
FLMALLOC IOTYPE=W,LRECL=4000,RECFM=F,RECNUM=4000
*
FLMALLOC IOTYPE=W,LRECL=4000,RECFM=F,RECNUM=4000
```

When you have completed all these steps you will have a language definition similar to the following one. (Figure 47 on page 110 contains comments to explain the flow of operations.) When you are ready to reassemble your project definition, add a COPY statement in your main project definition file to include these macros.

```
*********************************************************
*  FINNOGA 4 LANGUAGE DEFINITION
*********************************************************
*
         FLMLANGL LANG=FINNOGA,VERSION=FINN4
*
*********************************************************
*  TYPES TO SEARCH FOR INCLUDES
*********************************************************
*
         FLMINCLS TYPES=(INCLUDE,@@FLMTYP)
*
*********************************************************
*  PARSE TRANSLATOR DEFINITION
*********************************************************
*
         FLMTRNSL   CALLNAM='FINNOGA PARSER',                          C
               FUNCTN=PARSE,                                           C
               COMPILE=FINPARSE,                                       C
               DSNAME=SCLM.PROJDEFS.LOAD,                              C
               PORDER=1,                                               C
               OPTIONS=(@@FLMSIZ,@@FLMSTP,@@FLMLIS)
*
*   -- SOURCE --
*
            FLMALLOC  IOTYPE=A,DDNAME=SOURCE
             FLMCPYLB  @@FLMDSN(@@FLMMBR)
*********************************************************
*  BUILD TRANSLATOR DEFINITION
*********************************************************
*
         FLMTRNSL   CALLNAM='FINNOGA 4',                               C
               FUNCTN=BUILD,                                           C
               COMPILE=FNGAA40,                                        C
               GOODRC=0,                                               C
               PORDER=3,                                               C
               OPTIONS='SOURCE,NOMACRO,OBJ(@FLMMBR),',                 C
               PARMKWD=PARM1
*
*   -- (1) OBJECT
*
            FLMALLOC IOTYPE=P,KEYREF=OBJ,DFLTTYP=OBJ,RECFM=FB,LRECL=80, C
                RECNUM=5000
*
*   -- (2) NOT USED
*
            FLMALLOC IOTYPE=N
*
*   -- (3) NOT USED
*
            FLMALLOC IOTYPE=N
*
*   -- (4) INCLUDE LIBRARIES
*
            FLMALLOC IOTYPE=I,KEYREF=SINC
*
*   -- (5) SOURCE
*
            FLMALLOC IOTYPE=S,KEYREF=SINC
*
*   -- (6) LISTING
*
            FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=VBA,LRECL=125,         C
                RECNUM=5000,PRINT=Y,DFLTTYP=FINLIST
```

Figure 47. Finnoga 4 Language Definition (Part 1 of 2)

```
*
*   -- (7) NOT USED
*
            FLMALLOC IOTYPE=N*
*   -- (8) FINNOGA COMPILER LIBRARIES
*
            FLMALLOC IOTYPE=A
             FLMCPYLB SYS1.FINNOGA.LIB
*
*   -- (9) NOT USED
*
            FLMALLOC IOTYPE=N
*
*   -- (10) WORK FILE
*
            FLMALLOC IOTYPE=W,LRECL=4000,RECFM=F,RECNUM=4000
*
*   -- (11) WORK FILE
*
            FLMALLOC IOTYPE=W,LRECL=4000,RECFM=F,RECNUM=4000
*
*5665-402 (C) COPYRIGHT IBM CORP 1980, 1989
```

Figure 48. Finnoga 4 Language Definition (Part 2 of 2)

Showing users how to write CC architecture definitions

Once you have written the language definition, and assembled and link-edited the project definition, your users can use SCLM to build their Finnoga 4 applications. To do so, however, they must know what information to supply in their architecture definitions. Table 15 on page 111 lists the SCLM-controlled inputs and outputs for the Finnoga 4 build. It includes the ddnames of the data sets that are input to and output from the Finnoga 4 compiler. In addition, a KEYREF value and brief description of each ddname is given.

Table 15. DDnames and KEYREFs

ddname	KEYREF	Description of data set(s) allocated
SYSLIN	OBJ	A partitioned data set into which the Finnoga 4 compiler writes the object module. The OBJ keyword in the compiler's option string specifies the member name to use.
SYSLIB	SINC	One or more partitioned data sets through which the Finnoga 4 compiler searches for include members.
SYSIN	SINC	A sequential data set that contains Finnoga 4 source to be compiled.
SYSPRINT	LIST	A sequential listings data set. The Finnoga 4 compiler writes out a copy of the source that was compiled along with any error, warning, and informational messages.

In addition, the PARM1 parameter is used in the FLMTRNSL macro for the Finnoga 4 compiler.

When your users write CC architecture definitions for their Finnoga 4 applications, they must include each of the preceding KEYREFs. A typical Finnoga 4 CC architecture definition looks like this:

```
SINC   PROG   SOURCE
SINC   SUB1   SOURCE
OBJ    PROG   OBJ
LIST   PROG   FINLIST
PARM1  OPTIMIZE
```

This CC architecture definition, along with the language definition previously written, tells SCLM to compile the concatenation of Finnoga 4 members PROG and SUB1 in data set type SOURCE. The resulting

object module and listing are to be saved in data set types OBJ and FINLIST, respectively. When the source is compiled, you want to use the OPTIMIZE compiler option.

You do not have to specify the modules that are included from ddname SYSLIB. Simply allocate SYSLIB to the proper libraries (with an IOTYPE=I) and the compiler will find the included members.

This simple template is all you have to give to your users. When they edit their Finnoga 4 source, they need to specify FINNOGA as the language name. Then they create their architecture definitions like the preceding one. SCLM and the language definition you created will perform the rest of the work.

Convert your JCL decks to architecture definitions

Suppose your Finnoga 4 users have a library of JCL that they have been using to compile their Finnoga 4 source. The following example uses a sample Finnoga 4 compile job and shows how you would write an architecture definition with the information in the JCL. The JCL deck that you use might look like this:

```
//JOB  ...
//FINNOGA EXEC PGM=FNGAA40,
//        PARM='SOURCE,NOMACRO,OBJ(PROG1),NOOPTIMIZE'
//SYSLIN   DD DSN=USER02.PRIVATE.OBJ,DISP=OLD
//SYSLIB   DD DSN=USER02.PRIVATE.FINNOGA,DISP=SHR
//SYSIN    DD DSN=USER02.PRIVATE.FINNOGA(MAIN),DISP=SHR
//         DD DSN=USER02.PRIVATE.FINNOGA(SUB1),DISP=SHR
//         DD DSN=USER02.PRIVATE.FINNOGA(SUB2),DISP=SHR
//SYSPRINT DD SYSOUT=A
//FINLIB   DD DSN=SYS1.FINNOGA.LIB,DISP=SHR
//SYSUT1   DD UNIT=SYSDA,VOL=SER=,DCB=(LRECL=4000,RECFM=F),
//         SPACE=(TRK,(10,10))
//SYSUT2   DD UNIT=SYSDA,VOL=SER=,DCB=(LRECL=4000,RECFM=F),
//         SPACE=(TRK,(10,10))
```

In this example, you want SCLM to control the modules that are input or output through ddnames SYSIN, SYSLIN, and SYSPRINT. For the Finnoga 4 language definition, the keywords SINC, OBJ and LIST have been assigned to those modules. You create the architecture definition by listing the modules involved in the build and identifying their roles with the keywords SINC, OBJ, and LIST. In addition, you tell SCLM to concatenate the NOOPTIMIZE option to the end of the OPTIONS string being passed to the translator using the PARM1 keyword.

```
SINC    MAIN   SOURCE
SINC    SUB1   SOURCE
SINC    SUB2   SOURCE
OBJ     PROG1  OBJ
LIST    MAIN   FINLIST
PARM1   NOOPTIMIZE
```

Now you are prepared to move this application under SCLM control:

1. Copy the members MAIN, SUB1, and SUB2 from 'USER02.PRIVATE.FINNOGA' to a development group in the SCLM project hierarchy. In this example, the data set type is SOURCE. Also copy over any included source members.
2. Use the SCLM Migration Utility to migrate your source members using the language name FINNOGA (the name specified on the FLMLANGL macro).
3. Use the SCLM editor to create the architecture definition. Unless you have modified the ARCHDEF language definition, the language of this architecture definition should be ARCHDEF. SCLM asks for the language name when you first enter the SAVE or END edit command.

Your user is now ready to compile this application using SCLM. The source members are under SCLM control as are the architecture definitions. The object module and the Finnoga 4 listing have not yet been created. To build this application, select Build (option 10.4) from the SCLM Main Menu and enter the project, group, type, and member name of the architecture definition (ARCHDEF).

Defining a preprocessor to SCLM

Suppose that some of your Finnoga 4 users run a preprocessor step on their Finnoga 4 source before compiling it. How do you define that two-step build process to SCLM? Using another fictitious product, the Panda Universal Preprocessor (PUPP), you can specify that some Finnoga 4 source is to be run through PUPP before it gets compiled.

Again, you need to list the ddnames used by the translator you want to define. In this case, assume that PUPP uses three ddnames:

Table 16. DDnames Used by a Hypothetical Preprocessor

DDname	Description of file(s) allocated
SYSIN	A sequential data set containing the Finnoga 4 source to be preprocessed.
SYSOUT	A sequential data set to which the preprocessed Finnoga 4 source is written. You want to compile the contents of this data set.
SYSPRINT	A listing data set containing Panda Universal Preprocessor messages and warnings.

In this example, the ddnames are not numbered because you will not use the PUPP ddname substitution list. Instead, you will use the ddname substitution list supported by the Finnoga 4 compiler to link the two build steps together.

Your users want SCLM to keep the listing data set produced by PUPP, but they do not want to keep the intermediate copy of the preprocessed source (the output in SYSOUT). The preprocessed source should be passed to the Finnoga 4 compiler and then deleted.

Because you want to preprocess some but not all of the Finnoga 4 source, you should define two different build processes to SCLM. You have already defined the latter build process (for language FINNOGA), and you will not change that language definition. For the two-step build process, however, you will create a new language definition with a different language name. The users must assign the correct language name to each Finnoga 4 source member.

The new language definition is very much like the first language definition, so you can copy the first definition into a second PROJDEFS.SOURCE member and modify it there.

The new language definition (copied from the first definition) has two FLMTRNSL macros: one for the parser, and the other for the Finnoga 4 compiler. You will add a third FLMTRNSL for the preprocessor, using the same macros and keywords as you used in the previous example. Enter this example before the FLMTRNSL for the Finnoga 4 compiler and after the last FLMALLOC for the parser. The order of execution is then parse, preprocess, and compile.

```
         FLMTRNSL   CALLNAM='PANDA U PREP',                            C
               FUNCTN=BUILD,                                           C
               COMPILE=PANDA01,                                        C
               GOODRC=0,                                               C
               PORDER=1,                                               C
               OPTIONS='NOTRACE'
*
*   -- SOURCE
*
           FLMALLOC IOTYPE=S,KEYREF=SINC,DDNAME=SYSIN
*
*   -- PREPROCESSED SOURCE
*
           FLMALLOC IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,            C
               DDNAME=SYSOUT
*
*   -- LISTING
*
           FLMALLOC  IOTYPE=O,KEYREF=OUT1,RECFM=VBA,LRECL=125,         C
               RECNUM=5000,PRINT=Y,DFLTTYP=PUPLIST,DDNAME=SYSPRINT
*
```

Figure 49. Panda Universal Preprocessor

The following list describes the keywords that change so you can invoke the new language definition:

Keyword
Description

FUNCTN=
Identifies this translator as a build translator. There are now two build translators in this language definition: one for PUPP and one for the Finnoga 4 compiler. Define the PUPP translator first and the Finnoga 4 translator second to tell SCLM the order in which the translators are to be invoked.

OPTIONS=
Specifies the options string to be passed to the PUPP compiler. In this case, you do not want the trace option activated.

DDNAME=
Specify the DDNAME= keyword because you are not using a ddname substitution list to pass ddnames to PUPP. This parameter specifies which ddnames to allocate (the ddnames that PUPP uses).

IOTYPE=W
Specifies that ddname SYSOUT is to be allocated as a work file. In this example, the users do not want to save the processed source. When the build completes, this file is deleted. In a later step, this file is passed to the Finnoga 4 compiler.

KEYREF=OUT1
Specifies that the listing PUPP writes to ddname SYSPRINT is to be saved under SCLM control. You usually use KEYREF=LIST for this purpose. However, KEYREF=LIST is already being used by the translator definition for the Finnoga 4 compiler. Because you have already used the standard set of CC ARCHDEF keywords, you must use the OUTx keywords.

OUTx keywords are used to identify additional build outputs. You can use OUT0, OUT1, ...,OUT9 to specify additional outputs that SCLM is to control.

PRINT=Y
This listing and the Finnoga 4 listing are both written to the build listing data set.

Passing the source to the compiler

You must next make one change to the macros that define how to invoke the Finnoga 4 compiler. The source to be compiled no longer comes directly from the SCLM-controlled source libraries. Instead, you want SCLM to take the preprocessed source that PUPP writes to ddname SYSOUT and pass it to the Finnoga 4 compiler. This requires a change to the FLMALLOC macro that defines the ddname that gets put

into the SYSIN position in the ddname substitution list for the Finnoga 4 compiler. The new macro is illustrated as follows:

```
*
*   -- (5) SOURCE
*
      FLMALLOC IOTYPE=U,DDNAME=SYSOUT
```

You use a different IOTYPE value (IOTYPE=U) to indicate that the ddname to be placed in the ddname substitution list has already been allocated in a previous build step. In this case, DDNAME=SYSOUT tells SCLM to place the name SYSOUT in position 5 of the ddname substitution list and go on to the next ddname. When the Finnoga 4 compiler runs, it reads the source from ddname SYSOUT.

The new language definition is shown in Figure 50 on page 116. Note that the new language has been specified on the FLMLANGL macro.

```
*********************************************************
*  FINNOGA 4 LANGUAGE DEFINITION
*********************************************************
*
         FLMLANGL LANG=FINPUPP,VERSION=FINN4
*
*********************************************************
         TYPES TO SEARCH FOR INCLUDES
*********************************************************
*
         FLMINCLS TYPES=(INCLUDE,@@FLMTYP)
*
*********************************************************

*  PARSE TRANSLATOR DEFINITION
*********************************************************
*
         FLMTRNSL   CALLNAM='FINNOGA PARSER',                          C
               FUNCTN=PARSE,                                           C
               COMPILE=FINPARSE,                                       C
               DSNAME=SCLM.PROJDEFS.LOAD,                              C
               PORDER=1,                                               C
               OPTIONS=(@@FLMSIZ,@@FLMSTP,@@FLMLIS)
*
*   -- SOURCE --
*
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB  @@FLMDSN(@@FLMMBR)
*
*********************************************************
*  BUILD TRANSLATOR DEFINITION
*********************************************************
*
* PREPROCESSOR STEP
*
         FLMTRNSL   CALLNAM='PANDA U PREP',                            C
               FUNCTN=BUILD,                                           C
               COMPILE=PANDA01,                                        C
               GOODRC=0,                                               C
               PORDER=1,                                               C
               OPTIONS='NOTRACE'
*
*   -- SOURCE
*
           FLMALLOC IOTYPE=S,KEYREF=SINC,DDNAME=SYSIN
*
*   -- PREPROCESSED SOURCE
*
           FLMALLOC IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,            C
               DDNAME=SYSOUT
*
*   -- LISTING
*
           FLMALLOC  IOTYPE=O,KEYREF=OUT1,RECFM=VBA,LRECL=125,         C
               RECNUM=5000,PRINT=Y,DFLTTYP=PUPLIST,DDNAME=SYSPRINT
*
* COMPILE STEP
*
         FLMTRNSL   CALLNAM='FINNOGA 4',                               C
               FUNCTN=BUILD,                                           C
               COMPILE=FNGAA40,                                        C
               GOODRC=0,                                               C
               PORDER=3,                                               C
               OPTIONS='SOURCE,NOMACRO,OBJ(@FLMMBR)',                  C
               PARMKWD=PARM1
```

Figure 50. Finnoga/PUPP Language Definition (Part 1 of 2)

```
*
*   -- (1) OBJECT
*
            FLMALLOC IOTYPE=P,KEYREF=OBJ,DFLTTYP=OBJ,RECFM=FB,          C
                LRECL=80,RECNUM=5000
*
*   -- (2) NOT USED
*
            FLMALLOC IOTYPE=N
*
*   -- (3) NOT USED
*
            FLMALLOC IOTYPE=N
*
*   -- (4) INCLUDE LIBRARIES
*
            FLMALLOC IOTYPE=I,KEYREF=SINC
*
*   -- (5) SOURCE
*
            FLMALLOC IOTYPE=U,DDNAME=SYSOUT
*
*   -- (6) LISTING
*
            FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=VBA,LRECL=125,         C
                RECNUM=5000,PRINT=Y,DFLTTYP=FINLIST
*
*   -- (7) NOT USED
*
            FLMALLOC IOTYPE=N
*
*   -- (8) FINNOGA COMPILER LIBRARIES
*
            FLMALLOC IOTYPE=A
             FLMCPYLB SYS1.FINNOGA.LIB
*
*   -- (9) NOT USED
*
      FLMALLOC IOTYPE=N
*
*   -- (10) WORK FILE
*
      FLMALLOC IOTYPE=W,LRECL=4000,RECFM=F,RECNUM=4000
*
*   -- (11) WORK FILE
*
      FLMALLOC IOTYPE=W,LRECL=4000,RECFM=F,RECNUM=4000
*
*5665-402 (C) COPYRIGHT IBM CORP 1980, 1989
```

Figure 51. Finnoga/PUPP Language Definition (Part 2 of 2)

The following example illustrates an architecture definition to build a program using two translators:

```
SINC    PROG7  SOURCE
OBJ     PROG7  OBJ
LIST    PROG7  FINLIST
OUT1    PROG7  PUPLIST
PARM1   NOOPTIMIZE
```

Figure 52. Architecture Definition Example

The only difference between this architecture definition and the Finnoga 4 CC architecture definition is the presence of the OUT1 keyword. This keyword specifies the type and member into which the PUPP listing is saved. In addition to specifying the OUT1 keyword in their architecture definitions, users who use this language definition to build their Finnoga 4 source must also remember to specify the language name FINPUPP for that Finnoga 4 source in the FLMLANGL macro statement.

Converting JCL to SCLM language definitions

Many sites use Job Control Language (JCL) to run preprocessors, compilers, linkage editors, and other tools used in the development process. SCLM supports developers and project managers through the use of language definitions that tell SCLM how to parse, build, and promote members of an SCLM-controlled data set. Language definitions can also specify additional translators to execute for the COPY, PURGE, and VERIFY functions. Because the SCLM language definitions provide an easier method of implementing processing control than JCL does, many sites have found it beneficial to convert their JCL to SCLM language definitions. To ease the conversion process, SCLM provides sample language definitions that you can tailor to the special needs of your site.

This section explains how to construct SCLM language definitions to replace existing JCL decks. Examples illustrate the basic principles underlying a successful migration from JCL to SCLM and also demonstrate methods for avoiding potential problems and conflicts.

Before you begin

Before you try to convert your existing JCL decks to SCLM language definitions, you must obtain and review "expanded" listings of the JCL. The "expanded JCL" listings allow you to determine the actual values of the symbolic parameters in the JCL; these values include data set names, options, and other information that is required for successful translation to an SCLM language definition. You will also need to know the order in which programs are executed in the JCL, and the condition codes that are expected from each program. Your system administrator should be able to help you locate this information.

You should also review the information in Chapter 21, “SCLM macros,” on page 491, paying special attention to the following macros and their parameters:

- FLMTRNSL
- FLMTCOND
- FLMALLOC
- FLMCPYLB
- FLMINCLS
- FLMTOPTS

Capabilities and restrictions

There are two basic equivalencies that you will use to convert JCL cards to SCLM macro statements:

- Every JCL EXEC card with PGM=abc will correspond to an FLMTRNSL macro with COMPILE=abc in your language definition. Conditional execution of BUILD translators may be addressed through use of the FLMTCOND macro.
- Every JCL DD card will correspond to an FLMALLOC macro or an FLMSYSLB macro associated with an FLMALLOC macro in your language definition.

  In the case of STEPLIB, the JCL DD card will correspond to the DSNAME parameter in the FLMTRNSL macro. A STEPLIB concatenation of more than one data set would use the TASKLIB parameter. The TASKLIB parameter is set to the ddname associated with the data set concatenation. FLMCPYLBs are used to specify the data sets on an FLMALLOC macro with DDNAME set to the TASKLIB ddname. When both DSNAME and TASKLIB are specified, the DSNAME data set is searched first, followed by the TASKLIB data sets, followed by the system concatenation.

  In the case of SYSLIB-type ddnames for a compiler, the data sets must be specified FLMSYSLBs. Then either ALCSYSLB=Y must be specified on the FLMLANGL macro and/or FLMCPYLBs must be specified for the appropriate FLMALLOC macros. For an example of this, refer to the COBOL (FLM@COB2) or C/370 (FLM@C370) language definitions supplied with SCLM.

Three areas of restrictions can prevent a simple, one-to-one translation of JCL cards to SCLM macro statements:

- Backward referencing of data definition names (DDs)

  If a JCL DD card uses the "refer back" technique to reference a previous DD card (other than the card in the preceding step), or if a DD card refers to a data set using a ddname that differs from the data set's ddname in a prior step, conversion to an SCLM language definition can involve the use of an intermediate translator or a ddname substitution list in order to allocate the correct data set name for the program. (An intermediate translator is not needed if the succeeding translator supports DDNAME substitution lists; in this case, the succeeding translator can "hard code" the DDNAME and use IOTYPE=U on the FLMALLOC macro.)

- Complex conditional execution

  A JCL deck that specifies skipping all steps after a specified condition code from one or more previous steps is directly converted to appropriate FLMTRNSL macros with appropriate GOODRC values. Other conditional executions of BUILD translators can be addressed by using the FLMTCOND macro. For example, if the JCL is set up to run BUILD translator X if any previous return code is 4, but run Build translator Y if any previous return code is 8, you can use the FLMTCOND macro. FLMTCOND is only valid for use with BUILD translators. Conditional execution of non-BUILD translators can require modification of the translators or interface programs to handle the control of execution.

- TSO Address Space compatibility

  Some programs that run from JCL will not run in the TSO Address Space in which SCLM resides without a special interface translator. IBM has provided interface programs for several common IBM programs with this characteristic. For example, the FLMTMSI (SCRIPT), FLMTMJI (JOVIAL), and FLMTMMI (DFSUNUB0) translators all use the TSO Service Facility IKJEFTSR.

  If you have JCL that runs program XYZ without any errors, but fails when you try to run program XYZ from an FLMTRNSL macro, this may be the problem. You must write a translator to call the program using IKJEFTSR.

The following sections describe how to convert JCL cards and decks into functionally equivalent SCLM language definitions and provide suggested strategies for working around restrictions and conflicts.

Converting JCL cards to SCLM macro statements

This section contains examples of JCL decks and their SCLM language definition equivalents.

Executing programs

The SCLM FLMTRNSL macro is similar to a JCL EXEC (EXECUTE) card. Figure 53 on page 119 shows a single JCL card that runs a program named IEFBR14.

```
//STEP1    EXEC    PGM=IEFBR14
```

Figure 53. JCL: Execute IEFBR14

Figure 54 on page 119 shows an SCLM FLMTRNSL macro that performs the same task as the JCL card in Figure 53 on page 119.

```
FLMTRNSL   COMPILE=IEFBR14,FUNCTN=BUILD,PORDER=0
```

Figure 54. SCLM: Execute IEFBR14

FLMTRNSL's COMPILE option specifies the name of the program to execute (IEFBR14). The FUNCTN parameter specifies here that IEFBR14 will be invoked when the user requests a BUILD, and the PORDER value of 0 tells SCLM that neither an option list nor a ddname substitution list will be passed to IEFBR14.

The next figure is a slightly more complex example. We want to use a translator program named GAC to copy the contents of TSOSCxx.DEV1.SOURCE(MEMBER1) into TSOSCxx.DEV1.LIST(MEMBER1). The GAC program itself requires a SYSIN data set, which is empty in this example.

```
//STEP1    EXEC    PGM=GAC
//SYSIN    DD      DUMMY
//INPUT    DD      DSN=TSOSCxx.DEV1.SOURCE(MEMBER1),DISP=SHR
//OUTPUT   DD      DSN=TSOSCxx.DEV1.LIST(MEMBER1),DISP=SHR
```

Figure 55. JCL: Execute GAC

Figure 56 on page 120 shows the SCLM language definition that performs the same task as the JCL in Figure 55 on page 120.

```
FLMTRNSL   COMPILE=GAC,FUNCTN=BUILD,PORDER=0
FLMALLOC   IOTYPE=A,DDNAME=SYSIN
 FLMCPYLB   NULLFILE
FLMALLOC   IOTYPE=A,DDNAME=INPUT
 FLMCPYLB   TSOSCxx.DEV1.SOURCE(MEMBER1)
FLMALLOC   IOTYPE=A,DDNAME=OUTPUT
 FLMCPYLB   TSOSCxx.DEV1.LIST(MEMBER1)
```

Figure 56. SCLM Language Definition: Execute GAC

As before, the FLMTRNSL macro is used to specify the name of the program to run. The FLMALLOC and FLMCPYLB statements allocate the existing data sets to ddnames.

Conditional execution

In Figure 57 on page 120, program XYZ runs only if the return code from program ABC is less than five.

```
//STEP1     EXEC    PGM=ABC
//STEP2     EXEC    PGM=XYZ,COND=(4,LT)
```

Figure 57. JCL: Conditional Execution

In SCLM, the GOODRC parameter on the FLMTRNSL macro allows you to specify return code values for conditional execution. In Figure 58 on page 120, the GOODRC parameter for program ABC is set to 4. If ABC ends with a return code greater than four, processing ends; program XYZ will not execute.

```
FLMTRNSL   COMPILE=ABC,FUNCTN=BUILD,PORDER=0,GOODRC=4
FLMTRNSL   COMPILE=XYZ,FUNCTN=BUILD,PORDER=0
```

Figure 58. SCLM Language Definition: Conditional Execution

In Figure 59 on page 120, program XYZ runs only if the return code from program ABC is less than 5. Program MBS is to execute after program XYZ regardless of the previous return codes.

```
//STEP1     EXEC    PGM=ABC
//STEP2     EXEC    PGM=XYZ,COND=(4,LT)
//STEP3     EXEC    PGM=MBS
```

Figure 59. JCL: Complex Conditional Execution

In SCLM, the GOODRC parameter on the FLMTRNSL macro specifies when to skip all remaining translators in the language definition. To skip selected translators, the FLMTCOND macro can be used. In Figure 60 on page 121 the FLMTCOND macro specifies that execution may skip program XYZ but continue with program MBS.

```
FLMTRNSL   COMPILE=ABC,FUNCTN=BUILD,PORDER=0
FLMTRNSL   COMPILE=XYZ,FUNCTN=BUILD,PORDER=0
  FLMTCOND ACTION=SKIP,WHEN=(*,GE,5)
FLMTRNSL   COMPILE=MBS,FUNCTN=BUILD,PORDER=0
```

Figure 60. SCLM Language Definition: Complex Conditional Execution

Sample JCL conversion

This section contains commented sample JCL and language definitions that perform the same tasks: invoking the CICS preprocessor and then invoking the OS COBOL compiler to produce an object module. Figure 61 on page 123 contains the JCL used to accomplish these tasks; Figure 63 on page 125 contains the equivalent SCLM language definition. Each sample contains comments with step numbers. The step descriptions that follow relate a line or command from the JCL to the equivalent SCLM language definition macro, option, or command.

1. The JCL has a job step named TRN, which is the first translator called in this job.

   SCLM uses an FLMTRNSL macro to call this translator. This is the first FLMTRNSL macro for build in the language definition.
2. Job step TRN executes a program called DFHECP1$, the CICS preprocessor for COBOL.

   SCLM uses the COMPILE=DFHECP1$ statement on the FLMTRNSL macro.
3. The STEPLIB line in job step TRN tells the job where to find the program DFHECP1$.

   SCLM uses the DSNAME option on the FLMTRNSL macro. Both the STEPLIB and DSNAME point to the same data set, CICS.TS31.CICS.SDFHLOAD.
4. The SYSIN statement defines the data set that contains the member to compile.

   SCLM uses an FLMALLOC macro to allocate the SYSIN data set to a ddname for the CICS preprocessor. Because we are using PORDER=1, the FLMALLOC macro assigns the ddname, SYSIN, that the CICS preprocessor is expecting.
5. The TRN job step sends the preprocessor listing to the printer using the SYSPRINT statement.

   SCLM uses an FLMALLOC macro to allocate an output data set to the ddname SYSPRINT.
6. The SYSPUNCH line in the TRN step creates the output of the CICS preprocessor and passes it to the next job step (COB) as a temporary file.

   SCLM uses an FLMALLOC macro with IOTYPE=W to allocate a work (temporary) file with the ddname of SYSPUNCH. This work file is passed to the next job step (FLMTRNSL).
7. The JCL has a job step named COB, which is the second translator called in this job.

   SCLM uses an FLMTRNSL macro to call this translator. This is the second FLMTRNSL macro for build in our language definition.
8. The job step COB executes (EXEC PGM=) a program called IGYCRCTL, the compiler for COBOL.

   SCLM uses the COMPILE=IGYCRCTL statement on the FLMTRNSL macro.
9. To pass compiler options to the OS COBOL compiler, the COB job step uses a PARM= command.

   SCLM uses the OPTIONS= statement on the FLMTRNSL macro to perform the same task.
10. This job has conditional execution for the COB step via the COND(5,LE) JCL command. The COB step will not execute if the return code of the TRN step is greater than 4.

    SCLM sets the GOODRC keyword parameter for the TRN step (CICS preprocessor) equal to 4. Build halts execution of all translators following the TRN step in the language definition if the return code from the TRN step is greater than 4.
11. The STEPLIB statement in job step COB tells the job where to find the program IGYCRCTL.

    SCLM uses the DSNAME= option on the FLMTRNSL macro. Both the STEPLIB and DSNAME point to the same data set, IGY.SIGYCOMP.

12. The SYSLIB statement in job step COB tells the job where to find the system type includes.

    The language definition uses the FLMSYSLB macro with IOTYPE=I and the FLMINCLS macro to do the same task.

    SCLM allocates these project data sets allocated for IOTYPE=I before the data sets on the FLMCPYLB macro(s). ALCSYSLB=Y parameter must be specified on the FLMLANGL macro to ensure that the FLMSYSLB data sets are allocated to the IOTYPE=I ddnames.

    Because PORDER=3 is being used, the SYSLIB DD is the fourth ddname passed to the compiler in a ddname substitution list. The COBOL compiler uses the fourth ddname as SYSLIB no matter what value is assigned to the DDNAME keyword parameter on the FLMALLOC macro.
13. For each system library specified for the SYSLIB DD, the language definition has an FLMSYSLB macro. In this case both CICS.TS31.CICS.SDFHCOB and CICS.TS31.CICS.SDFHMAC are specified.
14. The COB job step sends the compile listing to the printer using the SYSPRINT statement.

    SCLM uses an FLMALLOC macro to allocate an output data set to the ddname SYSPRINT.
15. In the COB job step, the SYSIN DD statement identifies the data set that contains the member to compile. This is the output of the CICS preprocessor step TRN.

    SCLM uses an FLMALLOC macro with IOTYPE=U to refer to a ddname from a prior step. The language definition instructs MVS to allocate the data set assigned in the TRN step to the ddname SYSPUNCH.
16. The SYSLIN statement in the COB step identifies the output data set for object code created by the COBOL compiler.

    The language definition uses an FLMALLOC macro with IOTYPE=O to allocate an output file. This FLMALLOC macro is the first in the COB FLMTRNSL because when using PORDER=3, the COBOL compiler expects the output data set ddname to be first in a ddname substitution list.
17. The COB step allocates SYSUT1 as a temporary work file for the COBOL compiler.

    SCLM's language definition uses an FLMALLOC macro with IOTYPE=W to perform the same task. This must be the eighth file provided to the COBOL compiler because PORDER=3 tells SCLM that we are using a ddname substitution list.
18. The COB step allocates SYSUT2 as a temporary work file for the COBOL compiler.

    SCLM's language definition uses an FLMALLOC macro with IOTYPE=W to perform the same task. This must be the ninth file provided to the COBOL compiler because we are using a ddname substitution list.
19. The COB step allocates SYSUT3 as a temporary work file for the COBOL compiler.

    SCLM's language definition uses an FLMALLOC macro with IOTYPE=W to perform the same task. This must be the tenth file provided to the COBOL compiler because we are using a ddname substitution list.
20. The COB step allocates SYSUT4 as a temporary work file for the COBOL compiler. SCLM's language definition uses an FLMALLOC macro with IOTYPE=W to perform the same task. This must be the 11th file provided to the COBOL compiler because we are using a ddname substitution list.
21. The COB step allocates SYSUT5 as a temporary work file for the COBOL compiler.

    SCLM's language definition uses an FLMALLOC macro with IOTYPE=W to perform the same task. This must be the 13th file provided to the COBOL compiler because we are using a ddname substitution list.
22. The COB step allocates SYSUT6 as a temporary work file for the COBOL compiler.

    SCLM's language definition uses an FLMALLOC macro with IOTYPE=W to perform the same task. This must be the 14h file provided to the COBOL compiler because we are using a ddname substitution list.
23. The COB step allocates SYSUT7 as a temporary work file for the COBOL compiler.

SCLM's language definition uses an FLMALLOC macro with IOTYPE=W to perform the same task. This must be the 15th file provided to the COBOL compiler because we are using a ddname substitution list.

24. **SCLM language definition only**

    The language definition uses PORDER=3 for the COBOL compiler step (COB) to use a ddname substitution list. A ddname substitution list provides an ordered list(defined by the translator) of ddnames such that the position of a ddname in the list, and not the actual ddname, is used by the translator for a specific file.

    The input file for the compiler must be the output file from the CICS preprocessor. The ddname assigned to the TRN step is SYSPUNCH. Because this file has already been allocated to SYSPUNCH, another way (besides ddname) is needed to pass this file as the input to the compiler. By using PORDER=3, SCLM passes all the files that can be used by the COBOL compiler in the order specified for this compiler. To use PORDER=3, a specific parameter string must be built. The language definition must have an FLMALLOC macro for each of these parameters

    Those FLMALLOCs that are tagged for STEP 24 are not applicable for the COBOL compiler. SCLM places 8 bytes of hexadecimal zeros into the ddname substitution list for each FLMALLOC with IOTYPE=N.

```
//USERIDC  JOB (AS05CR,T12,C531),'USERID',NOTIFY=&SYSUID,CLASS=A,
//             MSGCLASS=O,MSGLEVEL=(1,1)
//*
//*      THIS PROCEDURE CONTAINS 2 STEPS
//*      1.   EXEC THE CICS PREPROCESSOR
//*      2.   EXEC THE Enterprise COBOL COMPILER
//*
//*      CHANGE THE JOB NAME AND THE ACCOUNTING INFORMATION TO MEET THE
//*      REQUIREMENTS OF YOUR INSTALLATION.
//*
//*      CHANGE 'PROGNAME' TO THE NAME OF THE CICS/COBOL PROGRAM YOU
//*      WANT TO COMPILE.  CHANGE 'USERID' TO YOUR USERID.
//*
//*      CHANGE 'DEVL' TO THE GROUP THAT CONTAINS THE PROGRAM TO BE COMPILED.
//*
//* STEP 1: TRN STATEMENT; STEP 2: EXEC PGM STATEMENT
//*
//TRN    EXEC PGM=DFHECP1$,
//            REGION=2048K
//*
//* STEP 3: STEPLIB STATEMENT
//*
//STEPLIB DD DSN=CICS.TS31.CICS.SDFHLOAD,DISP=SHR
//*
//* STEP 4: SYSIN STATEMENT
//*
//SYSIN DD DSN=USERID.DEVL.SOURCE(PROGNAME),DISP=SHR
//*
//* STEP 5: SYSPRINT STATEMENT
//*
//SYSPRINT DD SYSOUT=A
//*
//* STEP 6: SYSPUNCH STATEMENT
//*
//SYSPUNCH DD DSN=&&SYSCIN,;
//             DISP=(,PASS),UNIT=SYSDA,
//             DCB=BLKSIZE=400,
//             SPACE=(400,(400,100))
//*
//* STEP 7: COB STATEMENT; STEP 8: EXEC PGM STATEMENT
```

Figure 61. JCL: Invoke COBOL Preprocessor and Compiler (Part 1 of 2)

```
//* STEP 9: PARM STATEMENT; STEP 10: COND STATEMENT
//*
//COB    EXEC PGM=IGYCRCTL,REGION=2048K,COND=(5,LE),
//               PARM='RENT,NODYNAM,LIB'
//*
//* STEP 11: STEPLIB STATEMENT
//*
//STEPLIB DD DSN=IGY.SIGYCOMP.DISP=SHR
//*
//* STEP 12: SYSLIB STATEMENT; STEP 13: DD STATEMENT
//*
//SYSLIB DD DSN=CICS.TS31.SDFHCOB,DISP=SHR
//       DD DSN=CICS.TS31.SDFHMAC,DISP=SHR
//*
//* STEP 14: SYSPRINT STATEMENT
//*
//SYSPRINT DD SYSOUT=O
//*
//* STEP 15: SYSIN STATEMENT
//*
//SYSIN DD DSN=&&SYSCIN,DISP=(OLD,DELETE)
//*
//* STEP 16: SYSLIN STATEMENT
//*
//SYSLIN DD DSN=USERID.DEVLEV.OBJ(PROGNAME),DISP=SHR
//*
//* STEP 17: SYSUT1 STATEMENT
//*
//SYSUT1 DD UNIT=SYSDA,SPACE=(460,(350,100))
//*
//* STEP 18: SYSUT2 STATEMENT
//*
//SYSUT2 DD UNIT=SYSDA,SPACE=(460,(350,100))
//*
//* STEP 19: SYSUT3 STATEMENT
//*
//SYSUT3 DD UNIT=SYSDA,SPACE=(460,(350,100))
//*
//* STEP 20: SYSUT4 STATEMENT
//*
//SYSUT4 DD UNIT=SYSDA,SPACE=(460,(350,100))
//*
//* STEP 21: SYSUT5 STATEMENT
//*
//SYSUT5 DD UNIT=SYSDA,SPACE=(460,(350,100))
//*
//* STEP 22: SYSUT6 STATEMENT
//*
//SYSUT6 DD UNIT-SYSDA,SPACE=(460,(350,100))
//*
//* STEP 23: SYSUT7 STATEMENT
//*
//SYSUT7 DD UNIT=SYSDA,SPACE=(460,(350,100))
```

Figure 62. JCL: Invoke COBOL Preprocessor and Compiler (Part 2 of 2)

```
*********************************************************************
*          SCLM LANGUAGE DEFINITION FOR
*          COBOL WITH CICS 3.1 PREPROCESSOR
*
* CICS OUTPUT IS PASSED VIA THE CICSTRAN DD ALLOCATION TO COBOL.
*
* POINT THE FLMSYSLB MACRO(S) AT ALL 'STATIC' COPY DATASETS.
* CUSTOMIZE THE 'OPTIONS' AND 'GOODRC' FIELDS TO YOUR STANDARDS.
* ADD THE 'DSNAME' FIELD IF THE TRANSLATOR IS IN A PRIVATE LIBRARY.
* WHEN A NEW TRANSLATOR VERSION REQUIRES TOTAL RECOMPILATION FOR THIS
* LANGUAGE, THE 'VERSION' FIELD ON FLMLANGL SHOULD BE CHANGED.
*********************************************************************
*
*STEP 13
COBCICS  FLMSYSLB CICS.TS31.CICS.SDFHCOB
         FLMSYSLB CICS.TS31.CICS.SDFHMAC
*
         FLMLANGL LANG=COBCICS,VERSION=CICS31,ALCSYSLB=Y
*
* PARSER TRANSLATOR
*
          FLMTRNS  CALLNAM='SCLM COBOL PARSE',                        C
               FUNCTN=PARSE,                                          C
               COMPILE=FLMLPCBL,                                      C
               PORDER=1,                                              C
               OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,)
*      (*  SOURCE *)
           FLMALLOC IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
* BUILD TRANSLATORS
*        - CICS PRECOMPILE - STEP NAME TRN
*
* STEP 1
          FLMTRNSL CALLNAM='CICS PRE-COMPILE',                     C
               FUNCTN=BUILD,                                       C
* STEP 2
               COMPILE=DFHECP1$,                                   C
* STEP 3 (* STEPLIB *)
               DSNAME=CICS.TS31.CICS.SDFHLOAD,                     C
               VERSION=3.1,                                        C
* STEP 10 (* COND *)
               GOODRC=4,                                           C
               PORDER=1                                            C
               OPTIONS=(NOSEQ)
* STEP 4 (* SYSIN *)
           FLMALLOC IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,        C
               DDNAME=SYSIN
* STEP 5 (* SYSPRINT *)
           FLMALLOC IOTYPE=O,RECFM=FBA,LRECL=121,                  C
               RECNUM=35000,PRINT=Y,DDNAME=SYSPRINT
*
* STEP 6 (* SYSPUNCH *)
           FLMALLOC IOTYPE=W,RECFM=FB,LRECL=80,                    C
               RECNUM=5000,DDNAME=SYSPUNCH
*
* STEP 7 (*COBOL INTERFACE - STEP NAME COB *)
* STEP 8
           FLMTRNSL CALLNAM='COBOL COMPILE',                       C
               FUNCTN=BUILD,                                       C
               COMPILE=IGYCRCTL,                                   C
```

Figure 63. SCLM Language Definition: Invoke COBOL Preprocessor and Compiler (Part 1 of 2)

```
* STEP 11 (* STEPLIB *)
               DSNAME=IGY.SIGYCOMP,                                  C
               VERSION=3.3.1,                                        C
               GOODRC=4,                                             C
* STEP 24
               PORDER=3,                                             C
* STEP 9 (* PARMS *)
             OPTIONS=(RENT,NODYNAM,LIB)                              C
* DDNAME ALLOCATIONS
* STEP 16
* 1 (* SYSLIN *)
          FLMALLOC IOTYPE=O,KEYREF=OBJ,RECFM=FB,LRECL=80,           C
              RECNUM=5000,DFLTTYP=OBJ
* STEP 24
* 2 (* N/A *)
          FLMALLOC IOTYPE=N
* STEP 24
* 3 (* N/A *)
          FLMALLOC IOTYPE=N
* STEP 12; STEP 13
* 4 (* SYSLIB *)
          FLMALLOC IOTYPE=I,KEYREF=SINC
* STEP 15
* 5 (* SYSIN *)
          FLMALLOC IOTYPE=U,KEYREF=SINC,DDNAME=SYSPUNCH
* STEP 14
* 6 (* SYSPRINT *)
          FLMALLOC IOTYPE=O,KEYREF=LIST,RECFM=FBA,LRECL=133,        C
              RECNUM=25000,PRINT=Y,DFLTTYP=LIST
* STEP 24
* 7 (* SYSPUNCH *)
          FLMALLOC IOTYPE=N
           FLMCPYLB NULLFILE
* STEP 17
* 8 (* SYSUT1 *)
          FLMALLOC IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* STEP 18
* 9 (* SYSUT2 *)
          FLMALLOC IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* STEP 19
* 10 (* SYSUT3 *)
          FLMALLOC IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* STEP 20
* 11 (* SYSUT4 *)
          FLMALLOC IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* STEP 24
* 12 (* SYSTERM *)
          FLMALLOC IOTYPE=N
           FLMCPYLB NULLFILE
* STEP 21
* 13 (* SYSUT5 *)
          FLMALLOC IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* STEP 22
* 14 (* SYSUT6 *)
          FLMALLOC IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
*STEP 23
* 15 (* SYSUT7 *)
          FLMALLOC IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 5694-A01     COPYRIGHT IBM CORP 1980, 1989,2007
```

Figure 64. SCLM Language Definition: Invoke COBOL Preprocessor and Compiler (Part 2 of 2)

Note: For reference purposes, the language definition shown in Figure 63 on page 125 contains comments with step numbers placed in the middle of commands; for this language definition to run, these comments must be removed.

Chapter 6. Using SCLM and Tivoli Information Management for z/OS

Tivoli Information Management sample code is provided as member FLM00CVE in SAMPLIB. It illustrates communication between SCLM and Tivoli Information Management. The sample is implemented in the REXX language and uses the Information Management REXX high-level API. The sample verifies a programmer's authority to update an SCLM-controlled module based on the SCLM change code provided by the programmer.

FLM00CVE retrieves the Information Management problem record identified by the change code, and verifies:

1. The record exists.
2. The Problem Status field is set to OPEN.
3. The Assignee Name field is the same as the userid parameter passed by SCLM.

Required environment

- Tivoli Information Management for z/OS Version 1.2 or later must be installed on the target MVS system.
- The Information Management REXX HLAPI (BLGYRXM) must be installed on the system.
- A valid Information Management session name, class name, and default REXX/HLAPI Record-Retrieve PIDT table must exist. The sample uses session BLGSES00, class MASTER, and table BLGYPRR.
- For software verification purposes, at least one problem record meeting the desired criteria should exist in the Information Management database.

Description of user program interaction

The FLM00CVE REXX Exec can be invoked as a regular MVS Exec, but it is designed to be invoked as an SCLM change code verification user exit. If FLM00CVE is invoked as a user exit, the Information Management-specific arguments are passed by the SCLM option list defined in the FLMCNTRL macro. The SCLM-specific arguments are appended to the Information Management arguments.

Input parameters

Two different sets of parameters are passed to the sample as one parameter string. User options are specified in the Options entry of the FLMCNTRL macro. SCLM parameters are the standard set of parameters passed to the SCLM Exit.

Option list format

The option list format is as follows:

```
      pica_tabn,
      pica_clsn,
      pica_sess,
      pica_clsc,
      pica_dbid,
      pica_msgd,
      pica_spli,
      pica_stxt,
      pica_tint,
      pica_usrn,
      group,
      type,
```

© Copyright IBM Corp. 1990, 2019

```
      member,
      language,
      userid,
      auth code,
      change code
```

Information Management parameters

The required Information Management parameters are:

pica_tabn
Specifies the name of the Information Management Record Retrieval table. The table defines the fields within a problem record. The default is BLGYPRR (shipped with Information Management). This must be the name of the table used in your installation.

pica_clsn
Specifies the Information Management Privilege Class record that contains the registered user name authorized to retrieve a problem record. The default is MASTER. This must match your installation. The registered authorized user name (see pica_usrn) is optionally specified in option 10.

pica_sess
Specifies the name of the Information Management Session Member (BLGSESxx) load module. The default is BLGSES00. This parameter must match your installation.

The optional Information Management parameters are:

pica_clsc
Specifies the count of privilege class records that can be maintained in storage during the Information Management session. The default is one. The sample program uses only one privilege class record.

pica_dbid
Specifies the Problem Record database number. The default is 5, the standard Information Management database.

pica_msgd
Specifies the destination for Information Management API log messages. Messages can be either printed to an APIPRINT data set, returned on the message chain, or both. The default is C, return messages on the API message chain. The sample program interprets chained message return code and reason code values to provide English text messages. See “Error processing” on page 129 for more information.

pica_spli
Specifies the number of minutes that the activity log can print transaction results before the API closes and reopens the log. The default is `ten minutes` if message chaining (pica_msgd) is not selected, otherwise, it is zero.

pica_stxt
Specifies whether text data is to be retrieved from the problem record. Setting this value to NO suppresses text retrieval. The default is NO because the sample program does not process text fields in the problem record.

pica_tint
Specifies the transaction processing timeout interval. This field specifies the time in seconds that any Information Management API transaction can process before the API notifies the application of a timeout event. The default is 300 seconds.

pica_usrn
Specifies a name registered in the selected Privilege Class (see pica_clsn) that is authorized to retrieve problem records. The default is the TSO User ID of the SCLM user.

SCLM parameters

The SCLM parameters are:

group
Specifies the MVS data set Group name.

type
Specifies the MVS data set Type name.

member
Specifies the MVS partitioned data set Member name if selected, otherwise blank.

language
Specifies the language of the module selected. This is blank for Edit exits.

userid
Specifies the TSO User ID accessing SCLM. In the sample program, this value is compared to the Information Management Problem Record Assignee Name field (Information Management S-word: S0B5A) for authorization to modify the SCLM module.

auth_code
Specifies the authorization code of the member being edited.

change code
Specifies the Change Code entered by the SCLM user on the appropriate panel. This value is used by the sample program to specify the Information Management Problem Record Record_ID (RNID) to be retrieved. In the sample program, the Problem Record Current Status field (Information Management S-word: S0BEE) from the retrieved record is verified against the constant OPEN for authorization to modify the SCLM module.

Program flow

When the FLM00CVE program is invoked, the program flow is as follows:

1. Parse the argument string passed by invocation.
2. Perform the REXX/HLAPI Initialization function (HL01).
3. Perform the REXX/HLAPI Record Retrieve function (HL06).
4. Perform the REXX/HLAPI Termination function (HL02).
5. Verify that the user requesting to change the member has authority to do so based on information contained in the retrieved record.
6. Output error messages if applicable.
7. Return to caller passing return code as exit value.

Each of the steps above performs error-checking and return code analysis independently. If an error is noted, processing might terminate at that time or continue to another step. For example, after Information Management initialization has completed, Information Management Termination is attempted regardless of intervening errors; the transaction is not left hanging.

Error processing

When an error condition is encountered, the program issues an error message, if possible, and terminates processing with the appropriate return code. When a warning condition is encountered, the program issues a warning message and continues processing. When a warning or error is the result of an Information Management REXX/HLAPI call, a message appropriate to the reason code is displayed. If an Information Management message chain is available, the associated messages are also displayed.

The program initiates REXX/HLAPI with logging enabled. Error conditions are both printed to the session log and returned to the program in message chains, as appropriate.

For warning message instigated by the Information Management API interface, the program returns a return code of zero because SCLM considers any nonzero return code as an indication of failure. For API errors with return code 8 or higher, the program issues the appropriate messages and return code 8.

The program specifically tests for and reports the following input parameter errors:

- No input parameters.

- Missing or invalid REXX/HLAPI table name.
- Missing or invalid Information Management Class name.
- Missing or invalid Information Management Session ID.
- Missing or invalid User ID.
- Missing or invalid Change Code.
- Problem Record not found in the database.
- Problem Record Problem Status not "OPEN".
- Problem Record Assignee Name does not match User ID.
- Input parameters specified as "Ignored" are checked for presence and valid format, and a warning message is issued if warranted. However, the return code presented is zero.

Example

This example calls the FLM00CVE Exec through the SCLM verify change code exit.

```
IN FLMCNTRL MACRO:
  CCVFY=FLM00CVE,
  CCVFYDS=PROJ1.SAMPLIB.EXEC,
  CCVFYCM=TSOLNK,
  CCVFYOP=(BLGYPRR,MASTER,BLGSES00,1,5,C,300,NO,360,FLM00CVE,)
```

Where:

CCVFY=FLM00CVE
Specifies that the SCLM Verify Change Code exit be used and that member FLM00CVE be invoked.

CCVFYDS=PROJ1.SAMPLIB.EXEC
Specifies the MVS data set containing member FLM00CVE. In the example: "PROJ1.SAMPLIB.EXEC(FLM00CVE)"

CCVFYCM=TSOLNK
Specifies that FLM00CVE is invoked using the TSO service facility routine, the default for REXX Exec programs.

CCVFYOP=(exit routine parameters)
Specifies the parameters that are passed to the exit program.

Chapter 7. Understanding and using the customizable parsers

Parsers are provided as source code (in REXX) for those customers who need to extend or modify the behavior of the parsers supplied by IBM. This section explains the logic of the parsers as provided and gives examples of how to modify the parsers to suit your own needs and standards.

The customizable parsers supplied by IBM are:

FLMLRASM
Assembler H parser

FLMLRCBL
COBOL II parser

FLMLRCIS
C/C++ for MVS parser

FLMLRC2
C++ for Windows parser

FLMLRC37
C/370 parser

FLMLRDTL
DTL parser

FLMLRIPF
OS/2 IPF parser

These parsers can be found in the ISPF sample library, ISP.SISPSAMP.

The parsers as provided

The IBM-supplied parsers are provided as REXX source. If you do not require any changes to the functions provided, the source modules can be used. The parsers may also be compiled, pre-linked, and link-edited (using the IBM Compiler and Library for REXX/370 and the Linkage Editor) for optimum performance.

Use the CALLMETH=TSOLNK parameter on the FLMTRNSL macro to directly invoke SCLM translators written in REXX.

Sample language definitions

The sample language definitions are provided to demonstrate how to invoke the customizable parsers:

FLM@RASM
Assembler H sample language definition

FLM@RCBL
COBOL II sample language definition

FLM@RCIS
C/370 sample language definition

FLM@RC37
C/370 sample language definition

FLM@DTLC
DTL sample language definition

FLM@WBCC
C++ for Windows sample language definition

© Copyright IBM Corp. 1990, 2019

FLM@WIPF
OS/2 Help sample language definition

In addition, a sample REXX language definition, FLM@REXC, is provided to compile, pre-link, and link-edit REXX source code.

Parser error listings

For parsing errors with return codes of 4, 8, or 10, the parsers write error messages to a data set called *userid*.SCLMERR.LISTING. An error message consists of two or three lines. The first line is the error code: 4, 8, or 10. The second line and the third line (if it exists) contain one of the following pieces of information:

- One or more non-valid input parameters
- A dependency name that is greater than 8 characters in length
- A dependency name that cannot be stored in the dependency buffer because it is full
- A line of source containing an error
- A single quote or double quote that is mismatched and its line number

For more information about the return codes from the parsers, refer to Chapter 22, “SCLM translators,” on page 565.

Modifying the parsers

This section describes the general design of the customizable parsers and provides several examples of updating the parsers.

The parsers read each line of the source code and process tokens on each line. Tokens are discrete elements on a line of source code; they are language-dependent. For example, consider the following COBOL statement:

```
        MOVE 'SMITH' TO NAME.
```

Seven tokens appear in this example: `MOVE`, the two single quotation marks, `SMITH`, `TO`, `NAME`, and the period.

State variables are used to hold the current conditions and expectations created by the processing of prior tokens in order to process the current token. For example, if a single quote is found, the single quote state variable (`state.single`) is turned on. All tokens, regardless of multiple lines, are ignored until the matching single quote is found, or until the end of file is reached. In the COBOL and Assembler parsers, dependency names may be enclosed in quotes; all data after the dependency name is ignored until the matching quote is found. Dependency keywords (`COPY` or `EXEC SQL INCLUDE`) inside quotes are ignored. For example, consider the following COBOL statement:

```
        MOVE 'COPY B' TO ACTION.
```

B will not be placed into the dependency buffer because `COPY` will not be processed as a dependency keyword.

Because of these state variables, dependencies, comments (in C/370), quotes, and so on can span lines. Concatenation of keywords and dependency names (particularly in COBOL) is not supported by the parsers. If dependency names are split between lines, the partial dependency name will not be added by the REXX parser.

Adding more elaborate parsing error messages

This section provides an example of modifying a customizable parser to add more complete error messages to the *userid*.SCLMERR.LISTING data set. This support can be added to all of the customizable parsers. The COBOL parser, FLMLRCBL, will be used in this example.

The **error_listing** routine is used to place the `error_string1` and `error_string2` strings into the error messages data set. `error_string1` and `error_string2` are set before invoking **error_listing**. The following list identifies, in order, the routine, the expanded English error message, and the error string to be changed in FLMLRCBL.

Routine
Change Required

initialization
Change:

```
error_string1 = miss_parm1 ' ' ||,
                miss_parm2 ' ' ||,
                miss_parm3
```

to

```
error_string1 = 'MISSING PARAMETER(S): ' ||,
                miss_parm1 ' ' ||,
                miss_parm2 ' ' ||,
                miss_parm3
```

initialization
Change:

```
error_string1 = 'LISTSIZE=',
                ||sclm_dep_array_size
error_string2 = '   LISTSIZE < ',
                DEP_ELEM_SIZE
```

to

```
error_string1 = 'LISTSIZE PARAMETER MUST BE AT LEAST',
                DEP_ELEM_SIZE
error_string2 = '
```

initialization
Change:

```
error_string1 = 'LISTSIZE=',
                ||sclm_dep_array_size
```

to

```
error_string1 = 'LISTSIZE PARAMETER MUST BE A '||,
                'POSITIVE WHOLE NUMBER'
```

initialization
Change:

```
error_string1 = 'LISTINFO=',
                ||sclm_dep_addr
```

to

```
error_string1 = 'LISTINFO PARAMETER MUST BE A '||,
                'POSITIVE WHOLE NUMBER'
```

initialization
Change:

```
error_string1 = 'STATINFO=',
                ||sclm_stats_addr
```

to

```
error_string1 = 'STATINFO PARAMETER MUST BE A '||,
                'POSITIVE WHOLE NUMBER'
```

process_line
Change:

```
error_string1 = token
```

to

```
error_string1 = 'DEPENDENCY 'token' EXCEEDS 8 '||,
                'CHARACTERS ON LINE '||,
                stats.total_lines
```

add_dep
Change:

```
error_string1 = name
```

to

```
error_string1 = 'DEPENDENCY ARRAY CAPACITY EXCEEDED '||,
                'WITH DEPENDENCY 'name
```

termination
Change:

```
error_string1 = SINGLE_QUOTE state.single_line
```

to

```
error_string1 = 'MISMATCHED SINGLE QUOTE ON ' state.single_line
```

termination
Change:

```
error_string1 = DOUBLE_QUOTE state.double_line
```

to

```
error_string1 = 'MISMATCHED DOUBLE QUOTE ON ' state.double_line
```

termination
Change:

```
error_string1 = END_KEYWORD
```

to

```
error_string1 = 'DEPENDENCY ARRAY CAPACITY EXCEEDED,'
error_string2 = 'NOT ENOUGH SPACE TO WRITE END-OF-LIST KEYWORD'
```

Appending to the error listing file

If parser errors are found, error messages are written to the *userid*.SCLMERR.LISTING data set. This data set is created (re-written) each time an error is found, each time one of the REXX parsers is invoked. The **allocate_error_listing** routine is used to allocate this data set. The overwriting of this data set is suitable

for creating or modifying members with Edit. However, during multiple migrations of members, this data set will be overwritten each time a parser error occurs per parser invocation.

To keep all parser errors for all members, modify the **allocate_error_listing** routine to append to the *userid*.SCLMERR.LISTING data set, instead of overwriting it. Change

```
IF SYSDSN(ERRFILE) = 'OK' THEN
   disp = 'OLD'
ELSE
```

to

```
IF SYSDSN(ERRFILE) = 'OK' THEN
   disp = 'MOD'
ELSE
```

With this change, all invocations of the parser will append any error messages to the error file without overwriting the previous contents.

Compiling the parsers

To increase parser performance, any parsers written in REXX can be compiled and pre-linked using the IBM Compiler and Library for REXX/370. Using the FLM@REXC language definition, SCLM can be used to compile, pre-link, and link-edit the parsers. To compile a parser using FLM@REXC:

1. Add FLM@REXC to your SCLM project definition.
2. Make any required changes to FLM@REXC, such as changing specified data set names.
3. Re-assemble and re-link the project definition.
4. Migrate the parsers into SCLM using the REXXCOM language.
5. Build each of the parsers.
6. If necessary, copy the load modules (FLMLRASM, FLMLRCBL, FLMLRC37, FLMLRCIS, FLMLRC2, FLMLRDTL, and/or FLMLRIPF) to common data sets.
7. Change the language definitions to use the load modules instead of the interpreted versions.

   Remember to change the CALLMETH parameter on the FLMTRNSL macro.
8. Re-assemble and re-link the project definition.

Part 2. Developer's Guide

© Copyright IBM Corp. 1990, 2019

Chapter 8. The Software Configuration and Library Manager

The Software Configuration and Library Manager (SCLM) component of ISPF contains the capabilities of both a Library Manager and a Configuration Manager program.

Library Manager programs control source code, keeping developers from accidentally overwriting each other's code changes and providing a mechanism for moving the source code from one set of development libraries to the next. Also, SCLM can keep back-level versions of source files, with an audit trail of changes and other basic library management functions that you can use in your code development and maintenance processes.

Configuration Manager programs track how all the pieces of an application fit together. Not just the source code, but the object and load modules as well. SCLM adds additional capabilities, such as how test cases and documentation are associated with a source code module. SCLM uses this information to control compiling, linking, and promoting an application. SCLM "builds" are optimized such that only pieces that need to be regenerated when a change is made are built.

SCLM project environment

The SCLM *project environment* is made up of data sets used by SCLM to store and control the user application software for an individual project. The project environment contains three types of data associated with an individual project:

- User Application Data
- SCLM Control Data (see “Step 6: Allocate and create the control data sets” on page 17)
- Project Definition Data (see Chapter 1, “Defining the project environment,” on page 3)

User application data

User application data consists of the application data (programs) being developed for a single project. SCLM stores all user data associated with a single project as members within a hierarchical set of MVS partitioned data sets (ISPF libraries). These data sets are called the project partitioned data sets. Users refer to SCLM-controlled ISPF libraries with an SCLM naming convention containing three levels of qualification, specifically:

```
project_name.group_name.type_name
```

The first qualifier, project_name, is the unique project identifier associated with the hierarchy.

SCLM organizes project data sets into *groups*, the second identifier within the naming convention. Each group represents a different stage or state of the user data within the life cycle of a project. For example, assume a project has three groups named DEV1, TEST, and RELEASE. The DEV1 group represents data being modified. The TEST group represents data being tested. The RELEASE group represents data released for customer use. The groups of a project are organized into hierarchical order to form a tree-like hierarchy.

A group is made up of several data sets that can contain different *types* of data. Types, the third qualifier of the naming convention, are used to differentiate the kinds of data maintained in the groups of a project. For example, source code would be stored in one type and listings in another type. It is better not to mix different data types in SCLM. (Although SCLM allows you to do this, it is not recommended; data with different formats should be stored in different types.)

© Copyright IBM Corp. 1990, 2019

Thus a user working on an application for project PROJ1 might be assigned to the DEV1 group. The project can be using four different types of data. Therefore the user might have the following project partitioned data sets to work in:

```
PROJ1.DEV1.SOURCE        - all source modules
PROJ1.DEV1.OBJECT        - all compiler object files
PROJ1.DEV1.LISTING       - all compiler listings
PROJ1.DEV1.LOAD          - all executables (link-edit output)
```

Note: SCLM can use data sets with names consisting of three levels of qualification as is the practice in many ISPF environments. It can also use data sets with two or more levels of qualification. This is an option that the project manager must enable for a project to use. If this option is used, SCLM developers would still use the `project_name.group_name.type_name` naming convention when performing SCLM functions. See “Flexible naming of project partitioned data sets” on page 12 for more information about this option.

SCLM hierarchies

The groups within a project are organized in a hierarchical order with each group being subordinate to the group above it. A sample hierarchy is shown in Figure 65 on page 140.

Figure 65. Sample Project Hierarchy

The topmost group is not subordinate to any group and is known as the top group, root group, or the root of the hierarchy. There is only one top group in each hierarchy. The bottom groups in a hierarchy are called development groups. The names for the development groups in Figure 65 on page 140 are DEV1 and DEV2. All modifications and additions to user-created data must occur in the development groups of the hierarchy. Groups of equivalent rank within the hierarchy are considered to be within the same *layer* of the hierarchy. Most hierarchies have multiple layers.

Changes can be promoted to the next group, TEST, in the example hierarchy. Promote means to copy or move a member or a set of members from one group to the next group in the hierarchy. Each group can only promote members to the group to which it is subordinate. This link between groups is known as the *promote path*. or example in Figure 65 on page 140 the three promote paths are DEV1 to TEST, DEV2 to TEST, and TEST to RELEASE. Any number of groups can promote into the same group.

Hierarchies are always searched from bottom to top along a path called the *hierarchical view*. The hierarchical view can begin at any group in the hierarchy and follows the promote paths to the topmost group in the hierarchy. Therefore in Figure 65 on page 140, two examples of hierarchical views are DEV1 to TEST to RELEASE and TEST to RELEASE. Thus, when referencing data in the hierarchy, members at lower groups take precedence over members at higher groups. All data existing in groups TEST and RELEASE is accessible from development libraries in groups DEV1 or DEV2. When a change is made to a member in the DEV1 group, this change is not available to the DEV2 group until the changed member has been promoted to the TEST group.

Therefore, within a hierarchy, the user data located at the lower layers of the hierarchy is in a more volatile state than the data at the upper layers. The upper layers of the hierarchy usually contain versions of products ready or nearly ready for release to customers, while the lower layers contain versions of products currently under development.

Key/non-key groups

You can further identify groups in the project hierarchy as *key* groups and *non-key* groups. Key groups are defined as the groups within a hierarchy that contain all the software components of the application under development. A key group is a group into which you move data during a promotion. A project can have as many key groups as you want as long as any hierarchical view has no more than 123 groups. The actual limiting factor is the MVS limit of 123 extents for a concatenated partitioned data set.

SCLM allows a project to specify transition groups between key groups. These groups are known as non-key groups. A non-key group is a group into which you copy (rather than move) data during a promotion. When you promote data in a hierarchy, SCLM does not purge data from a key group until it reaches the next key group. Therefore, in a project with non-key groups, SCLM temporarily duplicates data in the non-key groups and the next lower key group. Figure 66 on page 141 illustrates the relationship between a key and a non-key group within a project hierarchy.

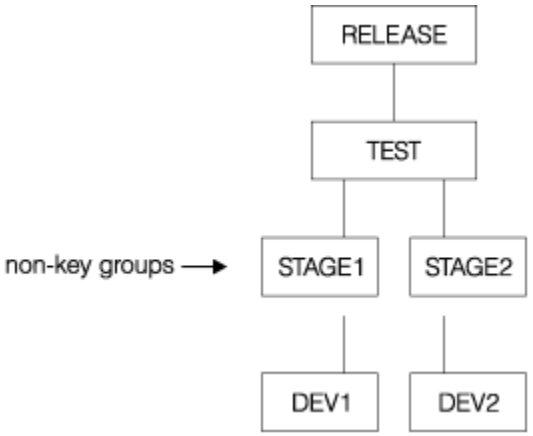


Figure 66. Key and Non-Key Groups Within the Project Hierarchy

As the figure shows, two non-key groups (the STAGE layer) appear between the development groups (the DEV layer) and the test and integration group (the TEST layer.) Developers use the STAGE groups as an interim place into which they promote their work before it moves to the next layer.

Using non-key groups enables you to display the critical elements of the hierarchical structure on ISPF panels. Because ISPF panels allow you to display only four key groups at one time, it is difficult to display the highest group in the hierarchy when you have a complex project that contains many layers.

Select key groups and non-key groups with the following set of guidelines:

- The lowest (development) groups must be key.
- Any group with more than one lower group promoting into it should be key.

Moving data through the hierarchy

Data moves within an SCLM hierarchy in two directions, up or down. When SCLM promotes members up the hierarchy from one group to the next group, the following rules apply:

- Copy members from key groups to non-key groups
- Move members from non-key groups to non-key groups
- Move members from key groups to key groups
- Move members from non-key groups to key groups and purge from the previous key group.
- Do not promote data from a primary non-key group.

In general, when SCLM accesses a hierarchy from a particular group, it concatenates only the necessary groups. If the lowest group in the hierarchy to be accessed is non-key, SCLM concatenates it with all the non-key groups above it, up to the next key group. From there, SCLM concatenates only the key groups. If

the starting group in the hierarchy to be accessed is key, SCLM concatenates only it and the key groups above it.

The one exception to this concatenation involves non-key groups that have more than one group promoting into them. Non-key groups of this kind are as significant as key groups, and SCLM must also concatenate them in a hierarchy. Groups that must be concatenated when a hierarchy is to be accessed are known as *primary groups*. Thus, all key groups and all non-key groups with more than one group promoting into them are primary groups.

After members are promoted from the development groups to the higher groups in the hierarchy, users can bring members back to the development groups by performing a *draw down*. A draw down copies the member at the higher group to the specified development group. For a member to be drawn down it must be within the *hierarchical view* of the development group. Members can only be drawn down to development groups. SCLM performs an automatic *draw down* when the member is edited.

Chapter 9. Using SCLM functions

With SCLM functions, you can view, create, update, delete, compile, link, promote, and report on data stored in the database of a project. In addition, you can generate reports with the build, promote, and utilities functions.

This chapter describes the panels and options you use to access SCLM functions and to generate reports. It also compares SCLM to ISPF and notes the differences in the EDIT environment under both utilities.

You can access all interactive SCLM functions through a set of panels by selecting the SCLM option from the ISPF Primary Option Menu. In addition to the SCLM panel interface, you can call a subset of SCLM functions independently with a command line processor or a program service interface. See Chapter 18, "Invoking the SCLM services," on page 359 for more information.

Note:

1. If SCLM does not appear on any of your menu panels or on the Menu pull-down, enter TSO SCLM on any ISPF command line. If SCLM is available to your terminal session, the SCLM Main Menu is displayed.
2. A virtual region size of 4096K is recommended when you use the SCLM dialog. Increase the virtual region size if you encounter abends related to insufficient memory.
3. SCLM maintains allocations of data sets in the hierarchy between uses of SCLM functions. This enhances the performance of SCLM; however, if data sets in the hierarchy are created, deleted, cataloged or uncataloged while SCLM is active, you should exit SCLM and reopen the SCLM Main Menu.

Name retrieval with the NRETRIEV command

The ISPF command table contains an entry named NRETRIEV. On enabled panels such as Edit, NRETRIEV retrieves the library names from the current library referral list, or data set name, z/OS UNIX file name, or workstation file name from the current data set referral list. The user is responsible for assigning the NRETRIEV command to a PF key.

When the cursor is *not* in the Other Data Set Name field, the Volume Serial field, or the Workstation File Name field, and the NRETRIEV key is pressed, the ISPF library fields are filled in from the current list. As long as the cursor is not placed in these fields, subsequent presses of the NRETRIEV key will retrieve the next library concatenation from the list.

When the cursor *is* in the Other Data Set Name field, the Volume Serial field, or the Workstation File Name field, and the NRETRIEV key is pressed, the data set name, z/OS UNIX file name, or workstation name is filled in from the current data set list. ISPF attempts to determine if the name in the list is a workstation z/OS UNIX file name, or data set name. As long as the cursor is placed in these fields, subsequent presses of the NRETRIEV key will retrieve the next data set name, z/OS UNIX file name, or workstation name from the list.

Use the personal list settings panel to force the NRETRIEV command to verify the existence of a data set before retrieving it. If verification is active, then a check is made to see if a data set name exists before a retrieval attempt. If a volume name is not in the personal list entry, then the catalog is checked to see if the data set name is cataloged. If a volume name exists, an OBTAIN macro is used to check the volume for the data set. Verification does not check ISPF library names, z/OS UNIX file names, or workstation names, and does not check for the existence of PDSE members. In the data set list Dsname Level field, verification is inactive and workstation names are never retrieved.

NRETRIEV is enabled on the following options:

- View, including extended move, copy, create, and replace panels
- Edit, including extended move, copy, create, and replace panels
- Library Utility (Option 3.1)

© Copyright IBM Corp. 1990, 2019

- Data Set Utility (Option 3.2)
- Move/Copy Utility (Option 3.3)
- Data Set List (Option 3.4)
- Reset ISPF Statistics (Option 3.5)
- Hardcopy Utility (Option 3.6)
- Workstation Transfer (Option 3.7.2)
- SuperC (Options 3.12 and 3.14)
- ISPF Table Utility (Option 3.16)
- z/OS UNIX Directory List Utility (Option 3.17)
- SCLM Options:
  - View (Option 1)
  - Edit (Option 2)
  - Member list (Option 3.1)
  - Migration (Option 3.3)
  - Unit of Work (Option 3.11)
  - Build (Option 4)
  - Promote (Option 5)
  - Easy Cmds (Option 6A)

SCLM considerations for NRETRIEV

The NRETRIEV command is enabled to work in several of the SCLM options. There are certain restrictions and considerations to keep in mind when you choose to use NRETRIEV in SCLM.

SCLM restrictions

- The NRETRIEV key within SCLM does not use the standard reference list or personal lists. Instead, it uses a stack that is stored internally. The stack is not editable. The stack is saved from session to session as a single-line table called ISRSLIST.

  Note: In the SCLM View option, the Other Data Set Name field *does* use the standard reference list because the Other Data Set Name field has no particular meaning to SCLM.
- In SCLM, there is no validation of saved or retrieved names. That means that if you type in a library name and press Enter, it is added to the list of saved names, even if SCLM does not process it. This contrasts with the standard reference list processing, which does not add a data set or library name until the data set or library is successfully allocated.
- On name retrieval (when the NRETRIEV key is pressed) there is no validation of the existence of data sets or libraries.
- The regular NRETRIEV command is screen independent (it uses a separate list indicator for each screen in split screen mode). There is only 1 position locator for SCLM lists. This means that split screens with SCLM NRETRIEV will use the same pointer into the list. An NRETRIEV on screen 1 followed by an NRETRIEV on screen 2 will get list entries 1 and 2 respectively.

Stack management for SCLM

A library name (or concatenation) is added to the list of saved library names by pressing Enter on a panel that supports saving names. If the library or concatenation exists in the list already, it is moved to the top of the list. Where the Project field or the first Group field is an output field (SCLM options 2, 3, 4, and 5), the output fields are not used in the comparison between what was typed on the panel and what is already in the list. This enables you to work in different but similar projects.

In other words, on the edit screen that has both the Project and Group1 as output fields, the concatenation:

```
SCLM Library:
   Project...: PDFTDEV
   Group ....: DGN      ....STG    ....INT    ....SVT
   Type .....: ARCHDEF
   Member ...:
```

would match:

```
SCLM Library:
   Project...: PDFTOS25
   Group ....: JSM      ....STG    ....INT    ....SVT
   Type .....: ARCHDEF
   Member ...:
```

Similarly, where groups 2, 3, and 4 are not used, those groups are not used when checking to see if the name already exists.

If a match is found, the existing entry in the list is moved to the top of the list.

SCLM main menu

Figure 67 on page 145 shows the primary options on the SCLM Main Menu.

```
  Menu  Utilities  Help
_____________________________________________________________________
                              SCLM Main Menu

Enter one of the following options:

   1  View        ISPF View or Browse data
   2  Edit        Create or change source data in SCLM databases
   3  Utilities   Perform SCLM database utility/reporting functions
   4  Build       Construct SCLM-controlled components
   5  Promote     Move components into SCLM hierarchy
   6  Command     Enter TSO or SCLM commands
   6A Easy Cmds   Easy SCLM commands via prompts
   7  Sample      Create or delete sample SCLM project
   A  SCLM Admin  Maintaining SCLM administrators
   X  Exit        Terminate SCLM

SCLM Project Control Information:
   Project . . . . PDFTDEV   (Project high-level qualifier)
   Alternate . . .           (Project definition: defaults to project)
   Group . . . . . MBURNS    (Defaults to TSO prefix)
Option ===>
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 67. SCLM Main Menu Panel (FLMDMN)

SCLM main menu options

When you select one of these options and press Enter, another panel appears that is determined by the option you selected.

View
See “View (option 1)” on page 146.

Edit
See “Edit (option 2)” on page 149.

Utilities
See “Utilities (option 3)” on page 156.

Build
See “Build (option 4)” on page 237.

Promote
See “Promote (option 5)” on page 245.

Command
Enter and execute a TSO, CLIST, REXX exec, or SCLM command from within SCLM.

Easy Cmds
Select an FLMCMD service to display a panel containing data entry fields for the parameters associated with that service. For details about the specific service panels, see the description of the relevant service in Chapter 19, “SCLM services,” on page 379.

Sample
See “Sample Project Utility (option 7)” on page 254.

SCLM Admin
See “Maintaining SCLM administrators (option A)” on page 255.

Exit
Exit from SCLM.

SCLM main menu action bar choices

Menu
See the topic about action bars in *z/OS ISPF User's Guide Vol I*.

Utilities
See the topic about action bars in *z/OS ISPF User's Guide Vol I*.

Help
Help for general and specific topics.

SCLM main menu panel fields

Project
A project's unique identifier. This field is required to access any SCLM function.

Alternate
The name of an alternate project definition to use. If this field is left blank, it defaults to the value specified in the Project field.

Group
This group defines the bottom of the hierarchical view used by the selected function, and can be any group in the hierarchy. This field defaults to your TSO PREFIX or to your user ID if no TSO PREFIX has been created. This field must be a development group if Edit (2) is chosen.

View (option 1)

The SCLM View function uses the ISPF View service with an SCLM shell around it. The View function allows you to display data in a project hierarchy or data that is not controlled by SCLM. The SCLM View interface analyzes the hierarchy structure for the project you specify and automatically provides the appropriate concatenation sequence for the groups. It presents the four lowest key groups identified in the project definition, starting from the Group specified on the Main Menu.

SCLM View is functionally equivalent to ISPF View. (See *z/OS ISPF User's Guide Vol II* for more information.) For example, you can specify a member name unless you want to see a member selection list. Additionally, you can modify the displayed library (or "group") concatenation sequence. You can also view a partitioned data set (PDS), a partitioned data set extended (PDSE), or a sequential data set. Figure 68 on page 147 shows the panel SCLM displays when you select option 1, View, from the SCLM Main Menu.

```
  Menu  RefList  RefMode  SCLM  Utilities  Workstation  Help
──────────────────────────────────────────────────────────────────────────────
                          SCLM View - Entry Panel

SCLM Library:
   Project . . . PDFTDEV
   Group . . . . MBURNS  . . . STG     . . . INT     . . . SVT
   Type  . . . . SOURCE
   Member  . . . ________       (Blank or pattern for member selection list)

Other Partitioned, Sequential or VSAM Data Set:
   Data Set Name  . . ____________________________________________________
   Volume Serial  . . ______     (If not cataloged)

Initial Macro . . . . ________         Options
Profile Name  . . . . ________         _  Confirm Cancel/Move/Replace
Format Name . . . . . ________         _  Browse Mode
                                       _  View on Workstation
                                       /  Warn on First Data Change
                                       _  Mixed Mode

Data Set Password . .            (If password protected)
Command ===> ____________________________________________________________
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 68. SCLM View - Entry Panel (FLMEB#P)

Note: The NRETRIEV command key is enabled to work with this option. See “Name retrieval with the NRETRIEV command” on page 143 for more information.

SCLM View - Entry Panel action bar choices

The action bar displays the same choices as those discussed in “SCLM main menu action bar choices” on page 146. Additional choices are:

Reflist

The Reflist pull-down menu has the following choices:

Reference Data Set List
Displays a list of up to fifteen data set names that have been entered in the "Other" Data Set Name field and other fields in ISPF that take a data set name as input.

Reference Library List
Displays a list of the last eight ISPF libraries that you have referenced.

Personal Data Set List
Displays a list of up to thirty data set names that you have created and saved.

Personal Data Set List Open...
Displays the **Open** dialog for all Personal Data Set Lists.

Personal Library List
Displays a list of up to eight ISPF Library specifications that you maintain.

Personal Library List Open...
Displays the **Open** dialog for all Personal Library Lists.

Refmode

The Refmode pull-down menu has the following choices:

List Retrieve
Sets referral lists, personal data set lists, and personal library lists into a retrieve mode. When you select an entry from the list, the information is placed into the Dsname Level field, but the Enter key is *not* simulated. You can then set other options before pressing the Enter key. (If this is the current setting, this choice is unavailable.)

List Execute
Sets referral lists, personal data set lists, and personal library lists into a retrieve mode. When you select an entry from the list, the information is placed into the Dsname Level field, and the Enter key *is* simulated. (If this is the current setting, this choice is unavailable.)

SCLM

The SCLM pull-down menu has the following choices:

Library
Displays the SCLM Library utility panel.

Sublib
Displays the SCLM Sublibrary Management Utility panel.

Migration
Displays the SCLM Migration Utility Entry panel.

DB Contents
Displays the SCLM Database Contents panel.

Architecture
Displays the SCLM Architecture Report panel.

Export
Displays the SCLM Export Utility panel.

Import
Displays the SCLM Import Utility panel.

Audit/Version
Displays the SCLM Audit and Version Utility panel.

Delete from Group
Displays the SCLM Delete from Group Utility panel.

Build
Displays the SCLM Build panel.

Promote
Displays the SCLM Promote panel.

SCLM View - Entry Panel fields

Project
The project that you specified on the SCLM Main Menu. An Alternate field also appears if you specified an alternate project definition. If you change this field, all groups in the concatenation sequence are treated as data that SCLM does not control.

Group
SCLM uses the group specified in the Group field on the SCLM Main Menu to determine the four key or primary groups in the hierarchy that initially appear on the panel. You can enter both SCLM-controlled groups and non-SCLM-controlled groups in the concatenation sequence at the same time.

If you specify a group that is defined in the project definition but not allocated, and you then request a member list, the library (LIB) members on the member list panel might not be what is expected. SCLM treats an unallocated group as if the group field were blank and ignores that group. When this situation exists, SCLM provides a panel that shows how the LIB numbers correspond to the existing groups.

Type
The identifier for the type of information in the group, such as SOURCE, ARCHDEF, or PANELS. If you change this field to a value that is **not** defined to the project definition, all the groups in the concatenation sequence are treated as data that SCLM does not control.

Member
The name of a member in an SCLM-controlled or non-SCLM-controlled partitioned data set. If you leave this field blank or type a pattern, a member list is displayed.

Data Set Name
Any fully qualified data set name, such as 'USERID.SYS1.MACLIB'. If you include your TSO user prefix (defaults to user ID), you must enclose the data set name in single quotation marks. If you omit the TSO user prefix, your TSO user prefix is added to the beginning of the data set name.

Volume Serial
A DASD volume identifier. ISPF does not allow a data set to be stored on more than one volume. SCLM does not use the system catalog when you specify a volume serial.

Initial Macro
An Edit macro to be processed before you begin viewing your sequential data set or any member of a partitioned data set. This initial macro allows you to set up a particular environment for the View session you are beginning. If you leave the Initial Macro field blank and your Edit profile includes an initial macro specification, the initial macro from your Edit profile is processed. To suppress the processing of an initial macro in your Edit profile, enter NONE in the Initial Macro field.

Profile Name
A profile name to override the default Edit profile.

Format Name
The name of a format definition or blank if no format is used. A format definition can include EBCDIC fields, DBCS fields, and a Mixed field. If the specified format includes a Mixed field definition and you specify NO in the Mixed Mode field, SCLM ignores the operation mode.

Confirm Cancel/ Move/Replace
Specifies that you want ISPF to display a confirmation panel whenever you issue a Cancel, Move, or Replace command.

Browse Mode
Specifies that you want to Browse the data set using the Browse function. This function is useful for large data sets and data sets that are formatted RECFM=U.

View on Workstation
Select this option to view the host data set member on the workstation using the workstation tool configured in the ISPF tool integrator. For more information, see the section on Workstation Tool Integration in the Settings (Option 0) chapter of the *z/OS ISPF User's Guide Vol II*. Do not select this option if you want to view the host data set member on the host using SCLM VIEW.

Warn on First Data Change
Specifies that you want ISPF to warn you that changes cannot be saved in View. The warning is displayed when the first data change is attempted.

Mixed Mode
You can browse unformatted mixed data that contains both EBCDIC (1-byte) characters and Double Byte Character Set (DBCS or 2-byte) characters. To do this, select mixed mode by entering a slash (/) next to the Mixed Mode field. If your terminal does not support DBCS, SCLM View ignores the Mixed Mode field.

Data Set Password
The password for OS password-protected data sets. This is not your TSO user ID password.

Edit (option 2)

The edit function is an interface to the ISPF editor. The SCLM editor ensures that editing occurs only in development groups. SCLM automatically locks the member when you begin the edit session.

The SCLM editor is the ISPF editor with an SCLM shell around it. If the member has changed when you end the edit session or if an explicit *SAVE* operation is performed, SCLM stores and parses the edited member and stores its accounting record. You can only edit members that are stored in data sets under the control of SCLM.

When you select the Edit option, the SCLM editor analyzes the hierarchy structure for the specified project and displays the sequence of the groups in your library concatenation. SCLM presents the four lowest key or primary groups for the project previously specified in the project definition. The SCLM lock feature,

together with the ISPF "draw down" feature, ensures that the member you want to modify is the most current version of a component in the library concatenation.

SCLM copies or draws down the member or compilation unit to your development library in the development group from its first appearance in a higher key or primary group in the library concatenation. The member or compilation unit remains locked until you delete it or promote it to a higher group.

SCLM Edit also supports editing host data sets on the workstation. SCLM Edit will draw down the member if necessary, lock it, and copy it into working storage. The data set name is converted to a workstation file name and that name is appended to the workstation's current working directory. The host data set is transferred to the workstation, and the working file is then passed to the user's chosen edit program. When the user finishes the edit session, the working file is transferred back to the host and stored in the SCLM development group. Accounting information is then saved for the member. The user will be prompted for a language if the member is new or does not have a language. For more information, see the section on Workstation Tool Integration in the Settings (Option 0) chapter of the *z/OS ISPF User's Guide Vol II*.

Figure 69 on page 150 shows the panel SCLM displays when you select Option 2, Edit, from the SCLM Main Menu.

```
  Menu  RefList  RefMode  SCLM  Utilities  Workstation  Help
──────────────────────────────────────────────────────────────────────
                          SCLM Edit - Entry Panel

SCLM Library:
   Project . . : PDFTDEV
   Group . . . . MBURNS   . . . STG_____ . . . INT_____ . . . SVT_____
   Type  . . . . SOURCE__
   Member  . . . ________        (Blank or pattern for member selection list)

Initial Macro  . . ________
Profile Name . . . ________       (If blank, defaults to data set type)

Options
_  Confirm Cancel/Move/Replace
_  Mixed Mode
_  Edit on Workstation
_  Preserve VB record length

Change code . . . . . . ________
Authorization code  . . ________  (If blank, the default auth code is used)
Parser Volume . . . . . ______    (If blank, the default volume is used)
Command ===> ____________________________________________________________
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 69. SCLM Edit - Entry Panel (FLMED#P)

Note: The NRETRIEV command key is enabled to work with this option. See “Name retrieval with the NRETRIEV command” on page 143 for more information.

SCLM Edit - Entry Panel fields

Project
The project that you specified on the SCLM Main Menu. An Alternate field also appears if you specified an alternate project.

Group
The development group that you specified in the Group field on the SCLM Main Menu. This group is followed by the next key group in the hierarchy up to four groups.

The SCLM editor ensures that editing occurs only in development groups by not allowing you to change the value of the first group field. SCLM guarantees that the group is a valid development library by verifying it against the specified project definition. (All other displayed groups are in unprotected fields and you can alter them.)

If the order of the groups is specified so that it does not match the hierarchical view for the development group, SCLM does not allow the edit session and displays the message "Invalid library order". If F1 is pressed twice, SCLM displays a panel showing all groups that comprise the hierarchical view of the development group.

If you specify a group that is defined in the project definition but not allocated, and then request a member list, the library (LIB) numbers on the member list panel might not be what is expected. SCLM treats an unallocated group as if the group field were blank and ignores that group. When this situation exists, SCLM provides a panel that shows how the LIB numbers correspond to the existing groups.

Type
The identifier for the type of information in the SCLM group, such as SOURCE, ARCHDEF, or PANELS.

Member
The name of an SCLM-controlled or non-SCLM-controlled partitioned data set member. Leaving this field blank or typing a pattern as a member name causes SCLM to display a member list.

Initial Macro
An edit macro to be processed before you begin editing. This initial macro overrides any IMACRO value in your profile.

If you leave the Initial Macro field blank and your edit profile includes an IMACRO specification, the initial macro from your edit profile is processed.

If you want to suppress the processing of an initial macro in your edit profile, enter `NONE` in the Initial Macro field. See *z/OS ISPF Edit and Edit Macros* for more information.

Profile Name
The name of an edit profile that you can use to override the default edit profile. See *z/OS ISPF Edit and Edit Macros* for more information.

Confirm Cancel/ Move/Replace
Allows you to specify whether a confirmation panel will appear for these options.

Mixed Mode
You can edit unformatted mixed data that contains both EBCDIC (1-byte) characters and Double Byte Character Set (DBCS or 2-byte) characters. To do this, you must specify Mixed Mode. When you select Mixed Mode, the editor looks for shift-out and shift-in delimiters surrounding DBCS data. If you do not select it, the editor does not accept mixed data. If your terminal does not support DBCS, SCLM Edit ignores the operation mode.

Edit on Workstation
Select this option to edit the host data set member on the workstation using the workstation editor configured in the ISPF tool integrator. For more information, see the section on Workstation Tool Integration in the Settings (Option 0) chapter of the *z/OS ISPF User's Guide Vol II*. Do not select this option if you want to edit the host data set member on the host using SCLM EDIT.

Preserve VB record length
When you select this field with a "/", it specifies that the editor store the original length of each record in variable-length data sets and when a record is saved, the original record length is used as the minimum length for the record. The minimum length can be changed using the SAVE_LENGTH edit macro command. The editor always includes a blank at the end of a line if the length of the record is zero or eight.

Change Code
Optionally, you can specify a change code to indicate why you updated the member. Change codes cannot contain commas.

Authorization Code
Optionally, you can specify a current authorization code for the member. If you do not specify an authorization code, the default authorization code is used for the member. Authorization codes cannot contain commas.

Parser Volume
The specific volume ID in which SCLM stores output from the SCLM parser. This field is not required.

Comparison of SCLM and ISPF editors

The SCLM edit function provides an interface to the ISPF editor. For example, you can specify a profile name and an initial macro before editing a member. With the SCLM editor, you can lock or parse a member, create or update an accounting record, and specify change or authorization codes. Recursive editing is only allowed within the data set concatenation currently being edited. Therefore, the member name to edit must be supplied as part of the edit command (see "EDIT command" on page 152).

The parser supplied with SCLM does not recognize ISPF packed data. If the ISPF pack mode is on, the parser supplied with SCLM returns statistical values reflecting packed data. You must unpack the data before it is parsed by SCLM to obtain correct statistical values.

When editing parts controlled by SCLM, it is important to use the SCLM editor. The ISPF editor has a configuration table that supports three levels of awareness of SCLM-controlled parts if trying to edit SCLM-controlled parts with the ISPF editor (outside of SCLM):

No awareness
ISPF edit allows SCLM members to be edited, with no warning or message.

Warning Mode
ISPF edit displays an SCLM WARNING message when editing an SCLM-controlled member. However, the ISPF edit will continue.

Fail Mode
ISPF edit does not allow the edit to start on an SCLM-controlled member.

If the ISPF editor is operating in Fail Mode, edit recovery operates in Warning Mode for purposes of the recovery; you will be able to recover the member, and the SCLM WARNING message appears.

ISPF uses two checks to determine if a member is SCLM-controlled:

- The SCLM flag for the member is on (this is set by SCLM SAVE)
- A `project.PROJDEFS.LOAD` data set exists, where the high-level qualifier of the data set being edited is equal to `project`.

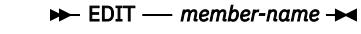
When the configuration table has Fail Mode set, only the first condition (a directory entry indicating the member was last edited by SCLM) results in the edit request being denied. If the member is not SCLM controlled by the first condition, the second condition is applied and can result in a warning message.

SCLM command macros

The following sections describe the command macros available for use with the SCLM editor.

EDIT command

The SCLM EDIT command allows a user to recursively edit a member within the same hierarchy concatenation of an SCLM supported type. That is, as long as the member exists within the groups and type specified in the Group and Type fields on the SCLM Edit - Entry panel, recursive editing is allowed.



SAVE command

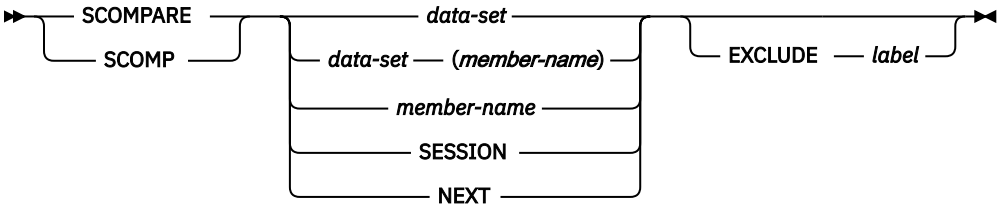
The SCLM SAVE command is similar to the ISPF Save command except that the member is automatically parsed and the accounting record of the member is created or updated.

The first time you save a member that has not been created using the SCLM editor (or migrated into SCLM), SCLM displays the SCLM Edit Profile panel (see Figure 70 on page 155) for you to specify a change code and the language of the member. The profile appears if SCLM has not been informed of the language of the member. The member is saved regardless of the parser return code on the first save.

SCOMPARE command

The SCLM SCOMPARE command is similar to the ISPF Edit COMPARE command, except SCLM decodes the data set and member (see Chapter 16, “Member encoding and decoding,” on page 333), if required, prior to it being compared. The SCLM SCOMPARE command also searches the entire SCLM hierarchy to determine the data set and member to be used instead of just searching the four groups allocated and specified on the edit entry panel.

To enable the data set and member to be compared and decoded (if required), SCLM needs to determine the fully-qualified data set name prior to invoking the ISPF edit COMPARE command to perform the compare. If the data set and member is encoded, SCLM decodes the data set and member into a temporary data set. This temporary data set is then passed to the ISPF Edit COMPARE command to perform the compare.



The data set parameter is be checked to see if the data set exists. This data set is used to perform the compare against the member being edited.

If only the *member-name* has been specified, SCLM searches the accounting records for the first occurrence of the *member-name* in the hierarchy starting at the current development group. The group where the member was found and current type are used to generate the data set name and member to be compared.

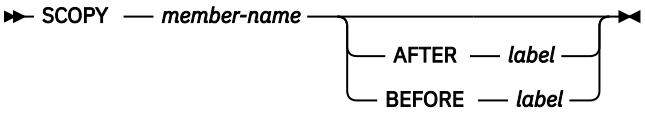
If the SESSION parameter has been specified, SCLM searches the accounting records for the first occurrence of the member currently being edited, starting at the current development group. The group where the member was found and current type are used to generate a data set name and member to be compared.

If the NEXT parameter has been specified, SCLM searches the accounting records for the first occurrence of the member currently being edited starting at the group above the current development group. The group where the member was found and current type are used to generate a data set name and member to be compared.

If the EXCLUDE parameter has been specified, the matching lines are excluded so that you only view the changes. EXCLUDE may also show a number of lines above and below the changes allowing you to see the context of the change.

SCOPY command

The SCLM SCOPY command is similar to the ISPF command except that the SCLM editor allows the copying of encoded SCLM members. The SCLM SCOPY command does not offer an extended panel for moving a member from outside the hierarchy.



The AFTER label parameter indicates the line after which to place the member that is being copied. To create an AFTER label, enter an "A" or "a" in the Line Command field (usually represented by a column of six-digit numbers on the far left side of your display) for the line you want. The BEFORE label parameter indicates the line before which to place the member that is being copied.

SCREATE command

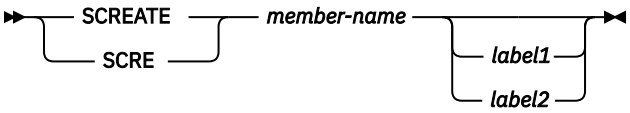
The SCLM SCREATE command is similar to the ISPF Edit CREATE command except that the SCLM editor automatically creates an accounting record for the created member, locks it out, and parses it.

If you do not enter a change code on the SCLM Edit - Entry panel (when one is required), SCLM displays the SCLM Edit Profile panel shown in Figure 70 on page 155. Also, if the language of the member you want to create differs from the language of the member you are editing, enter the SPROF command on the Edit - Entry panel. The SCLM Edit Profile panel appears so that you can specify another language. Otherwise, the newly created member has the same member attributes as the current member.

Note:

1. If the member to be created already exists in your group, SCLM returns a message indicating that the member already exists. Thus you can avoid inadvertently overwriting members.
2. If the member to be created has a language that specifies ENCODE=Y, when saving, SCLM encodes the new member. For more information, see Chapter 16, “Member encoding and decoding,” on page 333.

The SCLM SCREATE command does not offer an extended panel for creating a member outside the hierarchy.



The label parameters indicate the lines from which the new member is created. For example, assume that member OLD has been previously defined to SCLM. The COBOL programming language is associated with member OLD. If you are editing member OLD, place "copy block" (CC) commands in the Line Command field (usually represented by a six-digit number on the far left side of your edit screen) of lines two and five of member OLD, and then issue this command from the command line.

```
SCREATE NEW
```

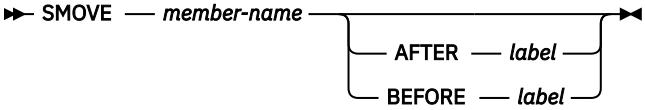
Member NEW will be added to the data set containing member OLD. Furthermore, member NEW will contain lines two through five of member OLD and will also inherit member OLD's association with COBOL. In this case, the block copy commands are the first and second labels passed with the SCREATE command.

SMOVE command

The SCLM SMOVE command is similar to the ISPF MOVE command except the SCLM editor deletes the accounting and build map information of the member being moved if it exists in the development group from which the SMOVE was issued. The SMOVE command also allows encoded members to be moved. For more information, see Chapter 16, “Member encoding and decoding,” on page 333.

The SCLM SMOVE command does not offer an extended panel for replacing a member outside the hierarchy.

Note: Once a member is successfully moved, the source member of the move is deleted. At this point, the contents of the source member only exist in the edit buffer. If you CANCEL out of the edit session where the SMOVE command was initiated without saving the changes, the data is lost.



The AFTER label parameter indicates the line after which to place the member that is being moved. To create an AFTER label, enter an "A" or "a" in the Line Command field (usually represented by a column of six-digit numbers on the far left side of your display) for the line you want.

The BEFORE label parameter indicates the line before which to place the member that is being moved. To create a BEFORE label, enter a "B" or "b" in the Line Command field for the line you want.

SPROF command

The SPROF command allows you to specify parameters that SCLM requires to track a member through the hierarchy. SCLM displays the SCLM Edit Profile panel, shown in Figure 70 on page 155, to specify a language for a new member. This panel is also displayed when you end the edit session if you did not enter a change code on the SCLM Edit - Entry panel when it is required, or if the language of the member has not yet been specified.

```
  Menu  SCLM  Utilities  Help
_____________________________________________________________

                        SCLM Edit Profile

    SCLM Library: PDFTDEV.SBURNF.SOURCE
    Member: NEWMEM

Press the Enter key with the language field blank to view a list
of valid languages or enter the desired values and press Enter.

Enter the Cancel command to exit with no change.

Language . . . . _________
Change code  . . _________  (Use "=" to retrieve last entry)
Description: . . ____________________________________________


Command ===> ____________________________________________________________
 F1=HELP      F2=          F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 70. SCLM Edit Profile (FLMEINFO)

SCLM Edit Profile Panel fields

Language
The language name to be used to process the member. This field is required and must be the same as the LANG keyword specified on the FLMLANGL macro.

Press Enter with the language field blank to select from a list of valid languages and their descriptions.

Change code
Specify a change code to indicate why you updated the member. This field is optional unless a change code verification routine is defined for the hierarchy. Change codes cannot contain commas.

Member Description
Specify a member description for use on the Utility Member List panel (FLMUSM#P) when the field "Show member Description" is selected on the SCLM Library Utility Entry panel (FLMUS#P).

You can change the information on this panel at any time during the edit session by invoking SPROF. If you alter the Language field or modify the member, SCLM parses and creates or updates the accounting record of the member when the member is saved. If you leave the language field blank or enter an invalid language, SCLM displays a selectable list of valid languages defined to the project.

SCLM processes the member and saves it in your development group if you alter the language. SCLM processes the member and saves it in your development group if you alter the change code and if the member does not exist in your development library. If you alter the change code but do not modify the member and it exists in the development group, SCLM regenerates only the accounting information.

Enter END from the SCLM Edit Profile panel to end SCLM edit profile specifications and return to the SCLM edit session. Enter CANCEL to cancel any changes you have made on the panel, end SCLM edit profile specifications, and return to the SCLM edit session.

SREPLACE command

The SCLM SREPLACE command is similar to the ISPF Edit REPLACE command except that the SCLM editor automatically parses, locks out, and creates an accounting record for the replaced member. Use this command, not SCREATE, when the member exists in the group.

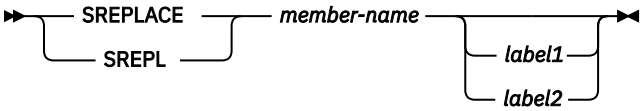
If you do not enter a change code on the SCLM Edit Entry panel (when it is required), SCLM displays the SCLM Edit Profile panel shown in Figure 70 on page 155. Also, the replaced member has the same member attributes as the current member.

If you use SREPLACE and specify a member that does not exist, SCLM calls SCREATE by default so that you can create the member.

If the member your are editing has a language that specifies ENCODE=Y, when saving the new member, SCLM will encode the member. For more information, see Chapter 16, “Member encoding and decoding,” on page 333.

The SCLM SREPLACE command does not offer an extended panel for replacing a member outside the hierarchy.

The label parameters indicate the lines from which the current member is replaced by the replaced member. The label parameters are optional.



To see an example of using commands with labels, see “SCREATE command” on page 154.

Overriding SCLM command macros

Because the SCLM editor uses ISPF edit macros to perform its functions, you should not override SCLM command macro definitions, especially the END, SAVE, CANCEL, and RETURN macros. If you need a user-defined "end" macro, use another command name such as QUIT. At the end of this alternate end macro, invoke the END, RETURN, SAVE, or CANCEL command to start the SCLM end routines.

If you override an SCLM macro by using DEFINE, the macro is not redefined until you begin a new edit session.

You can also override SCLM edit macros by entering the ISPF BUILTIN command (for example, BUILTIN SAVE).

Utilities (option 3)

Figure 71 on page 157 shows the panel SCLM displays when you select option 3, Utilities, from the SCLM Main Menu.

```
  Menu  Utilities  Help
──────────────────────────────────────────────────────────────────────────────
                              SCLM Utilities Menu

1  Library              View, browse, edit, delete, build or promote SCLM
                          controlled members and update member authorization
                          codes
2  Sublib Mgmt          Browse or delete intermediate records and forms
3  Migration            Register the contents of a library with SCLM
4  Database Contents    Create reports and tailored data sets against
                          SCLM database
5  Architecture Report  Create architecture report
6  Export               Extract SCLM accounting information
7  Import               Incorporate exported data into the hierarchy
8  Audit and Version    Display Audit and Version members
9  Delete from Group    Delete members, accounting records, build maps,
                          intermediate code and records from a group
10 Package Functions    View, delete and restore backed-up packages
11 Unit of Work         View and process Unit of Work elements
12 SCLM Explorer        Browse the relationship tables of your project

Option ===> ____________________________________________________________________
 F1=HELP      F2=          F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 71. SCLM Utilities (FLMUDU#P)

When you select one of these options, the corresponding utility is displayed.

“Library Utility” on page 157
“Migration Utility” on page 177
“Database Contents Utility” on page 179
“Architecture Report Utility” on page 188
“Export Utility” on page 195
“Import Utility” on page 199
“Audit and Version Utility” on page 203
“Delete from Group Utility” on page 213
“Package Backout Utility” on page 216
“Unit of Work Utility” on page 224
“SCLM Explorer” on page 230

Library Utility

The library utility allows you to browse accounting records, members, and build map records. In addition, you can use this utility to delete members and their accounting and build map data, view, edit, build and promote members, and update a member's authorization codes.

The library utility is completely interactive and parallels the ISPF library utility.

Figure 72 on page 158 shows the SCLM panel that appears when you select Option 1, Library, from the SCLM Utilities panel.

```
  Menu  SCLM  Utilities  Help
_____________________________________________________________________________
FLMUS#P              SCLM Library Utility - Entry Panel
Option ===> _____________________________________________________________________

blank Display member list    E Edit member            T Transfer owner
    A Browse account info    V View member            N NOPROM processing
    M Browse build map       C Build member           W Where used
    B Browse member          P Promote member
    D Delete member info     U Update auth code

SCLM Library:
  Project  . : SLMTEST6
  Group  . . . DEV1
  Type . . . . SOURCE
  Member . . . ________    (Blank or pattern for member selection list)

Select and rank member list data  . . AT   (T=TEXT, A=ACCT, M=BMAP, S=SUBP)

Enter "/" to select option
/  Hierarchy view                       Process . . 3  1. Execute
/  Confirm delete                                      2. Submit
/  View processing options for Edit                    3. View options
_  Show Member Description

 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F11=NRETRIEV F12=Cancel
```

Figure 72. SCLM Library Utility (FLMUS#P)

The fields on the SCLM Library Utility panel are:

Project
The project that you specified on the SCLM Main Menu. An additional field called Alternate is displayed if you specified an alternate project definition. You cannot change the Project or the Alternate fields on this panel.

Group
The group that you specified in the Group field on the SCLM Main Menu. The group field can be modified to specify other groups defined to the project.

Type
The identifier for the type of information in the ISPF library.

Member
The name of an SCLM library member. You can display a member list by leaving the Command field and the Member field blank or by leaving the Command field blank and entering a pattern as the member name. See "Specifying selection criteria" on page 181 for details. Valid pattern characters are the asterisk (*) and the logical NOT symbol (¬).

Select and rank member list data

A one, two, or three character string that indicates the kind of information that appears on the member list panel. You can specify strings composed of the following characters:

T, to display text data; A, to display accounting data; and M, to display build map data.

Each character can only be used once. The order of the characters determines the order of the data on the member list. This option limits the type of data that appears with each member on the list, and only members that have the types of data specified will appear. For example, a member that only has text will not appear if the string AM is specified. All types of data that exist for a member at a particular level are subject to processing by library utility commands.

If only two types of data are specified and one of those is A (accounting), the language associated with the member will also be displayed. If only A is specified, both the language and authorization code will be displayed.

Hierarchy view
Selects as input the library entered on the panel, as well as all the libraries in its hierarchy view. The hierarchy is searched from the bottom up for the first occurrence of the specified member. If you do not select "Hierarchy view", only the library entered on the panel is used as input. This option is valid with all Library Utility - Entry panel or member list commands except delete, which defaults to a NO value.

Confirm delete
Allows you to specify whether you want a confirmation panel to appear when attempting to delete objects (text, accounting information, or build map information) with the SCLM library utility. If you select this field, the Confirm Delete panel appears every time you request a delete. In addition to confirming the delete request, this panel enables you to choose which information you want to delete for the member. If you do not select this field, the Confirm Delete panel does not appear for deletions and all data is deleted without any additional user interaction.

View processing options for Edit
Allows you to indicate whether you want to verify or update edit processing options or allow them to default to the values that last appeared on the Edit Data Entry panel. When you select this option, the SCLM Edit Data Entry panel displays so that you can verify or update edit processing options. If you do not select it, Edit options default to those values that last appeared on the Edit Data Entry panel. The panel does not appear.

Show member description
Allows you to display the member list panel FLMUSM#P, which contains an extra line displaying the description associated with a member. The Description is entered via SPROF command.

Process
Allows you to specify the processing mode for the Build or Promote commands. The value of the "Process" field is unique to the library utility. You will not be carried to or from the "Process" field on any other SCLM panel.

Execute
Invokes SCLM Build or Promote in the foreground. The Build or Promote options default to those values that last appeared on the Build or Promote Data Entry panel. The panel does not appear.

Submit
Invokes SCLM Build or Promote in the background. The Build or Promote options default to those values that last appeared on the Build or Promote Data Entry panel. The panel does not appear.

View options
Displays the SCLM Build or Promote Data Entry panel so that you can verify or update Build or Promote processing options before execution.

Note: The value for "Confirm delete" is reset each time the library utility is entered. The values for "Select and rank member list data", "Process", "Hierarchy view", and "View processing options for Edit", are kept from session to session until you change them.

Library Utility commands

Type your selection in the Command field.

A, B, or M
SCLM displays the specified member or record if it is present.

While in Browse, all Browse commands are supported. Note that although a hierarchy view may be specified, the Library Utility only allocates the data set containing the existing version of the requested member. The Browse command executed from within View can only operate on members within the allocated data set.

V
SCLM displays the specified member if it is present.

D

SCLM deletes member data such as text, accounting, and build map records. When Confirm Delete has been selected on the Library Utility panel, you can choose which information to delete for the member (text, accounting information, and/or build map information). Otherwise, all information for the member is deleted. Delete is only allowed at the group specified on the Library Utility panel.

If you delete a member from a key group that also exists in a non-key group in a higher layer of the hierarchy, you must delete the member from the non-key group manually.

E

The SCLM Editor is invoked for the member specified in the Member field. A development group must be specified in the Group field. Once in the SCLM Editor, all Edit commands are supported. The library utility allocates the first four key groups for a project. If the member exists at a higher group, the group containing the member will be allocated, replacing the original fourth allocated group. The COPY, MOVE, and EDIT commands can only operate on members within the allocated data sets. The use of COPY or MOVE from within an Edit session invoked from the utility is not recommended.

C

SCLM Build is performed on the specified member.

N

SCLM modifies the "account status" on the accounting record to determine if the editable member is promotable or not. This option is only available on members with an accounting record and an "accounting status" of:

- EDITABLE
- NOPROM-N,or
- NOPROM-R

For more information on this option, see Chapter 15, “Leaving a Member Behind on Promotion,” on page 319.

P

SCLM Promote is performed on the specified member.

U

SCLM displays an input panel and updates the authorization code according to your input. Update is only allowed at the group specified on the Library Utility panel. (To delete or update any data, you must have at least UPDATE authority to the specified data set.) Any value entered in the "New authorization code" field on the input panel remains there until it is changed by the user or the library utility is exited and entered again. There is a brief period during which changes made to a member's authorization code by another session or user will not be recognized. If you receive an unexpected error message while updating a member's authorization code, use the browse accounting record command to check the member's current authorization code. If the authorization code needs to be updated, try the update authorization code command again.

T

SCLM modifies the "Change user ID" field on the accounting record to transfer ownership of the member to another user. This allows the new owner to modify the member. This option is available if the following conditions are true:

- member level locking is enabled
- The user who is accessing the option matches the "Change user ID" in the accounting record or is defined as an SCLM administrator
- The accounting record exists in a development group

W

Invokes the Where-used function. SCLM displays a list of components which include the selected component. Library utility commands (except W) may then be issued against the list members.

To perform commands against several members at once, use the member selection list.

Member selection list

You can browse, view, delete, build, promote, display Where-used information, or update the authorization code for members by making selections from a member selection list. To display a member selection list, perform the following steps:

1. Leave the Command field blank.
2. Type the group and type information in the appropriate fields. The Project field contains the project you specified on the SCLM Main Menu. You cannot change this field here.
3. Leave the Member field blank or enter a pattern.
4. Choose the data to appear and the order to display it on the member list panel by entering a string in the "Select and rank member list data" field.
5. Indicate whether you want a hierarchy view by entering a slash (/) in the "Hierarchy view" field.
6. Press Enter.

Note: The NRETRIEV command key is enabled to work with this option. See “Name retrieval with the NRETRIEV command” on page 143 for more information.

Figure 73 on page 161 shows the panel SCLM displays when you complete the instructions for displaying a member list. This display contains text, accounting, and build map data, indicating that the string "TAM" was entered for the "Select and rank member list data" field. Use the scroll commands or the LOCATE command to scroll the list.

```
  Menu  SCLM  Functions  Utilities  Test  Help
──────────────────────────────────────────────────────────────────────────
FLMUSL#P st : SLMTEST6.DEV1.SOURCE - HIERARCHY VIEW -          Member 1 of 23
Command ===> ____________________________________________ Scroll ===> CSR

A=Account  M=Map      B=Browse  D=Delete    E=Edit     V=View
C=Build    P=Promote  U=Update  T=Transfer  N=Noprom   W=WhereUsed

_   Member   Status      Account  Language   Text       Chg Date   Chg Time
_   AAAA                DEV1     TXT2       DEV1       2002/08/02 13:31:12
_   CPYRITE             DEV1     DTL        DEV1       2002/01/21 13:08:15
_   DDDDD                                   DEV1       2002/06/27 10:43:30
_   DTL2                DEV1     DTL        DEV1       2002/01/21 13:08:04
_   FLM01EQU            DEV1     HLAS       DEV1       2007/01/29 12:07:33
_   FLM01MD1            DEV1     HLAS       DEV1       2002/02/14 12:24:05
_   FLM01MD2            DEV1     PLIO       DEV1       2002/02/14 12:24:10
_   FLM01MD3            TEST     HLAS       TEST       2002/02/14 12:23:52
_   FLM01MD4            RELEASE  HLAS       RELEASE    2001/10/30 16:58:57
_   FLM01MD5            RELEASE  HLAS       RELEASE    2001/10/30 16:58:57
_   FLM01MD6            DEV1     HLAS       DEV1       2002/01/22 13:06:08
_   HANK                DEV1     TEXT       DEV1       2002/05/24 10:26:00
_   HANK2               DEV1     TEXT       DEV1       2002/04/17 11:04:40
_   HANK3               DEV1     TEXT       DEV1       2002/06/27 12:57:47
_   PMR60436            DEV1     COB2       DEV1       2002/01/31 12:18:10
_   TESTS1              DEV1     PLIO       DEV1       2002/05/06 11:28:22
_   TESTS2              DEV1     PLIO       DEV1       2002/02/15 12:01:11
_   TEST6               TEST     TEXT       TEST       2002/06/17 14:11:05
_   TSTPETE1            DEV1     TEXT       DEV1       2002/04/15 10:19:36
_   VRCPTD1             DEV1     DTL        DEV1       2002/01/21 13:08:24
_   Z1                  DEV1     DTL        DEV1       2002/01/21 17:48:57
_   Z2L                 DEV1     DTL        DEV1       2002/01/21 17:49:14
_   Z300103             DEV1     DTL        DEV1       2002/01/08 17:50:09

 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F11=NRETRIEV F12=Cancel
```

Figure 73. Member Selection List (FLMUSL#P)

Another way to view a member list is shown in Figure 74 on page 162. In this example, the string "AT" was specified for the "Select and rank member list data" field, causing accounting and text data, in that order, to appear on the member list panel. Also note that a hierarchy view with the member description was requested for this member list.

```
  Menu  SCLM  Functions  Utilities  Help
_____________________________________________________________________________
FLMUSM#P st : SLMTEST6.DEV1.SOURCE - HIERARCHY VIEW -          Member 1 of 23
Command ===> _____________________________________________ Scroll ===> CSR

A=Account  M=Map       B=Browse   D=Delete    E=Edit    V=View
C=Build    P=Promote   U=Update   T=Transfer  N=Noprom  W=WhereUsed

    Member   Status      Account  Language   Text       Chg Date   Chg Time

_   AAAA                 DEV1     TXT2       DEV1       2002/08/02 13:31:12

_   CPYRITE              DEV1     DTL        DEV1       2002/01/21 13:08:15
    copywrite copy book
_   DDDDD                                    DEV1       2002/06/27 10:43:30

_   DTL2                 DEV1     DTL        DEV1       2002/01/21 13:08:04
    DTL source for panel TTMENU
_   FLM01EQU             DEV1     HLAS       DEV1       2007/01/29 12:07:33
    Assembler copybook - Register equates
_   FLM01MD1             DEV1     HLAS       DEV1       2002/02/14 12:24:05
    Routine Initialization
_   FLM01MD2             DEV1     PLIO       DEV1       2002/02/14 12:24:10
    Routine Prolog
_   FLM01MD3             TEST     HLAS       TEST       2002/02/14 12:23:52

_   FLM01MD4             RELEASE  HLAS       RELEASE    2001/10/30 16:58:57

_   FLM01MD5             RELEASE  HLAS       RELEASE    2001/10/30 16:58:57

_   FLM01MD6             DEV1     HLAS       DEV1       2002/01/22 13:06:08

 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 74. Member Selection List with Hierarchy and Member Description View (FLMUSM#P)

The fields that appear on the SCLM Member Selection List panel are:

Member
The names of the members fitting the criteria you specified on the SCLM Library Utility - Entry panel.

Status
SCLM displays the status of the member according to the line command you select. The status field indicates the action that was taken for the selected member. For example, a status of *EDITED will appear next to any member for which the 'E' command is selected, even if the member is not saved. The status for delete indicates the group at which the delete occurred. The status displayed for each command is shown in the following example:

```
   A   Display an accounting record       *BRACCT
   B   Browse a member                    *BRTEXT
   C   Build a member                     *BUILT
   D   Delete a member                    *D-GROUP1
   E   Edit a member                      *EDITED
   M   Display a build map record         *BRBMAP
   N   Change Promote Processing          *NOPROM
   P   Promote a member                   *PROMOTED
   T   Transfer ownership                 *TRANSFRD
   U   Update an authorization code       *UPDATED
```

```
   V   View a member                      *VIEWED
   W   Display Where-used Info            *WHEREUSE
```

When an error occurs or the member name is changed on the edit or Build Data Entry panel, the status for the member will be blank.

Account
A group name in this field indicates that the accounting information for the associated member exists.

Language
The language of the member appears in this column when accounting data is requested and when space permits.

Text
A group name in this field indicates that the member exists.

Chg Date
The value of this field depends on the type of data requested for display. When text data is requested, this field contains the last change date for the member from the PDS directory. If accounting data is requested but text is not, this field contains the change date from the accounting record. If only build map data is requested, the change date from the build map appears.

Chg Time
The value of this field depends on the type of data requested for display. When text data is requested, this field contains the last change time for the member from the PDS directory. If accounting data is requested but text is not, this field contains the change time from the accounting record. If only build map data is requested, the change time from the build map appears.

Bld Map
A group name in this field indicates that the build map record for the associated member exists.

Authcode
The current authorization code for the member appears in this column when accounting data is requested and when space permits.

The following primary commands are valid on the Member Selection List:

SORT
The SORT command sorts the member list by any field displayed on the member list, except the line command field and Status field. The field names are the column headings.

REFRESH
The REFRESH command, which can also be entered as REF, refreshes the member list, adding new members, removing those that have been deleted, and updating the information displayed for each member. It also resets the line command field and Status field and sorts the member list again by member name. Any pending line commands are processed before the REFRESH command.

HIER
The HIER command is used to reset the Hierarchy View value specified on the Library Utility panel from the member list. Syntax is as follows:

```
HIER ON|OFF
```

HIER OFF displays only those members found in the group specified on the Library Utility panel. HIER ON displays the first occurrence of a member found in the specified group or any higher group within the view of the project hierarchy.

LOCATE
The LOCATE command scrolls the list to the requested member.

UP
Scrolls up.

DOWN
Scrolls down.

All of the Library Utility line commands can also be entered as primary commands from the member list command line. The syntax for the primary commands is:

```
command member
```

where *command* is the 1-character command and *member* is the member against which the command is to be performed. The Edit (E) primary command can be used to edit a new member. At the end of the edit session, the new member will be added to the list in sorted order.

Accounting record

If you enter the A line command to display an accounting record, SCLM displays a panel showing the information recorded for the member as shown in Figure 75 on page 164.

```
          PDFTDEV.SVT.EXEC(FLMEBLD): Accounting Record

                                                                More:      +
  Physical Data Set . : PDFTDEV.SVT.EXEC
  Accounting Status . : EDITABLE        Change Group . . . . : MOS
  Change User ID  . . : P020136         Authorization Code . : BASE
  Member Version  . . : 2               Auth. Code Change  . :
  Language  . . . . . : REXX            Translator Version . :
  Creation Date . . . : 1997/06/26      Change Date  . . . . : 1997/06/30
  Creation Time . . . : 16:55:02        Change Time  . . . . : 10:09:00
  Promote User ID . . : PDFTOOL         Access Key . . . . . :
  Promote Date  . . . : 1997/07/14      Build Map Name . . . :
  Promote Time  . . . : 19:02:40        Build Map Type . . . :
  Predecessor Date  . : 0000/00/00      Build Map Date . . . : 1997/06/30
  Predecessor Time  . : 00:00:00        Build Map Time . . . : 10:09:00

 Enter "/" to select option
 _ Display Statistics
 _ Number of Change Codes      : 0
 _ Number of Includes          : 0
 _ Number of Compilation Units : 0
Command ===> ____________________________________________________________
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F12=Cancel
```

Figure 75. Accounting Record (FLMUSA#P)

The display fields on the Accounting Record panel cannot be modified.

Use a slash (/) to select an option and press Enter to display additional panels. You can browse the statistics or lists of change codes, includes, compilation units, or user entries referenced by a member. You can also scroll the lists.

Physical Data Set
The physical data set in which the SCLM-controlled member actually resides. SCLM allows you to define project data sets that don't have conventional SCLM data set names by providing SCLM aliases for them. When this is the case, the name appearing on the panel title is the SCLM alias for the actual data set in the "Physical Data Set" field.

Accounting Status
The status of the member.

EDITABLE
Members that you can edit

NON-EDIT
Members that SCLM creates as a result of build processing

LOCKOUT
Members that are locked at the development group in which they exist but have not been parsed. You can use the SCLM Editor or Migration Utility to change the status of these members to EDITABLE before attempting to build or promote them.

INITIAL
Members for which a lock has been requested. This status generally appears while a member is being edited. When the edit is complete, the status changes to EDITABLE.

NOPROM-N
Members that you can edit which, on promotion, are not copied to the next level. The build maps containing this member are not rebuilt, even if the associated language specifies an FLMLRBLD macro which would cause a rebuild.

NOPROM-R
Members that you can edit. On promotion the member and the build maps containing this member are not copied to the next level. Once the copy phase is complete, SCLM invokes a build to rebuild these build maps using the version of the member at the next level or above.

Change User ID
The user ID of the person who made the last update to the member.

Member Version
The number of times that an EDITABLE member was drawn down. The member version is also updated whenever the language of the member is changed. For a NON-EDIT member, such as OBJ, it is the number of times that the member was generated by SCLM. New members use a version of 1.

Language
The language of the member.

Creation Date
The date the member was first registered with SCLM.

Creation Time
The time the member was first registered with SCLM.

Promote User ID
The user ID of the person who last promoted the member.

Promote Date
The date the member was last promoted.

Promote Time
The time the member was last promoted.

Predecessor Date
The change date of the member that this member overlays when it is promoted up the hierarchy.

Predecessor Time
The change time of the member that this member overlays when it is promoted up the hierarchy.

Change Group
The name of the group in which the member was last updated.

Authorization Code
The current authorization code for the member.

Auth. Code Change
A nonblank value indicates that SCLM is attempting to update the Authorization Code for this member. If the update completes successfully, the value of this field becomes the new authorization code of the member.

Translator Version
The version of the translator used during build processing.

Change Date
The last date a developer modified the member.

Change Time
The last time a developer modified the member.

Access Key
An identifier used to restrict access to a member.

Build Map Name
For NON-EDIT members, this field specifies the name of the build map that was created when the NON-EDIT member was created. For EDITABLE members, this field is blank.

Build Map Type
For NON-EDIT members, this field specifies the type of the build map that was created when the NON-EDIT member was created. For EDITABLE members, this field is blank.

Build Map Date
The date used by SCLM to determine if the member has changed since the last build. For EDITABLE members, this field is usually the same as the Change Date field. When the Change Date field is updated, the Build Map Date field is updated. For NON-EDIT members, this field is the date of the last build of the member.

Build Map Time
The time used by SCLM to determine if the member has changed since the last build. For EDITABLE members, this field is usually the same as the Change Time field. When the Change Time field is updated, the Build Map Time field is updated. For NON-EDIT members, this field is the time of the last build of the member.

Display Statistics
SCLM displays the Accounting Record Statistics panel, shown in Figure 76 on page 167.

Number of Change Codes
The number of change codes entered against the member. See Figure 77 on page 168.

Number of Includes
The number of include references in the source member. See Figure 78 on page 169.

Number of User Entries
The number of user data entry records associated with the member.

Statistics

SCLM displays statistical information, as shown in Figure 76 on page 167, when you enter a "/" in the Display Statistics field on the Accounting Record panel. These statistics are parser-dependent.

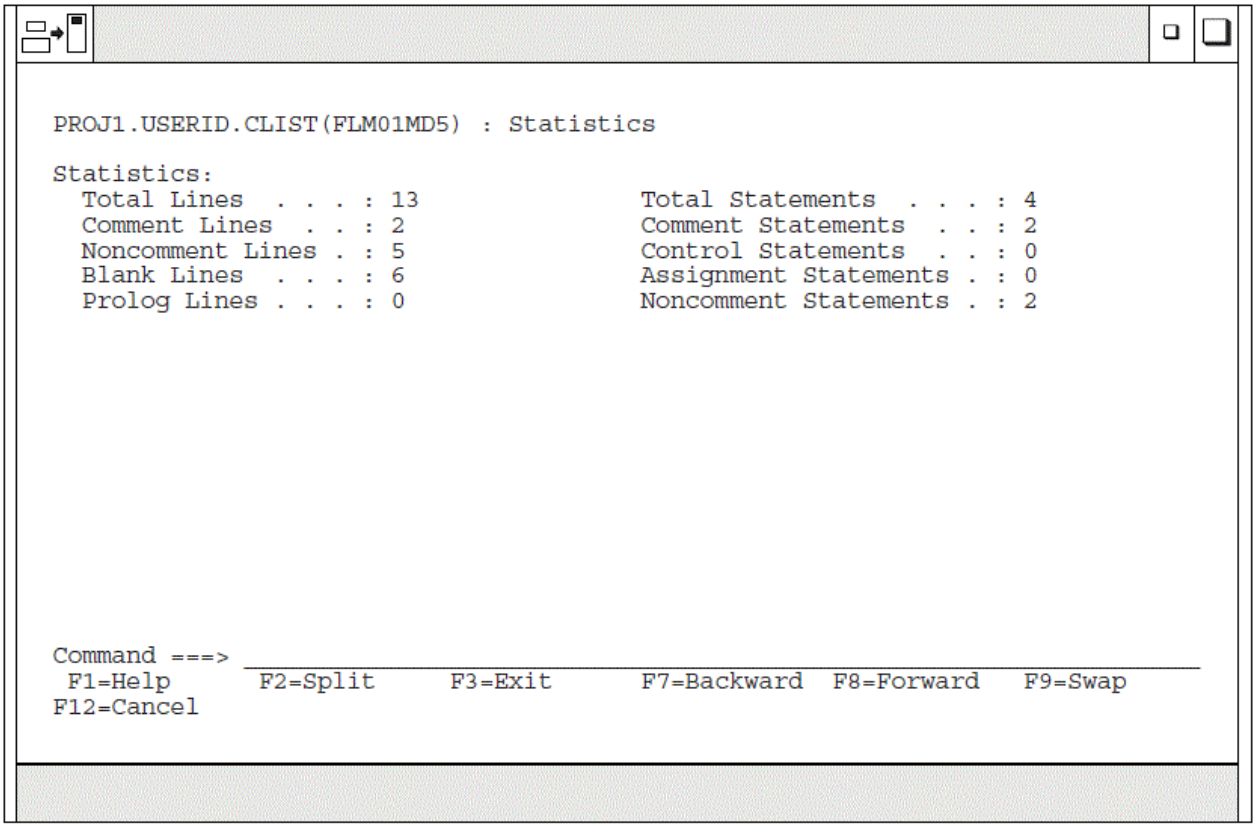


Figure 76. Accounting Record Statistics (FLMUSS#P)

The fields on the Accounting Record Statistics panel are:

Total Lines
The total number of lines in the member, which is equal to the sum of comment lines, noncomment lines, and blank lines.

Comment Lines
The number of comment lines. A comment line is any line that has comment information only. If a line has both a statement and a comment, SCLM considers it a noncomment line.

Noncomment Lines
The number of source lines. A noncomment line is a source line that contains at least part of a noncomment statement. If a line has both a statement and a comment, SCLM considers it a noncomment line.

Blank Lines
The number of blank lines in the member. A blank line is language-independent; no nonblank characters can be on it.

These statistics are parser-dependent.

Prolog Lines
The number of prolog lines in the member.

Total Statements
The sum of the comment statements and the noncomment statements in the member.

Comment Statements
The number of comment statements. A comment statement is denoted by a set of beginning and ending comment delimiters for the particular language being parsed. If an ending delimiter is not defined for a language, the end of the line is used. A comment statement can span several lines, or several comment statements can exist on a single line.

Control Statements
The number of logical control statements.

Assignment Statements
The number of assignment statements.

Noncomment Statements
The number of complete statements that SCLM can process. Noncomment statements are language-dependent, follow language syntax rules, and are separated by the language delimiter. A noncomment statement can span several lines, or several noncomment statements can exist on a single line.

Note: The parser that is invoked for the member determines the field values. The definitions apply for ISPF-supplied parsers.

Change code list

Figure 77 on page 168 is an example of the information SCLM displays when you enter a "/" in the "Number of Change Codes" field on the Accounting Record panel. If you are not allowed to delete the records you specify, the line command field is hidden and only the Change Code, Change Date, and Change Time are displayed.

```
         PDFTDEV.MOS.SOURCE(PROG01): Change Code List           Member 1 of 2

     Line Command:      D - Delete change code
     Enter Cancel command to exit without processing selections

Delete  Status  Change Code      Change Date    Change Time
------  ------  -----------      -----------    -----------
                CC02             2000/02/04     13:41:00
                CC01             2000/02/04     13:40:43
****************************** Bottom of Data ******************************









Command ===> _____________________________________________ SCROLL ===> PAGE
 F1=Help      F3=Exit     F12=Cancel
```

Figure 77. Change Code List - Records That Can Be Deleted (FLMUSC#P)

The fields on the Change Code List panel are:

Delete
You specify that you want to delete the change code when you enter D in this field. SCLM selects the change code for deletion.

Status
SCLM displays *SELECT to indicate the change code you selected. Enter the END command to confirm the delete request.

Change Code
A value assigned to indicate why a member was updated.

Change Date
The last date a developer modified the member for the associated change code. The Change Date on the top of the list is the most recent.

Change Time
The last time a developer modified the member; it is associated with the Change Date.

Note: If you alter the change code, but do not modify the member, the change date and time of the change code will be the same as the date the member was last modified.

Include list

Figure 78 on page 169 is an example of the information SCLM displays when you enter a "/" in the "Number of Includes" field on the Accounting Record panel.

Figure 78. Include List (FLMUSI#P)

The fields on the Include List panel are:

Include
: The name of an include reference in the source member. An include reference is a generic term for code that SCLM inserts when it compiles the source member. The syntax of an include statement in a program is language-dependent and is defined by language syntax rules.

Include set
: The include-set name is used to associate an include with the types in the hierarchy where that include can be found. The include-set name is returned by the parser. A blank name indicates that the include is associated with the default include set.

User data entries

Figure 79 on page 170 is an example of the information SCLM displays when you enter a / in the "Number of User Entries" field on the Accounting Record panel.

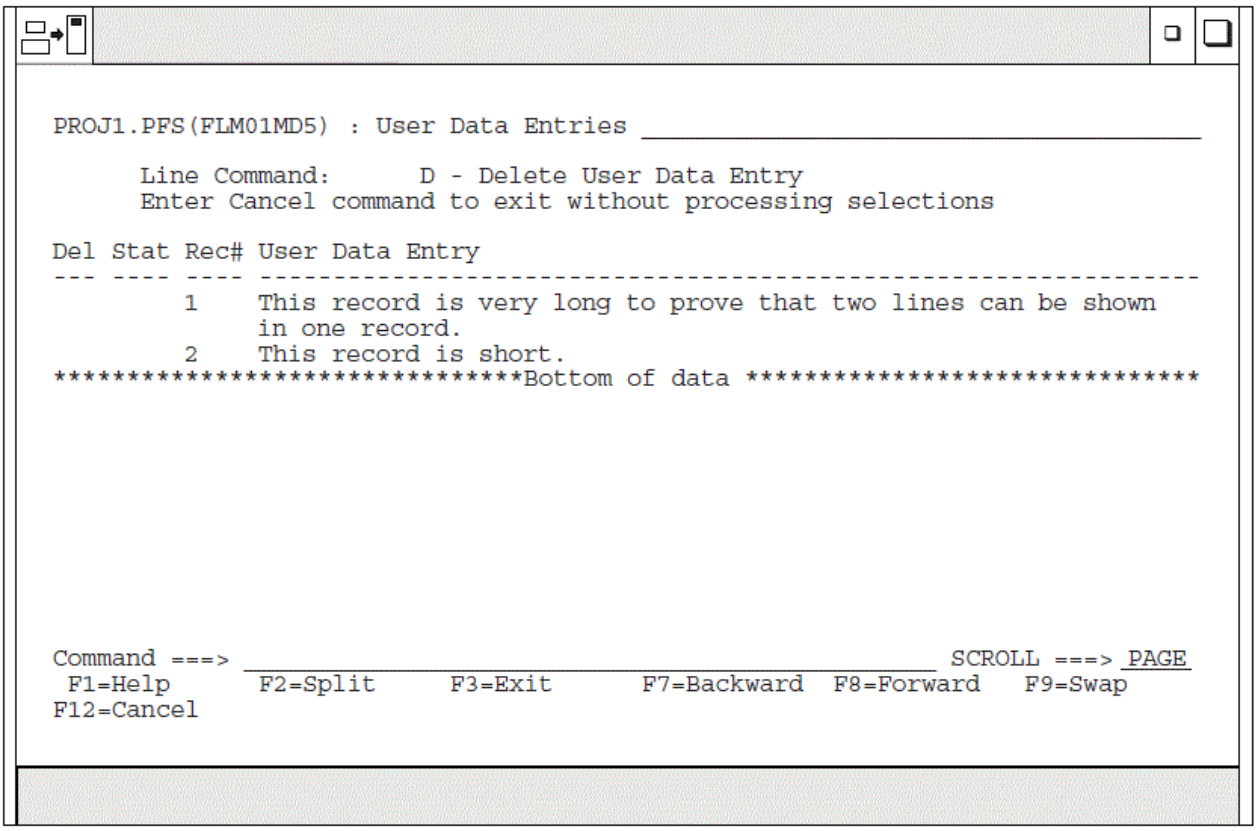


Figure 79. User Data Entries (FLMUSE#P)

The fields on the User Data Entries panel are:

Del
You specify that you want to delete the user data entry record when you select D in this field.

Stat
SCLM displays *SEL to indicate the user data entry record you selected. Enter the END command to confirm the delete request.

Rec#
SCLM displays a record number with the first line of each user data entry record.

User Data Entry
Project-specific information entered into the accounting record by the SAVE service. The user data entry record can span two lines for a maximum of 128 characters.

Build map record

Enter the M line command on the SCLM Library Utility panel or on the member selection list to display a build map record. The Build Map Record panel, shown in Figure 80 on page 171, displays the fixed build map information SCLM records for a member.

```
          PDFTDEV.SVT.SOURCE(ISRSVCBD): Build Map Record

General data:
  Change User ID . : P020136             Change Group . . : MOS
  Member Version . : 117                 Change Date  . . : 2000/01/10
  Language . . . . : CCMAP               Change Time  . . : 21:51:58
  Creation Date  . : 1997/10/14          Promote Date . . : 2000/01/21
  Creation Time  . : 17:18:43            Promote Time . . : 21:27:17
                                         Promote User ID. : PDFTOOL

  Language Version . . : PLX240          Build Map Date . : 2000/01/10
  Build Map Name . . . : ISRSVCBD        Build Map Time . : 21:51:58
  Build Map Type . . . : SOURCE


Enter "/" to select option
_ Review Build Map Contents




Command ===> _____________________________________________________________
 F1=Help      F3=Exit     F12=Cancel
```

Figure 80. Build Map Record (FLMUSB#P)

The fields on the Build Map Record panel are:

Change User ID
The user ID of the person who made the last update to the member.

Member Version
The number of times that the build map has been generated by SCLM. The first time a build map is generated a version of 1 is used.

Language
The language of the build member. This language is determined by SCLM Build; it is not specified by the user or the project manager.

Creation Date
The date the build map was first created.

Creation Time
The time the build map was first created.

Change Group
The name of the group in which the member was last updated.

Change Date
The last date the member was modified.

Change Time
The last time the member was modified.

Promote Date
The date the member was last promoted.

Promote Time
The time the member was last promoted.

Promote User ID
The user ID of the person who last promoted the member.

Translator Version
The version of the translator used during build processing.

Language Version
The version of the language that SCLM uses in language-based builds.

Build Map Name
The name of the member with which the build map is associated.

Build Map Type
The type of the member with which the build map is associated.

Build Map Date
The date of the build that created the build map.

Build Map Time
The time of the build that created the build map.

Review Build Map Contents
SCLM displays the Build Map Contents panel, shown in Figure 81 on page 172, when you select this field.

Build map contents

When you enter a / in the Review Build Map Contents field, SCLM displays the build map contents in a browse data set, as shown in Figure 81 on page 172. The data set shows the contents of a build map record for an architecture defined in a CC architecture member.

```
 BROWSE    PDFTDEV.SVT.SOURCE(ISRSVCBD): Build Map Contents     Line 00000000
******************************** Top of Data ********************************
                              Build Map Contents
                              ------------------

Keyword  Member                                Type     Last Time Modified  Ver
-------- ------------------------------------ -------- ------------------- ---
SINC     ISRSVCBD                              SOURCE   2000/01/10 21:39:17 85
OBJ      ISRSVCBD                              OBJ      2000/01/10 21:51:58 514
I1*      SPFPROC                               SOURCE   1999/10/04 19:01:00 12
I1*      DCLCMLST                              SOURCE   1999/01/11 14:33:00 2
I1*      DCLSCFIG                              SOURCE   2000/01/10 21:13:32 75
I1*      DCLSSYS                               SOURCE   1995/05/11 11:24:00 4
I2*      DCLSTLDX                              SOURCE   1995/05/11 11:25:00 6
I1*      DCLSTLD                               SOURCE   2000/01/10 21:14:54 58
I1*      DCLSTFD                               SOURCE   2000/01/10 21:14:46 30
I3*      SPFTSCN                               SOURCE   1989/02/10 15:48:00 1
I2*      SPFTSC                                SOURCE   1999/06/23 13:08:00 21
I1*      DCLSTSC                               SOURCE   1994/01/21 14:52:00 2
I3*      SPFTSPN                               SOURCE   1994/03/02 15:54:00 1
I2*      SPFTSP                                SOURCE   1999/12/09 14:19:09 41
I1*      DCLSTSP                               SOURCE   1993/01/27 16:22:00 4
Command ===> _____________________________________________  Scroll ===> PAGE
 F1=Help      F3=Exit   F5=Rfind  F12=Cancel
```

Figure 81. Build Map Contents (FLMUSBRP)

The fields on the Build Map Contents panel are:

Keyword
You can use certain keywords to identify architecture information. See “Architecture statements” on page 277 for more details. The internal build map keywords, denoted with an asterisk, are described as follows.

The architecture member example contains two keywords: OBJ, and LIST. If a keyword is denoted with an asterisk (*), it includes references found in source member FLM01MD5.

Member
The name of the member referenced in the architecture member.

Type
The name of the type containing the member.

Last Time Modified
For an EDITABLE member, this field is the last time SCLM parsed and stored the specified member. For SCLM-generated (NON-EDIT) members, such as OBJ and LIST, this field is the last time SCLM generated the member.

Internal Keywords

Keywords that SCLM uses to track references. The internal keyword I# indicates the group in which the members were first referenced. The following internal keywords are produced by SCLM internal processing and supported by SCLM. They cannot be used in the actual architecture definitions.

Keyword

Description

PINCL*

An architecture definition that generates the output shown on the previous build map entry. The output represents an input to the translate process.

INT*

An intermediate that was generated by the build of the member that is being viewed. This keyword represents the output of a translate process.

INTDEP*

Intermediate member on which the member being viewed is dependent. This keyword represents the input of a translate process.

WITH*

Indicates an upward dependency.

DYNI*

Indicates a dynamic include.

Ix*

Includes as determined by the accounting record for the main source member, where *x* is in the range (1-99).

EXTDPEND*

Indicates an external dependency.

NOPROM

Indicates member is left behind on promotion.

Authorization code update

Type U on the Library Utility panel or the member selection list to display the Authorization Code Update panel. Figure 82 on page 174 shows the panel SCLM displays for you to update the authorization code for a member.

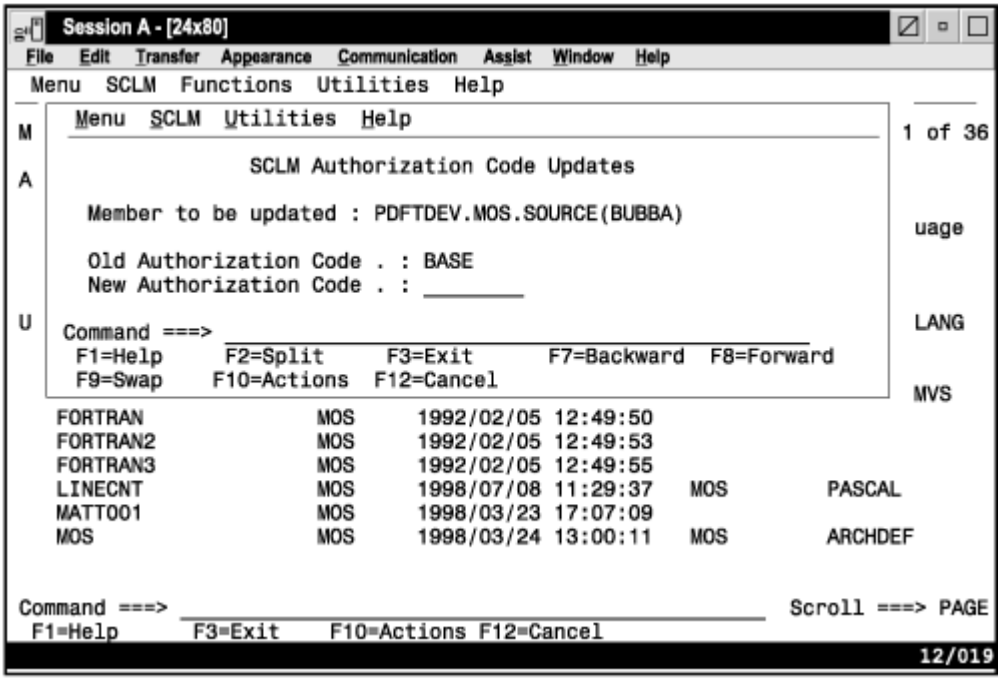


Figure 82. Authorization Code Update (FLMUSU#P)

The fields on the Authorization Code Update panel are:

Member to be updated
The member name you entered in the Member field on the SCLM Library Utility panel.

Old Authorization Code
The current authorization code for the member.

New Authorization Code
The new authorization code for the member.

Enter the new authorization code in this field. Then press Enter to confirm the update request and update the authorization code, or enter END to cancel the update request. Authorization codes cannot contain commas.

Transfer ownership

Type T on the Library Utility panel or against the member selection list to display the SCLM Transfer Ownership panel (FLMUSR#P).

The fields on the Transfer Ownership panel are:

Member to be updated
The member name you entered in the Member field on the SCLM Library Utility panel.

Old Member Userid
The ID of the user who currently has the member locked.

New Member Userid
The ID of the user who will control the member from now on.

Where-used

Enter the W line command on the SCLM Library Utility panel or on the member selection list to display a list of the components which include the selected component. A component is considered to be included if it has an entry in the Include List of another component, or if it has an *include-type* entry in the Build Map of another component. *Include-type* build map entries are: SINC, PROM, I*, PINCL*, INCL, DYNI*, and INCLD.

The Where-used panel shown in Figure 83 on page 175 displays the list of components which include the selected component.

```
  Menu  SCLM  Functions  Utilities  Test  Help
 ------------------------------------------------------------------------------
 FLMUSF#P d : LEEBURR.DEV1.SOURCE(FLM01EQU)                      Member 1 of 38
 Command ===> _____________________________________________ SCROLL ===> CSR

 A=Account  M=Map      B=Browse  D=Delete    E=Edit      V=View      Dyn(ON)
 C=Build    P=Promote  U=Update  S=UOW/Edit  T=Transfer  N=NOPROM   X=Expand
   ----+----1----+--
 S Member            Lv Type     AcctGrp  BmapGrp  AcctLang BmapLang  Keyword
 _   ASM@NO01        1  SOURCE   DEV1     DEV1     NOBJ     GLMAP     SINC
 _   ASM@NO02        1  SOURCE   DEV1     *NOBMAP* NOBJ     *NOBMAP*  SINC
 _ + FLM01CMD        1  ARCHDEF  DEV1     DEV1     ARCHDEF  CCMAP     I1*
 _ + FLM01MD3        1  SOURCE   DEV1     DEV1     HLAS     GLMAP     I1*
 _ + FLM01MD4        1  SOURCE   RELEASE  DEV1     HLAS     GLMAP     I1*
 _ + FLM01MD5        1  SOURCE   RELEASE  DEV1     HLAS     GLMAP     I1*
 _ + FLM01MD6        1  SOURCE   RELEASE  DEV1     HLAS     GLMAP     I1*
 _ + LB@SRC01        1  SOURCE   DEV1     DEV1     HLAS     CCMAP     SINC
 _   NEW@MD01        1  SOURCE   DEV1     *NOBMAP* NOBJ     *NOBMAP*  SINC
 _   NEW@MD02        1  SOURCE   DEV1     DEV1     HLAS     CCMAP     SINC
 _ + TST@SCLM        1  SOURCE   TEST     TEST     C20B000  GLMAP     I1*
 ******************************* Bottom of data
********************************



  F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
 F10=LEFT     F11=RIGHT    F12=Cancel
```

Figure 83. Where-used panel (FLMUSF#P)

The fields that appear on the Where-used panel are:

S
Input field for row command.

All row commands described in "Library Utility commands", except for the W command, can be used. Two additional commands are available:

X
Can be used to expand and collapse nested rows (where possible).

S
Invokes Unit Of Work if the selected member is an Archdef; otherwise, Edit is invoked.

Note: You must exit and re-enter the panel to refresh the WhereUsed list when parts are added or deleted by commands issued from the WhereUsed member list.

Name
The name of the including (or parent) components. The names are indented according to nesting level and the field may be scrolled left and right. The following indicators can appear to the left of the name:

+
The row can be expanded.

-
The row can be collapsed.

*****
The row has a circular reference and will not be expanded.

<
The row has exceeded the nesting level.

Lv
The nesting level.

Type
The type of the member.

AcctGrp
Group where the member's account record was found.

If the Acct record is not found, then AcctGrp shows "*NOACCT*".

BmapGrp
Group where the member's build map was found.

If the Bmap record is not found, then BmapGrp shows "*NOBMAP*".

AcctLang
Language of the member's account record.

BmapLang
Language of the member's build map record.

Keyword
The keyword used when referring to the included component.

Change Promote Processing (NOPROM)

Type N on the Library Utility panel or against the member selection list to display the SCLM Not Promoted Member Update panel (FLMUSN#P).

```
  Menu  SCLM  Utilities  Help
 ──────────────────────────────────────────────────────────────

                SCLM Not Promoted Member Update
Command ===> ____________________________________________________

   SCLM Library:
   PROJECT  . . . . : SLMTEST6
   GROUP  . . . . . : DEV1
   TYPE . . . . . . : SOURCE
   MEMBER . . . . . : FLM01EQU

   Options
   NOPROM:  1  1. No promote (Rebuild)
               2. No promote (No Rebuild)
               3. Remove no promote status

 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward
 F9=Swap     F10=Actions  F12=Cancel
```

Figure 84. SCLM Not Promoted Member Update panel (FLMUSN#P)

The fields on this panel are:

PROJECT
The SCLM project currently being maintained.

GROUP
Group of the member that is being changed.

TYPE
Type of the member that is being changed.

MEMBER
Member that is being changed.

NOPROM
Determines how the promote processing for this member is to be modified. Valid values are:

No promote (No Rebuild)
Sets the Accounting Status field in the accounting record to NOPROM-N. On promotion, the member is not copied to the next level. The build maps containing this member are not rebuilt, even if the associated language specifies an FLMLRBLD macro which would cause a rebuild.

No promote (Rebuild)
Sets the Accounting Status field in the accounting record to NOPROM-R. On promotion, the build maps containing this member and the member itself are not copied to the next level. Once the copy phase is complete, SCLM invokes a build to rebuild these build maps using the version of the member at the next level or above.

Remove no promote status
Sets the Accounting Status field in the accounting record to EDITABLE. Normal promotion of the member and associated build maps occurs.

For more information on this option, see the Chapter 15, “Leaving a Member Behind on Promotion,” on page 319.

Migration Utility

Using the migration utility, you can introduce members or groups of members to an SCLM project and place them under SCLM control in a development group. The migration utility also lets you verify authorization codes, prohibit simultaneous updates of members, and collect statistical, dependency, and historical information for each member processed without using the SCLM edit function. SCLM collects *dependency* information, which identifies software components that need another software component to complete successfully.

Before you start MIGRATE, the members must exist in the development library you specify. Upon successful completion of MIGRATE, each member selected will have valid SCLM accounting information. A typical scenario used to migrate existing project data follows:

1. Copy all of the members that have the same language into a development library.
2. Start MIGRATE using * for the member pattern and the appropriate language to parse all members and store their statistical, dependency, and historical information.
3. Copy all of the members that have a different language into the development library.
4. Start MIGRATE again using * for the member pattern and the new language.
5. Continue until all of the members have been migrated.

If some of the members have SCLM accounting information, the MIGRATE service verifies that the accounting information matches the member in the development library. MIGRATE takes no action for members that already have valid SCLM accounting information, unless executed in forced mode.

Use this utility when you have a large number of members that have not been entered in your project database, such as members that you did not create with the SCLM edit function.

In addition to the SCLM editor, the Migration Utility lets you indicate the members you want tracked. Use this utility to enter one or more members into a database of a project (for example, during a conversion to SCLM). In development groups, you can also use it to lock, parse, and create accounting records for members that have not been registered to SCLM.

Like the SCLM editor, the migration utility verifies authorization codes, prohibits simultaneous updates of members, and collects statistical, dependency, and historical information for every member processed. SCLM stores this information in the database of a project. For a complete description of the lock, parse, and store process, refer to:

- “LOCK—Lock a Member or Assign an Access Key” on page 430
- “PARSE—Parse a Member for Statistical and Dependency Information” on page 442
- “STORE—Store Member Information in an Accounting Record” on page 459

Figure 85 on page 178 shows the panel that appears when you select Option 3, Migration, from the Utilities Panel.

```
  Menu  SCLM  Utilities  Jobcard  Help
──────────────────────────────────────────────────────────────────────
                    SCLM Migration Utility - Entry Panel
Command ===> ____________________________________________________________

Selection criteria:
  Project  . : PDFTDEV
  Group  . . . PDFTDEV
  Type . . . . MOS
  Member . . . SOURCE    (Pattern may be used)

Member information:
  Authorization code  . . REL       Mode  . . . 1  1. Conditional
  Change code . . . . . . 2                       2. Unconditional
  Language  . . . . . . . PASCAL                  3. Forced
  Subproject  . . . . . . ________
Output control:
              Ex Sub                Process . . 2  1. Execute
  Messages . . 3  3    1. Terminal                 2. Submit
  Report . . . 3  3    2. Printer
  Listings . . 3  3    3. Data set   Printer . . _
                       4. None       Volume  . . ______

 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 85. SCLM Migration Utility (FLMUM#P)

Note: The NRETRIEV command key is enabled to work with this option. See “Name retrieval with the NRETRIEV command” on page 143 for more information.

The action bar displays the same choices as those discussed in “SCLM main menu action bar choices” on page 146. An additional choice is Jobcard.

The fields for the Migration Utility - Entry panel are.

Project
The project that you specified on the SCLM Main Menu. You cannot change this field. An Alternate field also appears if you specified an alternate project.

Group
The group in which the members to be migrated are located. This group must be defined in the project definition and must be a development group.

Type
The type in which the members to be migrated are located. This type must be defined in the project definition.

Member
The name of the member you want processed. You can use patterns for the member name. See “Specifying selection criteria” on page 181 for details.

Authorization code
The authorization code for a member. SCLM cannot process a member if the authorization code assigned to a member is not in the group being accessed. Authorization codes cannot contain commas.

Change code
The change code for the member. To enter a different change code for the member, type over the displayed change code. A change code verification routine can verify the code you entered before it processes the member. Change codes cannot contain commas.

Language
The language of the member. See Chapter 22, “SCLM translators,” on page 565 for a list of languages for which SCLM supplies parsers.

Mode
Select one of the following modes:

Conditional
To stop processing members if migrate discovers an error that is greater than the GOODRC parameter specified for a language parser in the project definition.

If you have a list of members that you want to place under SCLM control, and migrate fails for one of those members, processing stops after the first error. Migrate does not process any other members that match the specified criteria.

Unconditional
To continue processing regardless of errors discovered during parsing of each member.

If you have a list of members that you want to place under SCLM control, migrate attempts to process all the members matching the selection criteria, regardless of any errors encountered.

Forced
Forces SCLM to create a new accounting record for the members specified regardless of previous status. Processing stops after the first error is encountered.

If you have a list of members that need to be changed, migrate will create new accounting records for any members specified. This can be used to update language, authorization code or change code information for the specified members.

Output control
Specify the destination for messages, report, and listings when they are executed (Ex) or submitted (Sub), by entering the corresponding destination number: 1 for Terminal, 2 for Printer, 3 for Dataset, or 4 for None.

Process
You can call the processing part of the migration utility from the interactive or batch environment by selecting Execute or Submit, respectively. If you request batch processing by selecting Submit, you must specify the job statement information that is used in the JCL generated for batch processing.

For information about using a unique jobname on the jobcard in batch processing, see “Batch Processing” on page 252.

Printer
Specify the printer output class.

Volume
Specify the volume on which SCLM should save data sets.

Database Contents Utility

You can use the SCLM database contents utility to retrieve information about the project hierarchy from the project database and produce a report. You control the order and format of the data in the report. The utility generates a report that lists the members that match your selection criteria.

This accounting data can then be extracted for members in the database that meet the selection criteria you specify.

The output from the database contents utility can be used as input to other project-defined tools or as input to the SCLM services using the FILE format of FLMCMD.

Figure 86 on page 180 shows the panel that appears when you select Option 4, Database Contents, from the Utilities panel.

```
  Menu  SCLM  Utilities  Jobcard  Help
_____________________________________________________________________
                SCLM Database Contents Utility - Entry Panel
Command ===> _____________________________________________________________

Selection criteria: (Pattern may be used for Group, Type or Member)
  Project . . : PDFTDEV
  Group . . . . SBURNF   . . . ________ . . . ________
          . . . ________ . . . ________ . . . ________
  Type  . . . . *_______
  Member  . . . *_______

  Enter "/" to select option
  /  Change additional selection criteria
  /  Change customization parameters

Output control:
               Ex Sub                  Process . . 2  1. Execute
  Messages . . 3  3    1. Terminal                    2. Submit
  Report . . . 3  3    2. Printer
  Tailored . . 3  3    3. Data set     Printer . . H
                       4. None         Volume  . . ______
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 86. SCLM Database Contents Utility (FLMRC#P)

The fields on the Database Contents Utility panel are:

Project
The project that you specified on the SCLM Main Menu. An Alternate field also appears if you specified an alternate project.

Group
The groups that are to be reported. Only groups defined to the project definition are allowed.

Type
The name of the type you want processed. Only types defined to the project definition are allowed.

Member
The name of the member you want processed.

Change additional selection criteria
Select this field if you want to change the additional selection criteria. The panel shown in Figure 87 on page 181 appears when you select this.

If you change additional selection criteria, the changes are carried over from one execution to another. If you do not select this field, and thus do not change the additional criteria, the criteria from the last report are used.

Output control
Specify the destination for messages, reports, and tailored output when they are executed (Ex) or submitted (Sub), by entering the corresponding destination number: 1 for Terminal, 2 for Printer, 3 for Data set, or 4 for None. You cannot select Terminal for both Report and Tailored Output. Similarly, you cannot select None for both Report and Tailored Output. If the tailored output is to be used as input to a tool or to the SCLM services, Data set should be specified for Tailored Output.

If you enter Terminal, Printer, or Data set in the Tailored Output field, the panel shown in Figure 89 on page 184 appears.

Process
You can call the processing part of the database contents utility from the interactive or batch environment by selecting Execute or Submit, respectively. If you request batch processing by selecting Submit, you must specify the job statement information that is used in the JCL generated for batch processing.

Printer
Specify the printer output class.

Volume
Specify the volume on which SCLM should save data sets.

For information about using a unique jobname on the jobcard in batch processing, see "Batch Processing" on page 252.

Specifying selection criteria

The portion of the project database that SCLM displays is determined by the parameters you specify. You can use patterns to specify a variety of acceptable values for the accounting information fields. See "Selection parameters" on page 367 for more information and examples.

The panel in Figure 87 on page 181 is displayed if you select "Change additional selection criteria" field on the Database Contents Utility panel.

If you do not select this, the panel does not appear and the reports are generated with the values that already exist on the Additional Selection Criteria panel.

```
  Menu
_____________________________________________________________________

            SCLM Database Contents - Additional Selection Criteria
Command ===> ___________________________________________________________________

Selection criteria: (Patterns may be used)
  Authorization code  . . REL_____          Data type . . 1  1. Account
  Change code . . . . . . *_______                           2. Build map
  Change group  . . . . . USERID__                           3. Both
  Change user id  . . . . *_______
  Language  . . . . . . . *_______          Enter "/" to select option
                                            /  First occurrence only
Hierarchy search information:
  Architecture Control . . 3  1. In          Scope . . 1  1. Normal
                              2. Out                      2. Subunit
                              3. Not used                 3. Extended

  Architecture Group . . . USERID__
  Architecture Type  . . . ARCHDEF_
  Architecture Member  . . ________


 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 87. SCLM Database Contents - Additional Selection Criteria (FLMRCA)

The fields on the Additional Selection Criteria panel allow you to specify accounting and architecture information that the utility uses to identify the members to be processed.

Accounting information fields

When you specify values or patterns for the accounting information fields, the utility selects any member that has accounting information matching all of the patterns or values for all fields you specify.

Use the following accounting information fields to select members:

Authorization code
Members that are assigned an authorization code matching the authorization code. Authorization codes cannot contain commas.

The logical NOT symbol (¬) in the pattern specifies only the members that are not assigned an authorization code matching the pattern.

Change code
Members that can be edited that were assigned a change code matching the change code pattern. Change codes cannot contain commas.

Only one of the change codes assigned to the member must match the pattern. The logical NOT symbol (¬) in the pattern specifies only the members that are not assigned a change code matching the pattern.

Change group
Members that were last changed in a group matching the change group pattern.

Change user id
Members that were last changed by the user ID matching the change user ID pattern.

Language
Members whose language matches the language pattern.

Data type
Specify the following values:

Account
To report exclusively on accounting information.

Build Map
To report exclusively on build map information.

Both
To report on build map and accounting information.

Data type defaults to Account if nothing is specified.

First occurrence only
If you select this and use more than one group pattern, a precedence system determines which members are selected.

The group1 pattern takes precedence over the group2 pattern, which takes precedence over the group3 pattern, and so on. If SCLM finds versions of a member in groups matching more than one pattern, it selects only the version at the group with the most precedence. If more than one version of the member matches the pattern with the most precedence, it selects all of those versions.

If you do not select this field, SCLM selects all versions of all members.

Hierarchy search information

These fields allow you to use architecture definition criteria to select members. The architecture definition fields identify subapplications or software components.

To guarantee correct data, use the build function to update the architecture in the Architecture Control field. If you specify an architecture that has never been built, none of the members is selected. If you specify an architecture that has been built but is out of date, the resulting data is inaccurate. Promote the architecture in report-only mode to see which components are out of date. Patterns are not valid for architecture definition fields.

Architecture Control
Specify the following values:

In
To select members controlled by the architecture definition.

Out
To select members not controlled by the architecture definition.

Not used
To indicate that an architecture definition is not used to identify selected members.

Architecture Group
The group identifying the lowest group in the hierarchy where SCLM should find the architecture definition.

Architecture Type
The type containing the architecture definition that controls the selected members.

Architecture Member
The member containing the architecture definition that controls the selected members.

Scope
Specify the following architecture scope:

Normal
To select members that do or do not have compilation unit dependencies.

Subunit
To select members that do have compilation unit dependencies.

Extended
To select members that do have compilation unit dependencies.

The database contents report contains a list of all members that you select from the selection criteria. If you request tailored output, SCLM generates the data set from this list of accounting and build map information.

Figure 88 on page 183 shows an example of a database contents utility report that SCLM generates when you enter NONE in the Tailored Output field on the SCLM Database Contents Utility panel.

```
                    DATABASE CONTENTS UTILITY REPORT

                              SELECTION CRITERIA
PROJECT   : PROJ1
ALTERNATE: PROJ1       AUTHORIZATION CODE     : REL
TYPES     : SOURC*     CHANGE CODE            : *
MEMBERS   : *          CHANGE GROUP           : USER1
GROUP 1   : USER1      CHANGE USER ID         : *
GROUP 2   : INT        LANGUAGE               : *
GROUP 3   :            FIRST OCCURRENCE ONLY  : YES
GROUP 4   :            DATA TYPE              : ACCT
GROUP 5   :
GROUP 6   :

                    ARCHITECTURE SELECTION CRITERIA : IN
GROUP     : USER1
TYPE      : ARCHDEF
MEMBER    : FLM01LD4
SCOPE     : NORMAL

                      DATE: 02/23/1989    TIME: 11:26:18
```

```
DATABASE CONTENTS REPORT                                          PAGE        2
------------------------------ TYPE: SOURCE ---------------------------------
MEMBER     GROUP1     GROUP2     GROUP3     GROUP4     GROUP5     GROUP6
--------   --------   --------   --------   --------   --------   --------
FLM01MD4   USER1
FLM01MD5              INT
FLM01MD6              INT
------------------------------ TYPE: SOURCE2 --------------------------------
INCLUDE3              INT
```

Figure 88. Database Contents Utility Report

Note: An asterisk (*) next to the group name on a report indicates that the member represents build map information.

Tailored output

If you want to tailor the database contents output, select `Terminal`, `Printer`, or `Dataset` in the Tailored Output field on the Database Contents Utility panel. The Customization Parameters panel appears, shown in Figure 89 on page 184, which you use to generate the tailored output.

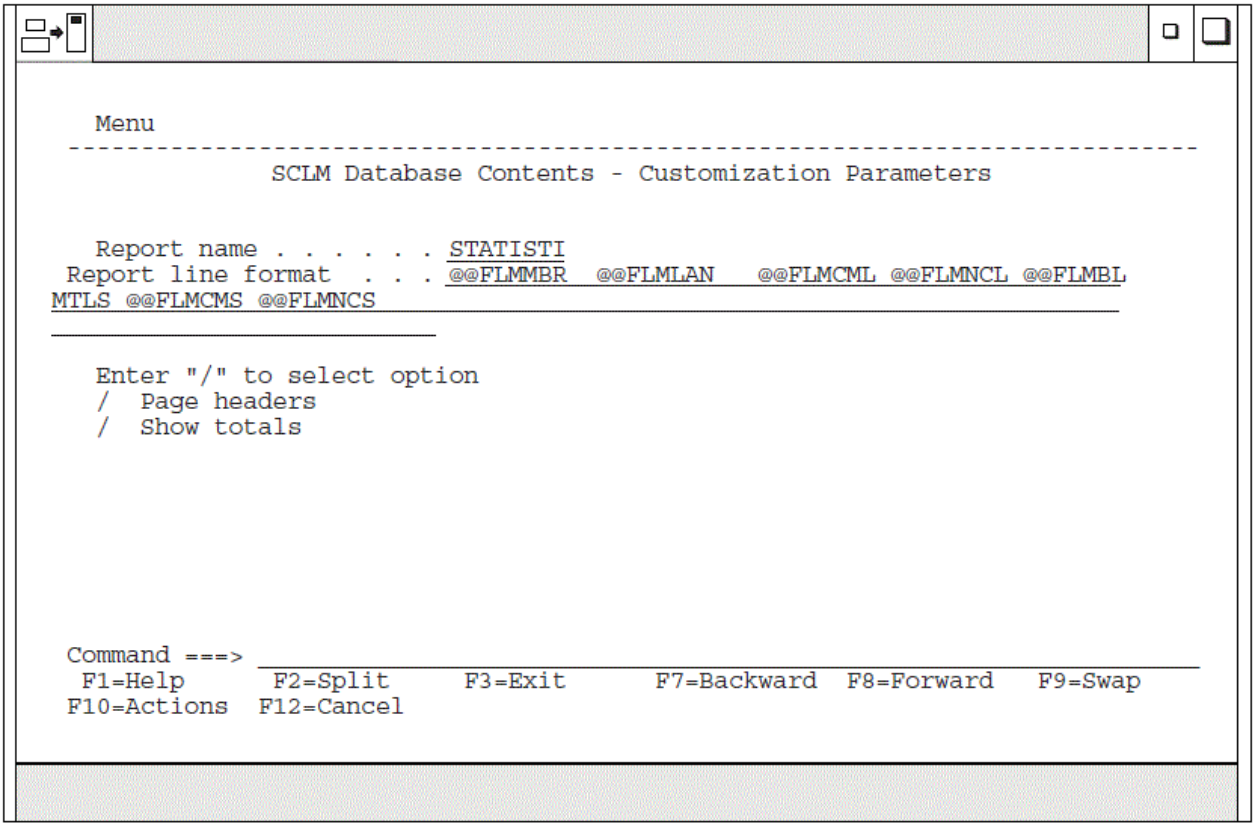


Figure 89. SCLM Database Contents - Customization Parameters (FLMRCT)

The fields on the Customization Parameters panel are:

Report name
The title of the report in the tailored output. The maximum length is 35 characters. Do not use commas in this field. The default value for Report name is STATISTICS REPORT.

Report line format
The format of a line of data in the tailored output. The line format can be up to 160 characters long.

Report line format has a default value, which is used when no values are specified:

```
@@FLMMBR  @@FLMLAN  @@FLMCML  @@FLMNCL @@FLMBLL @@FLMTLS
@@FLMCMS  @@FLMNCS
```

If you use SCLM variables with data lengths greater than 8 characters, place these variables at the end of the report line to ensure that the columns in the report line up evenly.

You can use any string or character as a literal. When you use literals, the string prints once on each output line.

The report line has a maximum size of 2048 characters. The tailored output prints 80 characters per line. This can produce multiple 80-character lines for one report line.

Press Enter to confirm these requests or enter END to cancel them.

Page headers

Select "Page headers" to include page and column header information in the tailored output. If you want to output a page header, input parameter information appears in the tailored output. You can also specify a title. Data is positioned in column 2 of the tailored output. Column 1 is used for carriage returns.

If you do not select "Page headers", page headers and carriage returns are suppressed. The data is positioned in column 1 of the tailored output.

The default value for "Page headers" is that they are selected.

Show totals

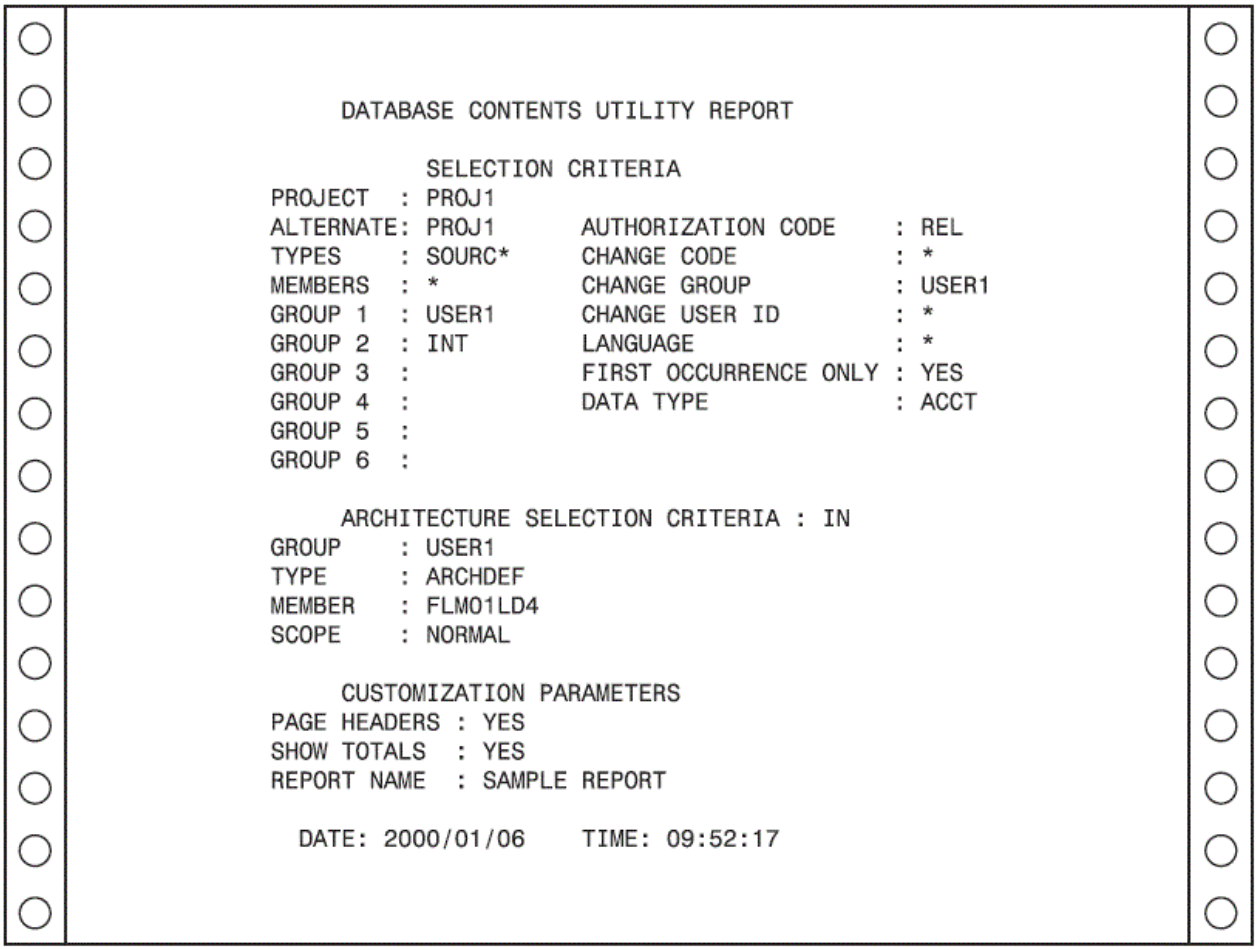
Select this to total the numeric data fields and show the totals in the tailored output. SCLM outputs a summary line at the end of the output that totals the values of the numeric fields in the output. The output also includes a count of the number of members reported. The default value for "Show totals" is that they are selected.

Figure 90 on page 186 shows an example of a tailored output. The title of the report is `Sample Report`. The report line format, specified as `@@FLMPRJ @@FLMGRP @@FLMTYP @@FLMMBR`, causes the utility to generate output consisting of the members reported in the database contents report and their associated included members.

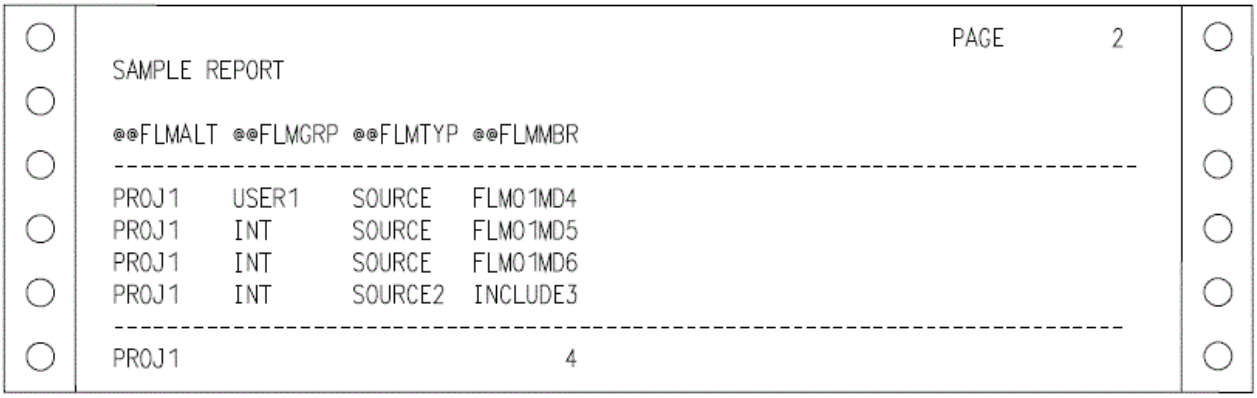
Tailored output examples

The tailored output that appears in Figure 90 on page 186 is a formatted representation of the accounting and build map information of the members that matched the selection criteria. The tailored output format specification consists of SCLM variables and constant values. The tailored output displays the SCLM variables as headers over the lines of variable values.

“SCLM variable and metavariable descriptions” on page 631 provides a list of SCLM variables that can be used in the database contents utility.



```
                DATABASE CONTENTS UTILITY REPORT

                      SELECTION CRITERIA
PROJECT   : PROJ1
ALTERNATE: PROJ1        AUTHORIZATION CODE     : REL
TYPES     : SOURC*      CHANGE CODE            : *
MEMBERS   : *           CHANGE GROUP           : USER1
GROUP 1   : USER1       CHANGE USER ID         : *
GROUP 2   : INT         LANGUAGE               : *
GROUP 3   :             FIRST OCCURRENCE ONLY  : YES
GROUP 4   :             DATA TYPE              : ACCT
GROUP 5   :
GROUP 6   :

     ARCHITECTURE SELECTION CRITERIA : IN
GROUP     : USER1
TYPE      : ARCHDEF
MEMBER    : FLM01LD4
SCOPE     : NORMAL

     CUSTOMIZATION PARAMETERS
PAGE HEADERS : YES
SHOW TOTALS  : YES
REPORT NAME  : SAMPLE REPORT

  DATE: 2000/01/06    TIME: 09:52:17
```



```
                                                                    PAGE        2
SAMPLE REPORT

@@FLMALT @@FLMGRP @@FLMTYP @@FLMMBR
--------------------------------------------------------------------------------
PROJ1    USER1    SOURCE   FLM01MD4
PROJ1    INT      SOURCE   FLM01MD5
PROJ1    INT      SOURCE   FLM01MD6
PROJ1    INT      SOURCE2  INCLUDE3
--------------------------------------------------------------------------------
PROJ1                              4
```

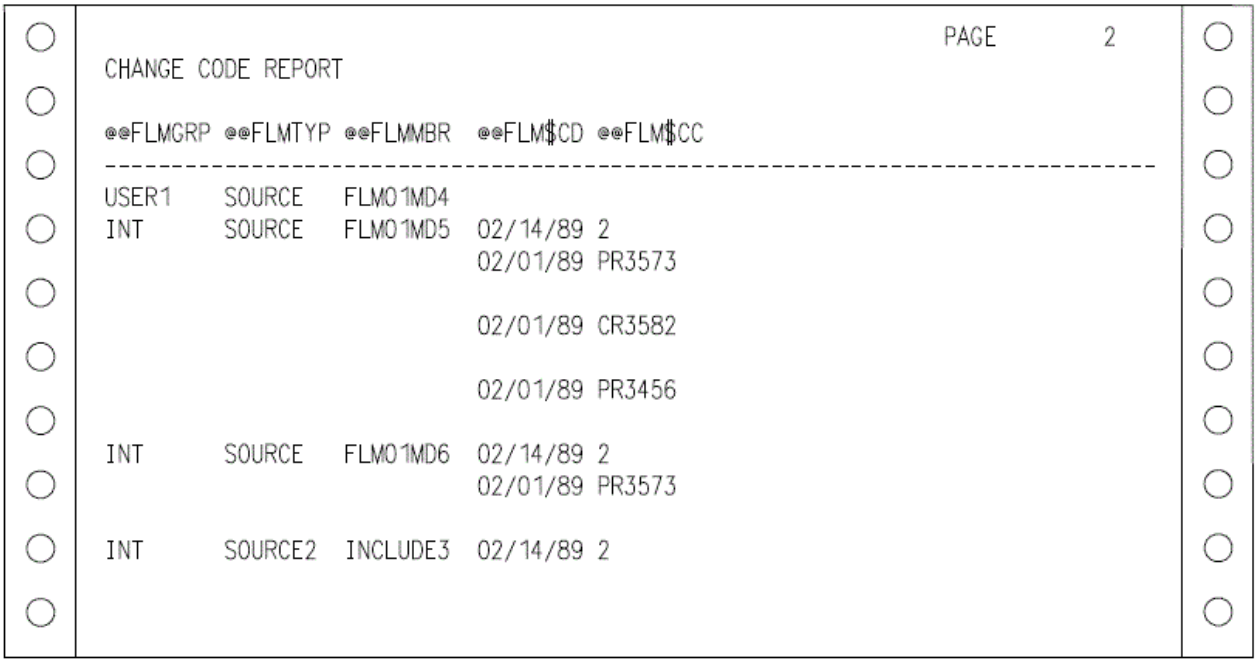
Figure 90. Database Contents Utility Tailored Output

The tailored output examples in figures Figure 91 on page 187 through Figure 94 on page 188 show examples of change code, accounting statistics, source listing, and cleanup reports.

Change Code Report

The report name is CHANGE CODE REPORT.

The report line format input for this example is: `@@FLMGRP @@FLMTYP @@FLMMBR @@FLM$CD @@FLM $CC`. The page headers appear on all pages of the report. Totals do not appear. Figure 91 on page 187 shows the tailored output.



```
                                                                    PAGE        2
CHANGE CODE REPORT

@@FLMGRP @@FLMTYP @@FLMMBR  @@FLM$CD @@FLM$CC
--------------------------------------------------------------------------------
USER1    SOURCE   FLMO1MD4
INT      SOURCE   FLMO1MD5  02/14/89 2
                            02/01/89 PR3573

                            02/01/89 CR3582

                            02/01/89 PR3456

INT      SOURCE   FLMO1MD6  02/14/89 2
                            02/01/89 PR3573

INT      SOURCE2  INCLUDE3  02/14/89 2
```

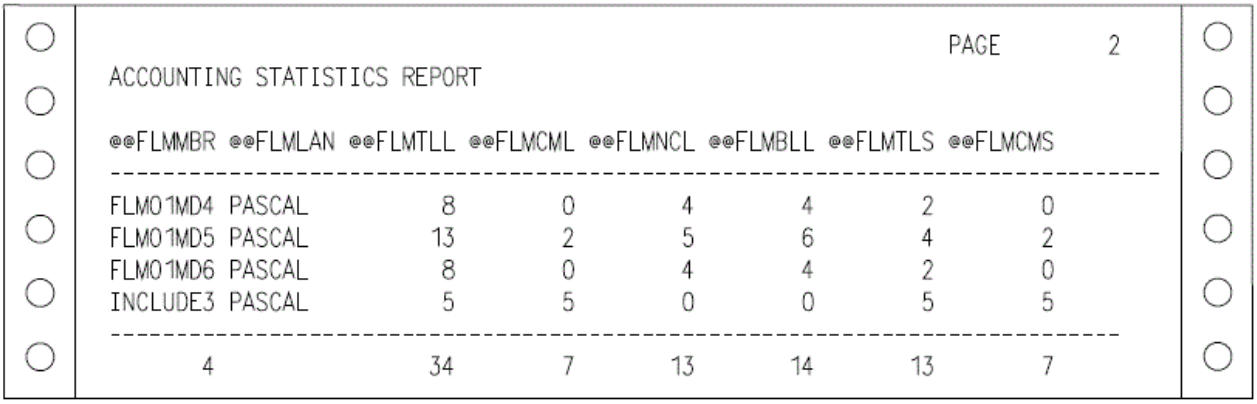
Figure 91. Change Code Report, Page 2

Accounting Statistics Report

The report name is ACCOUNTING STATISTICS REPORT.

The report line format input for this example is: `@@FLMMBR @@FLMLAN @@FLMTLL @@FLMCML @@FLMNCL @@FLMBLL @@FLMTLS @@FLMCMS`.

The page headers appear on all pages of the report. Totals appear for all numeric data. Figure 92 on page 187 shows the tailored output.



```
                                                                    PAGE        2
ACCOUNTING STATISTICS REPORT

@@FLMMBR @@FLMLAN @@FLMTLL @@FLMCML @@FLMNCL @@FLMBLL @@FLMTLS @@FLMCMS
--------------------------------------------------------------------------------
FLMO1MD4 PASCAL          8         0         4         4         2         0
FLMO1MD5 PASCAL         13         2         5         6         4         2
FLMO1MD6 PASCAL          8         0         4         4         2         0
INCLUDE3 PASCAL          5         5         0         0         5         5
--------------------------------------------------------------------------------
       4                34         7        13        14        13         7
```

Figure 92. Accounting Statistics Report, Page 2

Source Listing Report

This example shows a generated script data set that the SCRIPT/VS processor can process.

The report line format input for this example is: `.IM @@FLMMBR`.

The report does not have page headers, totals, or a name. Figure 93 on page 188 shows the tailored output.

```
.IM FLM01MD4
.IM FLM01MD5
.IM FLM01MD6
.IM INCLUDE3
```

Figure 93. Source Listing Report

Cleanup Report

The cleanup data set is a command data set that can be passed as input to the SCLM command processor. See "The FLMCMD interface" on page 359 for more information on the SCLM command processor.

The report line format input for this example is:
`DELETE,@@FLMPRJ,@@FLMALT,@@FLMGRP,@@FLMTYP,@@FLMMBR`.

The report does not have page headers, totals, or a name. Figure 94 on page 188 shows the sample tailored output.

```
DELETE,PROJ1   ,PROJ1   ,USER1   ,SOURCE  ,FLM01MD4
DELETE,PROJ1   ,PROJ1   ,INT     ,SOURCE  ,FLM01MD5
DELETE,PROJ1   ,PROJ1   ,INT     ,SOURCE  ,FLM01MD6
DELETE,PROJ1   ,PROJ1   ,INT     ,SOURCE2 ,INCLUDE3
```

Figure 94. Cleanup Report

Architecture Report Utility

The architecture report provides listings of all the components in a given application. The report generator examines the requested architecture and all of its references, and then constructs a formatted report. The report lists software components in each type referenced by the architecture. One advantage of the report is that it helps you to eliminate unnecessary code. The title page of the report identifies the date and time SCLM generated the report, names the architecture member you requested, and is based on the report cutoff you select. It also identifies any alternate project definition used.

The report is divided into two sections:

- Architecture

  Lists all architecture and source members subordinate to a given architecture to the report cutoff you specify. The architecture information is particularly useful during the development stages of a project to identify the current status of the application architecture. It is also useful at any time to determine a list of the software components of an application.

  The report uses an indentation format to present a visual concept of the structure of the application. It also lists the number architecture types processed.

- Cross-reference

  Lists all the members, by type, that are referenced by members in the first part of the report. Use this information to determine the origin of a member.

Figure 96 on page 191 shows an example of an architecture report.

SCLM displays the panel in Figure 95 on page 189 when you select Option 5, Architecture Report, on the Utilities panel.

Note: Compilation unit dependencies are not used to generate the architecture report.

The architecture report is divided into three parts: a header, architecture information, and cross-reference information. The architecture report header lists the accounting and architecture selection criteria plus the customization parameters you specify. The architecture information lists all of the software components, by type, in a specified application. This part of the report can help you eliminate unnecessary code. The cross-reference information indicates where a given software component is embedded in the architecture of the application.

```
  Menu  SCLM  Utilities  Jobcard  Help
................................................................................
                         SCLM Architecture Report Utility - Entry Panel

Report input:
  Project  . . : PROJ1         Alternate - INT
  Group  . . . . DEV1___
  Type . . . . . ________              Report
  Member . . . . ________              Cutoff  .  . 6  1. HL
                                                      2. LEC
                                                      3. CC
                                                      4. Generic
                                                      5. Top Source
                                                      6. None

Output control:
               Ex Sub                  Process . . 1  1. Execute
  Messages . . 3  3   1. Terminal                     2. Submit
  Report . . . 3  3   2. Printer
                      3. Data set      Printer . . _
                      4. None          Volume  . . ______

Command ===> ____________________________________________________________
 F1=Help      F2=Split    F3=Exit      F7=Backward F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 95. SCLM Architecture Report (FLMRA#P)

The fields on the SCLM Architecture Report Utility - Entry panel are:

Project
The project that you specified on the SCLM Main Menu. An Alternate field also appears if you specified an alternate project definition.

Group
The group used to identify the lowest group in the hierarchy where the architecture begins.

Type
The type containing the architecture definition that controls the selected member.

Member
The member containing the architecture definition.

Report Cutoff
You must specify one of the following report cutoff values (which determine the depth of the report):

HL (High-level)
To list only the HL architecture members in the application represented by the architecture member you specified in the Member field.

LEC (Linkedit control)
To list all of the HL and LEC architecture members in the application represented by the architecture member you specified in the Member field.

CC (Compilation control)
To list all of the HL, LEC, CC, Generic, and INCLD'ed members in the application represented by the architecture member you specified in the Member field.

GEN (Generic)
To list all of the HL and generic architecture members in the application represented by the architecture member you specified in the Member field.

Top Source
To list all of the HL, LEC, CC, Generic, and INCL'ed members and the top source members in the application represented by the member you specified in the Member field.

None
To list all HL, LEC, CC, and generic architecture members in each of the types and all source member names down to the lowest include group in the application represented by the architecture member you specified in the Member field.

The default value for Report Cutoff is None.

Output control
Specify the destination for messages and report when they are executed (Ex) or submitted (Sub), by entering the corresponding destination number: 1 for Terminal, 2 for Printer, 3 for Dataset, or 4 for None.

Process
You can call the processing part of the architecture report utility from the interactive or batch environment by selecting Execute or Submit, respectively. If you request batch processing by selecting Submit, you must specify the job statement information that is used in the JCL generated for batch processing.

For information about using a unique jobname on the jobcard in batch processing, see "Batch Processing" on page 252.

Printer
Specify the printer output class.

Volume
Specify the volume on which SCLM should save data sets.

Architecture Report example

Figure 96 on page 191 shows an example of the architecture report with a report cutoff of NONE. Figure 99 on page 194 shows an example of the architecture report with a report cutoff of LEC.

The architecture report provides lists of all the components in an application. The title page identifies the date and time the report was generated, the architecture member requested, and the report cutoff. It also identifies the alternate project definition, if specified.

```
********************************************************************************
********************************************************************************
**                                                                            **
**            SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)                **
**                                                                            **
**                             ARCHITECTURE REPORT                            **
**                                                                            **
**                            2000/01/06   00:01:30                           **
**                                                                            **
**                                                                            **
**                                                                            **
**                              PROJECT:   PROJ1                              **
**                              GROUP:     DEV1                               **
**                              TYPE:      ARCHDEF                            **
**                              MEMBER:    FLM01SB2                           **
**                              CUTOFF:    NONE                               **
**                                                                            **
**                                                                            **
********************************************************************************
********************************************************************************
================================================================================
*                                                                              *
*                              ARCHITECTURE REPORT                             *
*                                                                              *
*  H = HIGH LEVEL        C = COMPILATION CONTROL  T = TOP SOURCE E = ERROR     *
*  L = LINKEDIT CONTROL  G = GENERIC              I = INCLUDED   D = DEFAULT   *
*                                                                              *
================================================================================

CODE:  H   MEMBER:  FLM01SB2

----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---

H FLM01SB2 ARCHDEF
L  FLM01LD4 ARCHDEF
D   FLM01MD4 SOURCE
T    FLM01MD4 SOURCE
I     FLM01EQU SOURCE
D   FLM01MD6 SOURCE
T    FLM01MD6 SOURCE
I     FLM01EQU SOURCE
D   FLM01MD5 SOURCE
T    FLM01MD5 SOURCE
I     FLM01EQU SOURCE
L  FLM01LD3 ARCHDEF
D   FLM01MD3 SOURCE
T    FLM01MD3 SOURCE
I     FLM01EQU SOURCE
D   FLM01MD6 SOURCE
T    FLM01MD6 SOURCE
I     FLM01EQU SOURCE
D   FLM01MD5 SOURCE
T    FLM01MD5 SOURCE
I     FLM01EQU SOURCE

NUMBER OF HIGH LEVEL MEMBERS PROCESSED           =    1
NUMBER OF LINK EDIT CONTROL MEMBERS PROCESSED    =    2
NUMBER OF GENERIC MEMBERS PROCESSED              =    0
```

Figure 96. Architecture report with cutoff of NONE (Part 1 of 3)

```
NUMBER OF DEFAULT MEMBERS PROCESSED             =    4
NUMBER OF COMPILATION CONTROL MEMBERS PROCESSED  =    0
NUMBER OF TOP MEMBERS PROCESSED                 =    4
NUMBER OF INCLUDED MEMBERS PROCESSED            =    1
NUMBER OF ERROR MEMBERS FOUND                   =    0
================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     LIST                          *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
------    ---------------  ----       -------   -----------

FLM01MD3      FLM01MD3     SOURCE     LIST
FLM01MD4      FLM01MD4     SOURCE     LIST
FLM01MD5      FLM01MD5     SOURCE     LIST
FLM01MD6      FLM01MD6     SOURCE     LIST

TOTAL MEMBERS PROCESSED FOR TYPE  =  4

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     OBJ                           *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
------    ---------------  ----       -------   -----------

FLM01MD3      FLM01MD3     SOURCE     OBJ
FLM01MD4      FLM01MD4     SOURCE     OBJ
FLM01MD5      FLM01MD5     SOURCE     OBJ
FLM01MD6      FLM01MD6     SOURCE     OBJ

TOTAL MEMBERS PROCESSED FOR TYPE  =  4

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     SOURCE                        *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE     KEYWORD   INCLUDE-SET
------    ---------------  ----     -------   -----------

FLM01EQU      FLM01MD4     SOURCE     I1
              FLM01MD4     SOURCE
              FLM01MD3     SOURCE     I1
              FLM01MD3     SOURCE
              FLM01MD6     SOURCE     I1
              FLM01MD6     SOURCE
              FLM01MD5     SOURCE     I1
              FLM01MD5     SOURCE
FLM01MD3      FLM01MD3     SOURCE     SINC
              FLM01MD3     SOURCE     PROM
              FLM01LD3     ARCHDEF    INCLD
FLM01MD4      FLM01MD4     SOURCE     SINC
```

Figure 97. Architecture report with cutoff of NONE (Part 2 of 3)

```
              FLM01MD4     SOURCE     PROM
              FLM01LD4     ARCHDEF    INCLD
FLM01MD5      FLM01MD5     SOURCE     SINC
              FLM01MD5     SOURCE     PROM
              FLM01LD4     ARCHDEF    INCLD
              FLM01LD3     ARCHDEF    INCLD
FLM01MD6      FLM01MD6     SOURCE     SINC
              FLM01MD6     SOURCE     PROM
              FLM01LD4     ARCHDEF    INCLD
              FLM01LD3     ARCHDEF    INCLD

TOTAL MEMBERS PROCESSED FOR TYPE  =  22
================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     LMAP                        *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
------    ---------------  ----       -------   -----------

FLM01LD3      FLM01LD3     ARCHDEF    LMAP
FLM01LD4      FLM01LD4     ARCHDEF    LMAP

TOTAL MEMBERS PROCESSED FOR TYPE  =  2

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     LOAD                        *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
--------  ---------------  --------   --------  -----------

FLM01LD3      FLM01LD3     ARCHDEF    LOAD
FLM01LD4      FLM01LD4     ARCHDEF    LOAD

TOTAL MEMBERS PROCESSED FOR TYPE  =  2

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     ARCHDEF                     *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
------    ---------------  ----       -------   -----------

FLM01ARH      FLM01LD4     ARCHDEF    COPY
              FLM01LD3     ARCHDEF    COPY
FLM01LD3      FLM01LD3     ARCHDEF    PROM
              FLM01SB2     ARCHDEF    INCL
FLM01LD4      FLM01LD4     ARCHDEF    PROM
              FLM01SB2     ARCHDEF    INCL
FLM01SB2      FLM01SB2     ARCHDEF    PROM

TOTAL MEMBERS PROCESSED FOR TYPE  =  7
```

Figure 98. Architecture report with cutoff of NONE (Part 3 of 3)

```
********************************************************************************
********************************************************************************
**                                                                            **
**            SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)                **
**                                                                            **
**                             ARCHITECTURE REPORT                            **
**                                                                            **
**                            2000/01/06   00:02:30                           **
**                                                                            **
**                                                                            **
**                                                                            **
**                              PROJECT:   PROJ1                              **
**                              GROUP:     DEV1                               **
**                              TYPE:      ARCHDEF                            **
**                              MEMBER:    FLM01SB2                           **
**                              CUTOFF:    LINK EDIT CONTROL                  **
**                                                                            **
**                                                                            **
********************************************************************************
********************************************************************************

================================================================================
*                                                                              *
*                              ARCHITECTURE REPORT                             *
*                                                                              *
*  H = HIGH LEVEL        C = COMPILATION CONTROL  T = TOP SOURCE E = ERROR     *
*  L = LINKEDIT CONTROL  G = GENERIC              I = INCLUDED   D = DEFAULT   *
*                                                                              *
================================================================================

CODE:  H   MEMBER:  FLM01SB2

----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---

H FLM01SB2 ARCHDEF
L  FLM01LD4 ARCHDEF
L  FLM01LD3 ARCHDEF

NUMBER OF HIGH LEVEL MEMBERS PROCESSED           =    1
NUMBER OF LINK EDIT CONTROL MEMBERS PROCESSED    =    2
NUMBER OF ERROR MEMBERS FOUND                    =    0

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     SOURCE                      *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
------    ---------------  ----       -------   -----------

FLM01MD3      FLM01LD3     ARCHDEF    INCLD
FLM01MD4      FLM01LD4     ARCHDEF    INCLD
FLM01MD5      FLM01LD4     ARCHDEF    INCLD
              FLM01LD3     ARCHDEF    INCLD
FLM01MD6      FLM01LD4     ARCHDEF    INCLD
              FLM01LD3     ARCHDEF    INCLD

TOTAL MEMBERS PROCESSED FOR TYPE  =  6
```

Figure 99. Architecture report with cutoff of LEC (Part 1 of 2)

```
================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     LMAP                        *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
------    ---------------  ----       -------   -----------

FLM01LD3      FLM01LD3     ARCHDEF    LMAP
FLM01LD4      FLM01LD4     ARCHDEF    LMAP

TOTAL MEMBERS PROCESSED FOR TYPE  =  2

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     LOAD                        *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
------    ---------------  ----       -------   -----------

FLM01LD3      FLM01LD3     ARCHDEF    LOAD
FLM01LD4      FLM01LD4     ARCHDEF    LOAD

TOTAL MEMBERS PROCESSED FOR TYPE  =  2

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     ARCHDEF                     *
*                                                                              *
================================================================================

MEMBER    REF. ARCH. MEM.  TYPE       KEYWORD   INCLUDE-SET
------    ---------------  ----       -------   -----------

FLM01ARH      FLM01LD4     ARCHDEF    COPY
              FLM01LD3     ARCHDEF    COPY
FLM01LD3      FLM01LD3     ARCHDEF    PROM
              FLM01SB2     ARCHDEF    INCL
FLM01LD4      FLM01LD4     ARCHDEF    PROM
              FLM01SB2     ARCHDEF    INCL
FLM01SB2      FLM01SB2     ARCHDEF    PROM

TOTAL MEMBERS PROCESSED FOR TYPE  =  7
```

Figure 100. Architecture report with cutoff of LEC (Part 2 of 2)

Export Utility

The export utility writes accounting and cross-reference data to standalone and portable accounting and cross-reference databases that contain only those records associated with a specified group. The export utility does not change any data currently residing in the specified group. The output from the export utility is used as input to the import utility.

With the export utility, you can capture SCLM accounting information associated with a specified group. Use the export utility when you want to create a consistent set of data to archive or transport. You can specify that the exported accounting information be purged from an existing export VSAM data set.

Export only works on accounting information. Data in project partitioned data sets is *not* exported.

Before using the export utility, verify that the project manager has completed all the steps required to perform the export setup task. Specifically, export data sets must be defined and allocated for the group in the project from which the data is exported.

Figure 101 on page 196 shows the panel that appears when you select Option 6, Export, from the Utilities panel.

```
  Menu  SCLM  Utilities  Jobcard  Help
................................................................................
                         SCLM Export Utility - Entry Panel

Selection criteria:
  Project  . . : PROJ1         Alternate - INT
  Group  . . . . DEV1

  Enter "/" to select option
  /  Replace export data

Output control:
               Ex Sub                   Process . . 1  1. Execute
  Messages . . 3  3   1. Terminal                      2. Submit
  Report . . . 3  3   2. Printer
                      3. Data set       Printer . . _
                      4. None           Volume  . . ______



Command ===> ____________________________________________________________
 F1=Help      F2=Split    F3=Exit      F7=Backward F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 101. SCLM Export Utility (FLMDXE#P)

To export an SCLM group, enter information for each field. The fields for the Export Utility - Entry panel are:

Project
The project that you specified on the SCLM Main Menu. An Alternate field also appears if you specified an alternate project definition.

Group
The group from which you are exporting data.

Replace export data
Specify whether to replace the export accounting and cross-reference data in the export data sets with data from this export. If you do not select this field and the export data sets contain data, the data is not replaced, the export is not performed, and an error message is issued.

Export does *not* purge data from the project hierarchy primary accounting and cross-reference data sets.

Output control
Specify the destination for messages and reports when they are executed (Ex) or submitted (Sub) by entering the corresponding destination number.

Process
You can call the processing part of the export utility from the interactive or batch environment by selecting Execute or Submit, respectively. If you request batch processing by selecting Submit, you must specify the job statement information that is used in the JCL generated for batch processing.

For information about using a unique jobname on the jobcard in batch processing, see “Batch Processing” on page 252.

Printer
Specify the printer output class

Volume
Specify the volume on which SCLM should save data sets

Export Report example

Figure 102 on page 198 shows a sample export report.

The report contains a header indicating that it is an Export Report, which project definition and group are being exported, and the data set names of the VSAM files that contain the exported information. The header is followed by three sections: accounting records, build map records, and intermediate records. The report always contains a section for each type even if no records of that type were processed.

The Verify Status field contains the value PASSED unless one of the following conditions is true:

- The authorization code change field is nonblank for the record
- The accounting type is INITIAL
- The record could not be read

The Completion Status field contains the value PASSED if the record was exported; otherwise, it contains the value FAILED, which means there was some error writing the record to the export database. Completion Status should always contain the value NOT ATTEMPTED if the Verify Status field contains the value FAILED, because SCLM does not attempt to export a record if the record did not pass verification.

If the export cross-reference data set is defined for the project definition, the cross-reference records are also exported; but the export report does not include them. If the export cross-reference data set is not defined for the project definition, but the group being exported contains cross-reference records, the Verify Status is set to FAILED and the Completion Status is set to NOT ATTEMPTED. No intermediate records are processed.

```
********************************************************************************
********************************************************************************
**                                                                            **
**                                                                            **
**             SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)               **
**                                                                            **
**                                 EXPORT  REPORT                             **
**                                                                            **
**                              2002/08/27   11:55:02
**
**                                                                            **
**                           PROJECT:      BTRANS                             **
**                           ALTERNATE:    BTRANS                             **
**                           GROUP:        DEV1                               **
**                                                                            **
**                                                                            **
**    EXPORT ACCOUNTING FILE: BTRANS.EXPORT.ACCOUNT.DATABASE                  **
**    EXPORT CROSSREF FILE:                                                   **
********************************************************************************
********************************************************************************

 ACCOUNTING RECORDS:                                                 PAGE: 1
                              VERIFY     COMPLETION
      TYPE        MEMBER      STATUS     STATUS
    --------     --------     ------    -------------
    ARCHDEF      FLM01CMD     PASSED     PASSED
    ARCHDEF      FLM01LD1     PASSED     PASSED
    ARCHDEF      JTEST02      PASSED     PASSED
    ARCHDEF      PMR60436     PASSED     PASSED
    ARCHDEF      P02788A      PASSED     PASSED
    COPYLIB      BCEWCADA     PASSED     PASSED
    COPYLIB      BCEWCHNG     PASSED     PASSED
    COPYLIB      BCEWFLAG     PASSED     PASSED
    COPYLIB      BCEWPMVT     PASSED     PASSED
    COPYLIB      BRSGEC       PASSED     PASSED
    COPYLIB      BRSSECAU     PASSED     PASSED
    COPYLIB      BRSSVDC1     PASSED     PASSED
    COPYLIB      BRSSVDC2     PASSED     PASSED
    COPYLIB      BRSSZIC1     PASSED     PASSED
    COPYLIB      BRSSZIC2     PASSED     PASSED
    COPYLIB      BRSWINOP     PASSED     PASSED
    COPYLIB      BRSWMTPP     PASSED     PASSED
    COPYLIB      BRSWOLCT     PASSED     PASSED
    COPYLIB      CCOURAN      PASSED     PASSED
    COPYLIB      CPYA0001     PASSED     PASSED
    COPYLIB      DCACCNTN     PASSED     PASSED
    COPYLIB      ISIWLOCK     PASSED     PASSED
    COPYLIB      ISIWLOG      PASSED     PASSED
    COPYLIB      ISIWMCHG     PASSED     PASSED
    COPYLIB      ISIWMERR     PASSED     PASSED
    COPYLIB      ISIWNCGS     PASSED     PASSED
    COPYLIB      ISIWNUME     PASSED     PASSED
    COPYLIB      ISIWNVAL     PASSED     PASSED
    COPYLIB      ISIWSTAT     PASSED     PASSED
    COPYLIB      ISIWTARI     PASSED     PASSED
    COPYLIB      ISIWVALO     PASSED     PASSED
    COPYLIB      PPMWRM22     PASSED     PASSED
    COPYLIB      SYSWRACF     PASSED     PASSED
    COPYLIB      TITWPGMJ     PASSED     PASSED
    LMAP         FLM01LD3     PASSED     PASSED
    LMAP         PMR60436     PASSED     PASSED
    LOAD         FLM01LD3     PASSED     PASSED
    LOAD         PMR60436     PASSED     PASSED
```

Figure 102. Export Report (Part 1 of 2)

```
ACCOUNTING RECORDS:                                                PAGE: 2
                            VERIFY     COMPLETION
     TYPE        MEMBER     STATUS     STATUS
   --------     --------    ------    -------------
   OBJ          FLM01MD1    PASSED     PASSED
   OBJ          FLM01MD3    PASSED     PASSED
   OBJ          PMR60436    PASSED     PASSED
   PNL          VRCPT03     PASSED     PASSED
   SOURCE       CPYRITE     PASSED     PASSED
   SOURCE       DTL2        PASSED     PASSED
   SOURCE       FLM01MD1    PASSED     PASSED
   SOURCE       FLM01MD3    PASSED     PASSED
   SOURCE       FLM01MD6    PASSED     PASSED
   SOURCE       PMR60436    PASSED     PASSED
   SOURCE       P02788      PASSED     PASSED
   SOURCE       VRCPTD1     PASSED     PASSED
   SOURCE       Z1          PASSED     PASSED
   SOURCE       Z2L         PASSED     PASSED
   SOURCE       Z300103     PASSED     PASSED
   LIST         FLM01MD1    PASSED     PASSED
   LIST         FLM01MD3    PASSED     PASSED
   LIST         PMR60436    PASSED     PASSED

BUILD MAP RECORDS:                                                 PAGE: 3
                            VERIFY     COMPLETION
     TYPE        MEMBER     STATUS     STATUS
   --------     --------    ------    -------------
   ARCHDEF      FLM01CMD    PASSED     PASSED
   ARCHDEF      FLM01LD3    PASSED     PASSED
   ARCHDEF      PMR60436    PASSED     PASSED
   SOURCE       DTL2        PASSED     PASSED
   SOURCE       FLM01MD3    PASSED     PASSED
   SOURCE       PMR60436    PASSED     PASSED

INTERMEDIATE RECORDS:                                              PAGE: 4
                                          VERIFY  COMPLETION
CU QUAL  CU NAME                    CU TYPE  STATUS    STATUS
-------  -------------------------- -------  ------  -------------
******************** NO RECORDS PROCESSED ************************************
```

Figure 103. Export Report (Part 2 of 2)

Import Utility

The import utility reintroduces the exported SCLM accounting information into the current project after verifying that this data corresponds to the current contents of the SCLM-controlled data sets.

Before using the import utility, verify that the project manager has completed all the steps required to perform the import setup task. Specifically, a copy of the project database from which the items were exported must exist. This means that the PDS members must have been copied. Export VSAM data sets must be defined and allocated for the group in the project into which the data will be imported.

Like the SCLM editor, the import utility verifies authorization codes and prohibits simultaneous updates of members. The group specified to receive the import must be a development group. The import utility also ensures that all the software components to be imported are available and have accounting information. Finally, the import utility verifies that each software component is either new or directly based on the version that exists in the higher group.

The export database is purged after the import is successfully completed.

Figure 104 on page 200 shows the panel that appears when you select Option 7, Import, from the Utilities panel:

```
  Menu  SCLM  Utilities  Jobcard  Help
................................................................................
                         SCLM Export Utility - Entry Panel

Selection criteria:
  Project  . . : PROJ1         Alternate - INT
  Group  . . . . DEV1

  Enter "/" to select option
  /  Replace export data

Output control:
               Ex Sub                   Process . . 1  1. Execute
  Messages . . 3  3   1. Terminal                      2. Submit
  Report . . . 3  3   2. Printer
                      3. Data set       Printer . . _
                      4. None           Volume  . . ______



Command ===> ____________________________________________________________
 F1=Help      F2=Split    F3=Exit      F7=Backward F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 104. SCLM Import Utility (FLMDXI#P)

To import an SCLM group, enter information in each field. The fields for the Import Utility - Entry panel are:

Project
The project that you specified on the SCLM Main Menu. An Alternate field also appears if you specified an alternate project definition.

Group
The development group into which the import is to occur. This group can be any development group defined in the project definition.

Authorization code
The authorization code to be used for all the suitable members to be imported. This field defaults to the authorization code of each member at the time the member is exported. If the authorization code assigned to a member is not in the group being accessed, SCLM does not process the member. Authorization codes cannot contain commas.

Change code
Optionally specify a change code to be added to the change code list of each imported member. Change codes cannot contain commas. If you do not specify a change code, SCLM uses the change code at the time the member is exported.

Mode
Select one of the following modes:

Conditional
To stop the import process if there is a verification failure.

Unconditional
To bypass importation of only those elements that would introduce problems with project integrity.

Report
To perform verification and report generation processing only.

Output control
Specify the destination for messages and report when they are executed (Ex) or submitted (Sub), by entering the corresponding destination number: 1 for Terminal, 2 for Printer, 3 for Dataset, or 4 for None.

Process
You can call the processing part of the Import Utility from the interactive or batch environment by selecting Execute or Submit, respectively. If you request batch processing by selecting Submit, you must specify the job statement information which is used in the JCL generated for batch processing.

For information about using a unique jobname on the jobcard in batch processing, see "Batch Processing" on page 252.

Printer
Specify the printer output class.

Volume
Specify the volume on which SCLM should save data sets.

Import Report example

Figure 105 on page 202 is a sample import report.

The report contains a header indicating that it is an Import Report, which project definition and group are being imported into, and the data set names of the VSAM files containing the information that is being imported. The header is followed by three sections: accounting records, build map records, and intermediate records. The report always contains a section for each type even if no records of that type were processed.

The Verify Status field contains the value FAILED if any of the verification steps failed for the member; otherwise, it contains the value PASSED.

The Completion Status field contains the value PASSED if the record was actually imported; it contains the value FAILED if the import was attempted for a member, but failed; it contains the value NOT ATTEMPTED if the Verify Status field contains the value FAILED because no import of a record is attempted if the record did not pass verification. Certain verification steps will pass only for an Unconditional import; these cases result in a Verify Status of WARNING and the Completion Status for such a member depends on the mode of the import.

If an accounting record has cross-reference records and the accounting record imports successfully, its cross-reference records are also imported. The import report does not include cross-reference records.

```
********************************************************************************
********************************************************************************
**                                                                            **
**                                                                            **
**             SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)               **
**                                                                            **
**                                IMPORT  REPORT                              **
**                                                                            **
**                             2002/08/27   12:42:17
**
**                                                                            **
**                           PROJECT:      BTRANS                             **
**                           ALTERNATE:    BTRANS                             **
**                           GROUP:        DEV1                               **
**                           AUTH. CODE:                                      **
**                           CHANGE CODE:                                     **
**                           MODE:         UNCONDITIONAL                      **
**                                                                            **
**    EXPORT ACCOUNTING FILE: BTRANS.EXPORT.ACCOUNT.DATABASE                  **
**    EXPORT CROSSREF FILE:                                                   **
********************************************************************************
********************************************************************************

 ACCOUNTING RECORDS:                                                PAGE: 1
                              VERIFY     COMPLETION
      TYPE        MEMBER      STATUS     STATUS
     --------     --------    ------    -------------
     ARCHDEF      FLM01CMD    WARNING    PASSED
     ARCHDEF      FLM01LD1    WARNING    PASSED
     ARCHDEF      JTEST02     PASSED     PASSED
     ARCHDEF      PMR60436    PASSED     PASSED
     ARCHDEF      P02788A     PASSED     PASSED
     COPYLIB      BCEWCADA    PASSED     PASSED
     COPYLIB      BCEWCHNG    PASSED     PASSED
     COPYLIB      BCEWFLAG    PASSED     PASSED
     COPYLIB      BCEWPMVT    PASSED     PASSED
     COPYLIB      BRSGEC      PASSED     PASSED
     COPYLIB      BRSSECAU    PASSED     PASSED
     COPYLIB      BRSSVDC1    PASSED     PASSED
     COPYLIB      BRSSVDC2    PASSED     PASSED
     COPYLIB      BRSSZIC1    PASSED     PASSED
     COPYLIB      BRSSZIC2    PASSED     PASSED
     COPYLIB      BRSWINOP    PASSED     PASSED
     COPYLIB      BRSWMTPP    PASSED     PASSED
     COPYLIB      BRSWOLCT    PASSED     PASSED
     COPYLIB      CCOURAN     PASSED     PASSED
     COPYLIB      CPYA0001    PASSED     PASSED
     COPYLIB      DCACCNTN    PASSED     PASSED
     COPYLIB      ISIWLOCK    PASSED     PASSED
     COPYLIB      ISIWLOG     PASSED     PASSED
     COPYLIB      ISIWMCHG    PASSED     PASSED
     COPYLIB      ISIWMERR    PASSED     PASSED
     COPYLIB      ISIWNCGS    PASSED     PASSED
     COPYLIB      ISIWNUME    PASSED     PASSED
     COPYLIB      ISIWNVAL    PASSED     PASSED
     COPYLIB      ISIWSTAT    PASSED     PASSED
     COPYLIB      ISIWTARI    PASSED     PASSED
     COPYLIB      ISIWVALO    PASSED     PASSED
     COPYLIB      PPMWRM22    PASSED     PASSED
     COPYLIB      SYSWRACF    PASSED     PASSED
     COPYLIB      TITWPGMJ    PASSED     PASSED
     LMAP         FLM01LD3    PASSED     PASSED
     LMAP         PMR60436    PASSED     PASSED
     LOAD         FLM01LD3    PASSED     PASSED
     LOAD         PMR60436    PASSED     PASSED
```

Figure 105. Import Report (Part 1 of 2)

```
ACCOUNTING RECORDS:                                                PAGE: 2
                            VERIFY     COMPLETION
     TYPE        MEMBER     STATUS     STATUS
   --------     --------    ------    -------------
   OBJ          FLM01MD1    PASSED     PASSED
   OBJ          FLM01MD3    PASSED     PASSED
   OBJ          PMR60436    PASSED     PASSED
   PNL          VRCPT03     PASSED     PASSED
   SOURCE       CPYRITE     PASSED     PASSED
   SOURCE       DTL2        PASSED     PASSED
   SOURCE       FLM01MD1    FAILED     NOT ATTEMPTED
   SOURCE       FLM01MD3    WARNING    PASSED
   SOURCE       FLM01MD6    WARNING    PASSED
   SOURCE       PMR60436    PASSED     PASSED
   SOURCE       P02788      FAILED     NOT ATTEMPTED
   SOURCE       VRCPTD1     PASSED     PASSED
   SOURCE       Z1          PASSED     PASSED
   SOURCE       Z2L         PASSED     PASSED
   SOURCE       Z300103     PASSED     PASSED
   LIST         FLM01MD1    PASSED     PASSED
   LIST         FLM01MD3    FAILED     NOT ATTEMPTED
   LIST         PMR60436    PASSED     PASSED


BUILD MAP RECORDS:                                                 PAGE: 3
                            VERIFY     COMPLETION
     TYPE        MEMBER     STATUS     STATUS
   --------     --------    ------    -------------
   ARCHDEF      FLM01CMD    PASSED     PASSED
   ARCHDEF      FLM01LD3    PASSED     PASSED
   ARCHDEF      PMR60436    PASSED     PASSED
   SOURCE       DTL2        PASSED     PASSED
   SOURCE       FLM01MD3    PASSED     PASSED
   SOURCE       PMR60436    PASSED     PASSED


INTERMEDIATE RECORDS:                                              PAGE: 4
                                                VERIFY  COMPLETION
CU QUAL  CU NAME                        CU TYPE  STATUS    STATUS
-------  ------------------------------ -------  ------  -------------
********************* NO RECORDS PROCESSED ************************************
```

Figure 106. Import Report (Part 2 of 2)

Audit and Version Utility

The audit and version utility enables you to audit SCLM operations on SCLM-controlled members and create versions of editable members. Using the audit and version utility, you can view the audit information for a member, retrieve a version to a sequential data set not controlled by SCLM, to a partitioned data set not controlled by SCLM, or to an SCLM-controlled development group. This utility also enables you to delete audit and version information from the database.

The project manager controls the audit and version capabilities through the use of macros within the project definition. Audit information is stored in a VSAM data set, and versions of the SCLM members are stored in one or more partitioned data sets allocated for this use.

Attention: The data kept in audit VSAM data sets and the versioning partitioned data sets is for the exclusive use of the audit and version utility. Do *not* edit or alter these data sets without using the audit and version utility or the data may be lost.

Figure 107 on page 204 shows the panel that appears when you select Option 8, Audit and Version, from the SCLM Utilities panel.

```
  Menu  SCLM  Utilities  Help
_______________________________________________________________________________
                 SCLM Audit and Version Utility - Entry Panel

  Option . . 1  1. Versioning and Audit Tracking
                2. Versioning only
SCLM Library:
  Project  . : SLMTEST7
  Group  . . . JPHILP
  Type . . . . ________    ________  ________  ________
  Member . . . ________    (Member name or blank for member list)

Selection date range:
  Date from  . . __________  (Blank or start date for member list)
  Date to  . . . __________  (Blank or end date for member list)

  Enter "/" to select option
  _  Hierarchy view



Command ===> ___________________________________________________________________
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 107. SCLM Audit and Version Utility (FLMVUS#P)

The fields on the SCLM Audit and Version Utility - Entry panel are:

Option
"Versioning and Audit Tracking:" shows all audited actions for the selected members and date range.

"Versioning only:" shows only those entries that have version data associated with them. Includes records for attempts and failures that would otherwise have version data.

Project
The project that you specified on the SCLM Main Menu. An Alternate field also appears if you specified an alternate project definition.

Group
The group for which you want audit and versioning information. The specified group must have an audit VSAM data set defined in the project definition. It must also be defined on an FLMATVER macro in the project definition. If the Hierarchy option is selected, this field will be used to determine the group hierarchy to be searched and the results will include records from the current group and all parent groups.

Type
Specify up to four types of member for which you want the version and audit information displayed or retrieved. The types must be defined on an FLMATVER macro in the project definition.

Member
The member for which you are requesting information. If you leave this field and the Command field blank, SCLM displays the SCLM Version Selection panel. The Member field is optional. A trailing * may be entered in this field to request a selection list according to a pattern match.

Date from
The starting date of the range of dates to search for the specified member. The date must be in the form YYYY/MM/DD. If you specify a member and leave this field blank, SCLM searches from the beginning of the file to the TO date. If you specify a member and leave the "Date from" and "Date to" fields blank, all versions of the member appear.

SCLM verifies that the date you enter is valid and not greater than today's date. The "Date from" field is optional.

Date to
The ending date of the range of dates to search for the specified member. The date must be in the form YYYY/MM/DD. If you specify a member and leave this field blank, SCLM uses the current date as the end date for the search. If you leave the "Date from" and "Date to" fields blank, all versions of the member appear.

SCLM verifies that the date you enter is valid and greater than or equal to the "Date from" value. The "Date to" field is optional.

Hierarchy view
When this option is selected, SCLM searches for audit/versioning records for the current group and for all groups above it in the hierarchy. The current group is determined by the value in the Group field on this panel.

SCLM Version Selection

Using the SCLM Version Selection panel (FLMVSL#P), you can view the audit information and associated accounting information for that version of the member, compare versions of a member or compare the member version with an external data set, delete a version of a member, view the editing history of a version, retrieve a version of a member or view the current contents of a version.

To display the SCLM Version Selection panel, do the following from the SCLM Audit and Version Utility Entry panel:

1. Select "Versioning and Audit Tracking" or "Versioning Only" in the option field.
2. Enter the group name in the Group field.
3. If desired, enter the Type, Member, Date from and Date to information in the appropriate fields.
4. If desired, select the hierarchy option.
5. Press Enter.

   The SCLM Version Selection panel (see Figure 108 on page 205) displays the list of results.

```
  Menu  SCLM  Utilities  Help
_____________________________________________________________________
                           SCLM - Version Selection           Row 1 to 8 of 82

Project  . . . : SLMTEST7

Line Commands:   A Audit Info  C Compare     D Delete      X External Compare
                 H History     R Retrieve    V View

                               Action   Action     Action
S Member   Group    Type       Reason   Date       Time     Userid   V Status
- -------- -------- -------- -------- ---------- -------- -------- - --------
_ JJMSCPR  DEVELOP  ASM      PROMOTE  2002/08/02 16:20:20 JPHILP
_ JJMSCPR  DEVELOP  ASM      PROMOTE  2002/08/02 16:19:37 JPHILP   #
_ JJMSCPR  DEVELOP  ASM      DELETE   2002/08/02 16:17:52 JPHILP
_ JJMSCPR  DEVELOP  ASMLIST  PROMOTE  2002/08/02 16:20:20 JPHILP
_ JJMSCPR  DEVELOP  ASMLIST  PROMOTE  2002/08/02 16:19:39 JPHILP
_ JJMSCPR  DEVELOP  NCAL     PROMOTE  2002/08/02 16:20:21 JPHILP
_ JJMSCPR  DEVELOP  NCAL     PROMOTE  2002/08/02 16:19:39 JPHILP
_ JJMSCPR  DEVELOP  OBJ      PROMOTE  2002/08/02 16:20:21 JPHILP


Option ===> _____________________________________________ Scroll ===> PAGE
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 108. SCLM Version Selection Panel (FLMVSL#P)

Use the SCROLL commands or the LOCATE command to scroll the list.

The fields for the Version Selection panel, shown in Figure 108 on page 205, are:

Member
The names of the members matching the selection criteria on the SCLM Audit and Version Utility - Entry panel that have audit and version information.

Group
The name of the group you specified on the SCLM Audit and Version Utility - Entry panel.

Type
The types of the members matching the selection criteria on the SCLM Audit and Version Utility - Entry panel.

Action Reason
The action that was performed against the specified member. Valid values include:

- BUILD
- BLDDEL
- DELETE
- EXT LIB
- FREE
- IMPORT
- LOCK
- PROMOTE
- STORE
- UNLOCK
- UPTATHCD (update authorization code)
- UPTCHGCD (update change code)
- UPTUENTY (update user entry)

Action Date
The date the action listed in the Action Reason field occurred.

Action Time
The time the action listed in the Action Reason field occurred.

Userid
The user ID of the person who performed the action.

V
Indicates, using a hash symbol (#), whether a version of the member exists.

Status
Indicates the status of the line command. Possible values are:

- *SELECT
- *DELETED
- *FAILED
- *ERROR
- RETRVOLD
- RETRVNEW

To the left of each member listed is a space for entering a line command. You can enter multiple commands on the panel as long as the commands do not conflict. All requests are handled in succession unless an error occurs. If an error occurs, the selection list indicating the error reappears. You must correct the error before further processing can occur.

The available line commands are as follows:

A
Display the audit information for the member.

When you enter the A line command beside a member name, the SCLM Audit/Version Record panel appears, as shown in Figure 111 on page 208, giving you the information recorded for that member. From here, you can display the accounting information.

C

Display the version in the SCLM Audit and Version Utility - Compare panel.

When you enter the C line command beside a member version, SCLM displays the selected version information in the SCLM Audit and Version Utility - Compare panel, along with a subset of versions (not audit records) from the initial version selection results, where the Type and Member are the same. For more information, see "SCLM Version Compare" on page 209.

The C command can only be entered for member versions (not audit records).

D

Delete the audit record in the VSAM audit data set and delete the versioned member in the partitioned data set.

When you enter the D line command beside a member name, SCLM deletes the audit record and the corresponding versioned member, if one exists. The Status field displays the word "Deleted", indicating that the operation completed successfully.

X

Display the version in the SCLM Audit and Version Utility - External Compare panel.

When you enter the X line command beside a member version, SCLM displays the selected version information in the SCLM Audit and Version Utility - External Compare panel, in which you can specify the external data set to be used in the comparison. For more information, see "External Compare" on page 210.

H

Display the history of editing changes made between the selected version and the current version. The Key column indicates whether each line has changed and if so, in which version.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
------------------------------------------------------------------------
VIEW       SYS02225.T133058.RA000.USERID.VHIST.H01        Columns 00001 00072
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000001                  Version History
000002           Changes since, but not including Version 2
000003
000004 CURRENT 0      02/04/17  12:49:34.19  JPHILP
000005 VERSION 1      02/04/12  12:32:59.49  JPHILP
000006 VERSION 2      02/04/12  11:54:23.25  JPHILP
000007
000008 |------Key-----||----------Description----------|
000009 Ixxxxxx         Inserted into Version xxxxxxx
000010         Dxxxxxx Deleted from Version xxxxxxx
000011  (blank)        Unchanged since current version
000012
000013 |-------Key-----|-----------------------------------Source-----------
000014                          TITLE 'JJMSCPR  - COPYRIGHT CODE
000015                 */04*****************************************************
Command ===> ____________________________________________ Scroll ===> PAGE
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Figure 109. Audit and Version View panel (ISREDDE2) - sample data with history

The H command can only be entered for member versions (not audit records).

R

Display the version in the SCLM Audit and Version Utility - Retrieve panel.

When you enter the R line command beside a member version, SCLM displays the selected version information in the SCLM Audit and Version Utility - Retrieve panel, in which you can specify the data set into which the version will be retrieved.

The R command can only be entered for member versions (not audit records). For more information, see "Retrieve" on page 212.

V

Display the current contents of the selected member version, using the SCLM VERRECOV service.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
VIEW        USERID.VERBROWS.SLV89QK4                      Columns 00001 00072
****** ***************************** Top of Data ******************************
==MSG> -Warning- The UNDO command is not available until you change
==MSG>           your edit profile using the command RECOVERY ON.
000001          TITLE 'JJMSCPR  - COPYRIGHT CODE                  '
000002 */04*********************************************************************/
000003 */*                                                                     */
000004 */*                                                                     */
000005 */*  OCO Source Materials                                               */
000006 */*                                                                     */
000007 */*  5696-234                                                           */
000008 */*                                                                     */
000009 */*  (C) Copyright IBM Corp. 1992,2000                                  */
000010 */*                                                                     */
000011 */*  The source code for this program is not published or               */
000012 */*  otherwise divested of its trade secrets, irrespective of           */
000013 */*  what has been deposited with the U.S. Copyright Office.            */
000014 */*                                                                     */
000015 */*********************************************************************/
Command ===> _____________________________________________ Scroll ===> PAGE
 F1=Help      F2=Split     F3=Exit      F5=Rfind     F6=Rchange   F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Figure 110. Audit and Version View panel (ISREDDE2) - sample data

The V command can only be entered for member versions (not audit records).

SCLM Audit and Version Record

If you enter 'A' to display the SCLM Audit and Version record, the SCLM Audit/Version Record panel shown in Figure 111 on page 208 appears.

```
                        SCLM - Audit/Version Record

Project  . : SLMTEST7
Audit data:
  Group  . . . . . : DEVELOP         Calling service . . : PROMOTE
  Type . . . . . : ASM              Action Taken  . . . : PUT
  Member . . . . . : JJMSCPR         Action Result . . . : COMPLETE
  Audit Date . . . . : 2002/04/12      Fail Message  . . . :
  Audit Time . . . . : 11:54:23.25
  Userid . . . . . : JPHILP
  SCLM Change Date . : 2002/04/12
  SCLM Change Time . : 11:53:10
Version data:
  Data Set . . . . . : SLMTEST7.DEVELOP.ASM.VERSION
  Member . . . . . : JJMSCPR         Request format  . . : DELTA
  Change Date  . . . : 2002/04/12      Current format  . . : DELTA
  Change Time  . . . : 11:54:24

Enter "/" to select option;
_ Display Accounting Information

Command ===> ____________________________________________________________________
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F12=Cancel
```

Figure 111. SCLM Audit/Version Record Panel (FLMVBA#P)

The fields for the panel shown in Figure 111 on page 208 are:

Project

The project that you specified on the SCLM Main Menu. An Alternate field also appears if you specified an alternate project definition.

Group
The group for which the accounting information appears.

Type
The type for which the accounting information appears.

Member
The member for which the accounting information appears.

Audit Date
The date the audit was performed.

Audit Time
The time the audit was performed.

Userid
The userid of the person who caused the audit record to be created.

SCLM Change Date
The date the member was last edited.

SCLM Change Time
The time the member was last edited.

Data Set
The name of the PDS where the version data, if any, for this record is stored. This name is always present, whether or not version data exists.

Member
The name of the member in which version data is stored, if this record has version data. This field is blank if there is no version data.

Change Date
The date the versioned member was written.

Change Time
The time the versioned member was written.

Calling Service
The service that SCLM is running at the time; for example, BUILD, PROMOTE, STORE, LOCK, or DELETE.

Action Taken
The function that causes the audit / version to be taken.

For example, EDIT causes a SAVE. EDIT is the calling service and SAVE is the action taken. The action could be LOCK, DELETE, MIGRATE, and so on. The calling service and the action taken could be the same. For example, the BUILD service could cause the BUILD action to take a version.

Action Result
Indicates the status of the action taken.

Fail Message
Indicates a failure. This field contains the message number of the failing message.

If the action result is COMPLETED, you can display the related accounting information. Enter S to select this option (located at the bottom of the SCLM Audit / Version Record panel). See Figure 75 on page 164 for an example of the Accounting Record panel.

SCLM Version Compare

If you enter 'C' to select a version to be compared with other member versions, the SCLM Audit and Version Utility - Compare Panel, shown in Figure 112 on page 210, is displayed. Information about the selected version is shown in the top section of the panel. The bottom section of the panel lists all the matching versions of the member that were included in the initial version selection results. Member versions are considered matching when the Member and Type fields are the same. If the Hierarchy View option was selected on the SCLM Audit and Version Utility - Entry Panel, member versions in different groups appear on this list and can be compared.

```
FLMVSC#P        SCLM Audit and Version Utility - Compare Panel   Row 1 from 32

SCLM Library:
  Version  . . : SLMTEST7.DEVELOP.ASM(JJMSCPR)
  Version Date : 2002/04/12
  Version Time : 11:54:23
                                                          Sequence
  Compare Type               Listing Type                 Numbers
  1  1. File                 1  1. Delta                  1  1. BLANK
     2. Line                    2. CHNG                     2. SEQ
     3. Word                    3. Long                     3. NOSEQ
     4. Byte                    4. OVSUM                    4. COBOL

Listing DS Name

                              Action   Action     Action
S Member   Group    Type      Reason   Date       Time     Userid   V
- -------- -------- --------- -------- ---------- -------- -------- -
_ JJMSCPR  DEVELOP  ASM       PROMOTE  2002/06/17 13:56:07 JPHILP   #
_ JJMSCPR  DEVELOP  ASM       PROMOTE  2002/04/17 12:49:34 JPHILP   #

Command ===> _____________________________________________ Scroll ===> PAGE
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F12=Cancel
```

Figure 112. SCLM Audit and Version Utility - Compare Panel (FLMVSC#P)

The fields for the panel shown in Figure 112 on page 210 are:

Version
Displays the full name of the selected member version.

Version Date
The date on which the selected member version was written.

Version Time
The time at which the selected member version was written.

Compare Type
Specifies the granularity of the comparison, ranging from entire member to member (`File`) comparison down to single `Byte` differences. `Line` compare is useful for source data. `Word` compare is most useful for text data.

Listing Type
Specifies the context scope of the listing report. You can get a listing with summary information only (`OVSUM`), single line differences between files (`Delta`), differences plus or minus the five unchanged lines before and after changed lines (`CHNG`), or a listing that includes all of the lines in both files (`Long`).

Sequence numbers
Specifies whether sequence numbers in the compared files are to be ignored or treated as data. Choose `SEQ` to ignore differences in standard sequence number columns 72 through 80 for FB LRECL 80 members. Choose `NOSEQ` to treat *all* columns in the files as data. Choose `COBOL` to ignore differences in columns 1 through 8 of the data. Choose `Blank` to cause SuperC to ignore standard sequence number columns if the data set is FB 80 or VB 255. Otherwise, the comparison processes these columns as data.

Listing DS Name
The data set into which the compare listing is written. You can preallocate this data set, or let ISPF create one for you. If this data set is partitioned, you must specify a member name.

To the left of each version listed is a space for entering the S line command. This command selects the version against which you want to compare the currently selected member version. You can only select one of the listed versions.

Note: More sophisticated comparisons can be done using the ISPF Option 3.13, SuperC Compare Utility.

External Compare

If you enter 'X' to select a version to be compared with an external data set, the SCLM Audit and Version Utility - External Compare Panel, shown in Figure 113 on page 211, is displayed. Information about the selected version is shown in the top section of the panel.

```
              SCLM Audit and Version Utility - External Compare

SCLM Library:
  Version  . . : SLMTEST7.DEVELOP.ASM(JJMSCPR)
  Version Date : 2002/04/12
  Version Time : 11:54:23

Compare Version with:
  _  SCLM Group . . ________
  _  ISPF Data Set  ____________________________________________________
       Member    ________

                                                          Sequence
  Compare Type               Listing Type                 Numbers
  1  1. File                 1  1. Delta                  1  1. BLANK
     2. Line                    2. CHNG                     2. SEQ
     3. Word                    3. Long                     3. NOSEQ
     4. Byte                    4. OVSUM                    4. COBOL

Listing DS Name   ____________________________________________________________

Command ===> ____________________________________________________________
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F12=Cancel
```

Figure 113. SCLM Audit and Version Utility - External Compare Panel (FLMVSX#P)

The fields for the panel shown in Figure 113 on page 211 are:

Version
Displays the full name of the selected member version.

Version Date
The date on which the selected member version was written.

Version Time
The time at which the selected member version was written.

SCLM Group
To compare a version against the member store within SCLM, place a "/" against the SCLM Group and specify the group. This will search for the first occurrence of the selected member in the hierarchy starting at this group.

ISPF Data Set
To compare a version against an ISPF data set place a "/" against the ISPF Data Set and specify the data set name. This will search for the first occurrence of the member in the data set.

Member
If you have chosen to compare a version against an ISPF data set, you can specify the member to be used in the comparison. If this field is left blank, the first occurrence of the selected member in the data set will be used.

Compare Type
Specifies the granularity of the comparison, ranging from entire member to member (`File`) comparison down to single `Byte` differences. `Line` compare is useful for source data. `Word` compare is most useful for text data.

Listing Type
Specifies the context scope of the listing report. You can get a listing with summary information only (`OVSUM`), single line differences between files (`Delta`), differences plus or minus the five unchanged lines before and after changed lines (`CHNG`), or a listing that includes *all* of the lines in both files (`Long`).

Sequence numbers
Specifies whether sequence numbers in the compared files are to be ignored or treated as data. Choosing SEQ means to ignore differences in standard sequence number columns 72 through 80 for FB LRECL 80 members. Choosing NOSEQ means to treat *all* columns in the files as data. The COBOL selection means to ignore differences in columns 1 through 8 of the data. Choosing Blank causes SuperC to ignore standard sequence number columns if the data set is FB 80 or VB 255. Otherwise, the comparison processes those columns as data.

Listing DS Name
The data set into which the compare listing is written. You can preallocate this data set, or let ISPF create one for you. If this data set is partitioned, you must specify a member name.

Note: The SCLM Group and ISPF Data Set options are mutually exclusive, that is, you can only choose one of these options. If a "/" is placed in both options, a message will state that the ISPF Data Set option is invalid.

Retrieve

If you enter 'R' to select a version to be retrieved, the SCLM Audit and Version Utility - Retrieve Panel, shown in Figure 114 on page 212, is displayed. Information about the selected version is shown in the top section of the panel.

```
  Menu  SCLM  Utilities  Help
──────────────────────────────────────────────────────────────────────
FLMVSR#P       SCLM Audit and Version Utility - Retrieve Panel

SCLM Library:
  Version  . . : SLMTEST7.DEVELOP.ASM(JJMSCPR)
  Version Date : 2002/04/12
  Version Time : 11:54:23

SCLM retrieve group and type:
  To Group . . . ________        Authorization code . . ________
  To Type  . . . ________        (Defaults to auth code from audited member)

Other Data set:
  Data Set Name  ____________________________________________________________




Command ===> ____________________________________________________________
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 114. SCLM Audit and Version Utility - Retrieve Panel (FLMVSR#P)

The fields for the panel shown in Figure 113 on page 211 are:

Version
Displays the full name of the selected member version.

Version Date
The date on which the selected member version was written.

Version Time
The time at which the selected member version was written.

To Group
The SCLM Group into which the version is to be retrieved.

To Type
The SCLM Type into which the version is to be retrieved.

Auth Code
The Authorization code used in the retrieval process. If left blank, this defaults to the Authorization code from the selected member version.

Data Set Name
The ISPF data set into which the version is to be retrieved. If the data set is a PDS, a member must be specified.

Note: If a data set name is specified in this field, the To Group and To Name fields are ignored.

When you retrieve more than one member into a sequential data set, each member after the first is copied over the previous member. To retrieve more than one member to a sequential data set, copy the first member to another data set before retrieving a second member. We recommend that you use a partitioned data set if you intend to copy more than one member.

SCLM will not allow you to retrieve a second version of the same member but you can retrieve a version of a different member. To retrieve a second version of the same member you must first return to the SCLM Audit and Version Utility Entry panel and then come back to the SCLM Version Selection panel.

Note: When you retrieve the most recent version of a source member into a development group of the hierarchy, the accounting data and ISPF statistics match those of the member that is already in the hierarchy. Therefore, outputs are not produced when the member is built because the outputs that are already in the library are current.

In addition, when the recovered member is promoted to the level where the member resides, the existing member is not overwritten. If the content of the existing member has been corrupted and it is important to replace that member, you must save the member in the hierarchy after it is recovered. You can save the member using SCLM edit, migrate in forced mode, or the SAVE service.

If you perform a retrieve using the To Group and To Type fields, it may result in predecessor verification errors when promoting the member because both the member and its accounting information are restored. Using the Other Data set field restores only the member, but not its accounting information. If you do not plan to make further changes to the restored member, enter the SAVE command explicitly to force SCLM to generate new accounting information. Entering the END command with the AUTOSAVE option in the edit profile does not cause SCLM to generate new accounting information and results in a predecessor verification error.

Delete from Group Utility

You can use the Delete from Group utility to delete database components associated with a specified group. You can delete a member or members and all associated SCLM accounting information, including accounting records, build map records, cross-reference records, and intermediate records. You can further specify whether you want everything deleted, only build outputs, only accounting information and build map records, or only build map records. You can also specify that nothing actually be deleted but a deletion report be generated.

The Delete from Group utility does not delete members that have no accounting information.

Figure 115 on page 214 shows the panel that is displayed when you select Option 9, Delete from Group, from the Utilities panel.

```
  Menu  SCLM  Utilities  Jobcard  Help
──────────────────────────────────────────────────────────────────────────────
                 SCLM Delete from Group Utility - Entry Panel

Delete from Group Input:
  Project  . : PDFTDEV
  Group  . . . SBURNF__
  Type . . . . MSGSRCE__  (Pattern may be used)
  Member . . . _________  (Pattern may be used)

  Delete Flag . . _  1. Build map      Delete Mode . . 2  1. Execute
                     2. Account                           2. Report
                     3. Text
                     4. Output
Output control:
               Ex Sub                  Process . . . . 1  1. Execute
  Messages . . 3  3    1. Terminal                        2. Submit
  Report . . . 3  3    2. Printer
  Listings . . 3  3    3. Data set     Printer . . . . H
                       4. None         Volume  . . . . ______

Command ===>
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 115. SCLM Delete from Group Utility (FLMDDG#P)

To delete information from an SCLM group, you must enter information for each field. The fields for the Delete from Group Utility - Entry panel are:

Project
The project specified on the SCLM Main Menu. This field is display only. An Alternate field also appears if you specified an alternate project definition.

Group
The group for which information is to be deleted. Delete from Group only works on groups defined to the project. This field is required. There are no default values.

Type
The type from which information is to be deleted. You can use patterns for the type you want processed. See "Specifying selection criteria" on page 181 for details. Delete from Group only works on types defined to the project.

Member
The name or pattern of the members and SCLM information to be deleted. Only members that have accounting information are deleted. You can use patterns for the member name. See "Specifying selection criteria" on page 181 for details.

Delete Flag
The indicator of the type of data to be deleted.

Build map
All build map records that match the pattern are deleted.

Account
All accounting records, cross-reference records, intermediate records, and build map records that match the pattern are deleted. The accounting type will not be checked.

Text
All accounting records, cross-reference records, intermediate records, build map records, intermediate code, and text members that match the pattern are deleted. The accounting type will not be checked.

Output
All build map records, intermediate records and code, and all non-editable accounting records, their cross-reference records and associated text members that match the pattern are deleted.

Editable accounting records, their cross-reference records or associated text members are not deleted.

Delete Mode
The indicator for the action performed by the Delete from Group. Select one of the following values:

Execute
All members that match the selection criteria for the specified Delete Flag are deleted.

Report
No deletion will occur; contents of what would, upon execution, be deleted for the specified selection criteria and Delete Flag are reported. Report is always be the default whenever this panel appears. Even after you execute a delete from group, the mode is changed to Report.

To delete members, update authority to the hierarchy data sets containing the members is required, even if the Delete from Group utility is run in REPORT mode.

Output control
Specify the destination for messages and the report when they are executed (Ex) or submitted (Sub), by entering the corresponding destination number: 1 for Terminal, 2 for Printer, 3 for Dataset, or 4 for None. A listing data set will not be allocated when the Delete Mode is Report, even though Dataset is specified for the Listings field.

Process
You can call the processing part of the delete from group utility from the interactive or batch environment by selecting Execute or Submit, respectively. If you request batch processing by selecting Submit, you must specify the job statement information which is used in the JCL generated for batch processing.

For information about using a unique jobname on the jobcard in batch processing, see "Batch Processing" on page 252.

Printer
Specify the printer output class.

Volume
Specify the volume on which SCLM should save data sets.

Delete from Group Report example

Figure 116 on page 216 shows a sample Delete Group report.

The report contains a header indicating that it is a Delete Report, which project definition and group are specified, the type and member selection criteria, and the delete flag and mode. The header is followed by three sections: members, build maps, and Ada intermediate code. The report always contains all of these sections even if there is no activity to report for a section. Output members are denoted by an asterisk (*) at the beginning of the report line.

The VERIFY STATUS field contains the value PASSED unless the delete routine was unable to verify the record for one of the following reasons:

- User has no update authority
- Member has nonblank access key
- Error reading the record

The COMPLETION STATUS field contains the value PASSED if the member was actually deleted. The field contains NOT ATTEMPTED if the verification failed or the delete from group was run in REPORT MODE only. The field contains FAILED if an error occurred during the execution of the deletion. The field contains WARNING if the text member or intermediate code did not exist. The accounting record is still deleted.

Although cross-reference records are deleted, there is no section explicitly for them in the Delete Group report. If the accounting record is successfully deleted, its cross-reference records, if any, are also deleted.

The report header indicates that it relates to the Delete from Group utility. The header also shows which project definition and group are specified, the type and member selection criteria, and the delete flag and mode.

```
****************************************************************
****************************************************************
**                                                            **
**                                                            **
**       SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)     **
**                                                            **
**                     DELETE GROUP REPORT                    **
**                                                            **
**                     2000/03/26   13:30:39                  **
**                                                            **
**                     PROJECT:      PROJ1                    **
**                     ALTERNATE:    PROJ1                    **
**                     GROUP:        USER1                    **
**                     TYPE:         *                        **
**                     MEMBER:       *                        **
**                     FLAG:         TEXT                     **
**                     MODE:         REPORT                   **
****************************************************************
****************************************************************

 MEMBERS:                                              PAGE 1
                                VERIFY   COMPLETION
      GROUP    TYPE     MEMBER    STATUS   STATUS
    --------   ------   -------  ------    ----------
    USER1      SOURCE   ASM1     PASSED    NOT ATTEMPTED
    USER1      SOURCE   ASM2     PASSED    NOT ATTEMPTED
    USER1      SOURCE   PASMAIN  PASSED    NOT ATTEMPTED
   *USER1      LISTING  PASMAIN  PASSED    NOT ATTEMPTED
   *USER1      LMAP     PASMAIN  PASSED    NOT ATTEMPTED
   *USER1      LOAD     PASMAIN  PASSED    NOT ATTEMPTED
   *USER1      OBJ      PASMAIN  PASSED    NOT ATTEMPTED
    USER1      SOURCE   PASCPGM  PASSED    NOT ATTEMPTED
    USER1      SOURCE   PSCINCL1 PASSED    NOT ATTEMPTED
    USER1      SOURCE   PSCINCL2 PASSED    NOT ATTEMPTED
    USER1      SOURCE   PSCINCL3 PASSED    NOT ATTEMPTED
    USER1      SOURCE   SCRIPTHL PASSED    NOT ATTEMPTED
    USER1      SOURCE   SCRIPT1  PASSED    NOT ATTEMPTED

 BUILD MAPS:                                           PAGE: 2
                                VERIFY   COMPLETION
      GROUP    TYPE     MEMBER    STATUS   STATUS
    --------   ------   -------  ------    ----------
    USER1      SOURCE   PASCMAIN PASSED    NOT ATTEMPTED

 ADA INTERMEDIATE CODE:                                 PAGE:  3
                                               VERIFY   COMPLETION
 GROUP    CU QUAL  CU NAME        CU TYPE  STATUS     STATUS
--------  -------- -------------  -------  ------   -----------
********************* NO RECORDS PROCESSED **********************
```

Figure 116. Delete Group Report

Package Backout Utility

The Package Backout utility enables you to back up and recover non-editable types, using a backup group controlled within SCLM. The backout process restores an executable environment by promoting the previously backed up modules from the backup group. Source members are recoverable through versioning, using SCLM services and administration procedures external to the Package Backout processes.

The term "package" refers to an SCLM architecture member that is used during the build and promote processes within SCLM. This architecture member defines the modules/ARCHDEF members that are promoted using include or change code parameters.

The libraries that contain packages are determined by using the ISAPACK=Y flag on the FLMTYPE macro within the project definition. If an architecture member is promoted from a library which does not have an ISAPACK=Y flag then the package backout process will not be invoked and no modules will be backed up.

To recover source or editable types, you must implement versioning at the group being targeted for package backout.

During package backout, the Copy phase of the promote process is triggered, to allow DB2 BINDs to be performed against any recovered DBRMs, and the Purge phase is triggered to delete the backed-up modules. Promote copy and purge exit processing is also invoked during the package backout process. This ensures the integrity of backed out load modules and ensures that any other exit processing that is in place during a normal copy or purge promote process is maintained.

Package Backout involves two phases: Backup and Restore.

The backup phase occurs during a Promote process (see Figure 117 on page 218). For each member of a package marked for backout, it:

1. Copies the old members to the existing backup data set.
2. Saves the package details into the Package Details file.
3. Allows the promote to continue.

The restore phase occurs when requested by the user (see Figure 118 on page 220). Restore promotes the old members back to the original group.

Package backout enables users to quickly restore an executable environment. The backout process restores the previously backed up package modules through the promote process from the backup group.

Once the immediate problem has been resolved in the executable environment, the user must apply the changes to the source using the normal development process. Use version retrieval to retrieve the version of the source corresponding to the backed out member into a development group for editing, or make the change in the existing copy of the member in the hierarchy.

The Package Details file holds the date and time details of both the backed up members and the editable members in the package, so these can be used as input to determine the appropriate versions to be recovered.

A package has the status of "BACKEDUP" when it is initially backed up, and "RESTORED" after a package-level restore is performed.

A similar status is retained against the backed-up member, showing either "BACKEDUP", or "RESTORED" if it is restored using a member-level restore.

To be able to recover source parts using Package Backout, versioning must be implemented for any editable types (such as source) that are promoted to a level at which Package Backout has been implemented.

Note: Package Backout cannot control backout of editable types.

To backout editable types, use versioning to restore the relevant member of the -1 version using the Other Data set field.

Backup phase

Figure 117 on page 218 shows the backup phase.

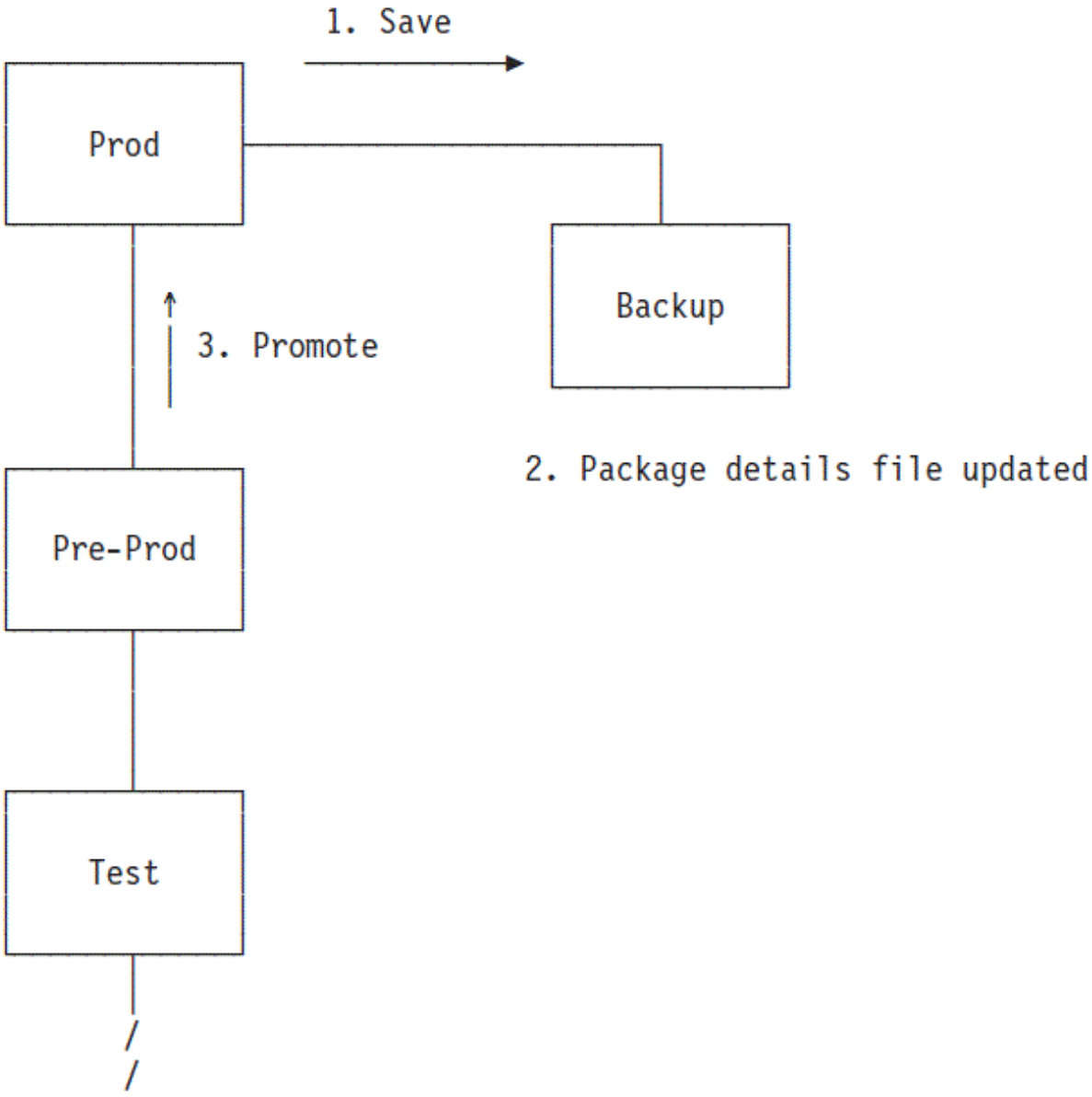


Figure 117. Package Backout—Backup Phase

Package details are maintained as members of the Package Details file PDS. This PDS needs to be defined by the SCLM Administrator.

The SCLM type for this PDS is nominated using the FLMTYPE macro, for example:

```
BACKUP   FLMTYPE PACKFILE=Y
```

These PDS members hold the package backout information, such as:

- Package status
- Group
- Type
- Member
- Old member timestamp
- New member timestamp
- Timestamp when backed out

- Member status
- Member-level selection flag

Accounting records of the non-editable types are not saved back to the backup level, nor restored to the higher group.

Any subsequent package promotion that involves the same type/member invalidates the ability of the member from the original package to be restored, and causes the member to be overwritten in the backup data set. The member cannot be restored, because the physical timestamp of the member differs from the timestamp in the Package Details file. The Package Backout routines check timestamps dynamically, to ascertain if the member is still eligible for restore processing.

A promote that involves a package that can be backed out can be restarted. If it is, the package members being backed up are simply recopied during the restarted promote.

Specifying the parameter REUSEDAY=nnnn forces SCLM to check the package date in the Package Details File for the package being promoted. If this package is not younger than the REUSEDAY value, then the package details member is deleted. If it is younger than the value then SCLM reuses the package. With Package reuse, if the module is being promoted again it overwrites the older backed up version of that module.

Restore phase

Figure 118 on page 220 shows the restore phase.

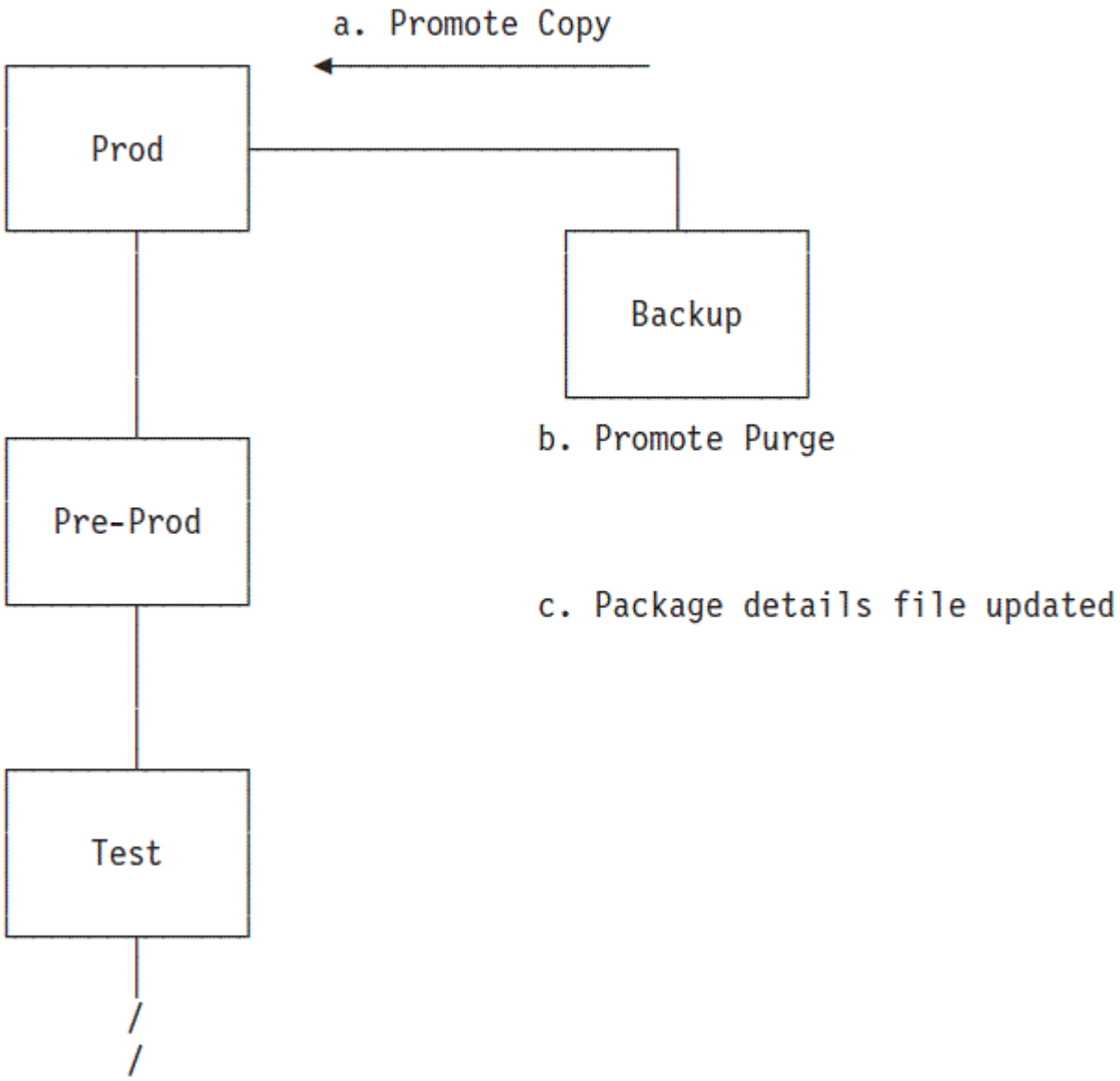


Figure 118. Package Backout—Restore Phase

The restore is limited to non-editable types. The details of all members, both editable and non-editable, are recorded in the Package Details file.

After recovery of the non-editable members, the build-map of the related editable member is in an inconsistent state. The SCLM Administrator must now act to recover the source into a development group. From this group edit compare can be used to merge any desired changes from intermediate levels, and the member can be fixed and then built, tested, and promoted through the normal development process.

The ability of the members in the package to be backed out is dynamically assessed before any backout operation. This status is established by checking the statistics timestamp of the old and new members, and comparing them to the timestamps recorded in the package details file. Any differences invalidate the member for restoration.

When a restore is requested, the equivalent of normal promote processing is performed from the backup group, with both the Promote Copy and Purge phases.

During recovery, the member in the backup library is purged. This is because once a member has been restored, it cannot be restored again.

You can choose to back out either the whole package, or one or more individual members.

By default, member-level restore is deactivated. To activate this, add the parameter BKMBRLVL=Y to the FLMGROUP macro.

If a promote of a backout package completes successfully, and the same package name is used again in a package, it will overwrite the details of the previous package and members.

Cleanup of backed-up packages can be performed online or through a batch job. The cleanup procedure purges package details from the Package Details file, and deletes related members in the backup data sets that have not already been restored and purged.

Package functions

Figure 119 on page 221 shows the panel that is displayed when you select Option 10, Package Functions, from the Utilities panel.

```
Menu  SCLM  Utilities  Help
_____________________________________________________________________________

                      SCLM Package Functions - Entry Panel

 SCLM Library:
   Project  . : SLMTEST7
   Group  . . . TEST____
   Type . . . : PACKAGE
   Member . . . ________    (Blank or pattern for member selection list)

 Package Member/Type Filter
   Member . . . ________    (Blank or pattern)
   Type . . . . ________    (Blank or pattern)

 Options
 /  Match backed up members only




 Command ===> ___________________________________________________________________
  F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
 F10=Actions  F12=Cancel
```

Figure 119. SCLM Package Functions Utility (FLMPF#P)

You use this panel to specify the package library and select the packages you want to work with. The fields for the Package Functions - Entry panel are:

Project
The project that was specified on the SCLM Main Menu. This field is display only. An Alternate field also appears if you specified an alternate project definition.

Group
The name of the backup group. This field is required. There are no default values.

Type
The type from which information is to be deleted. The type value is determined by the current project definition.

Member
The name or pattern of the members to be processed. If the member name is left blank or a pattern is entered, a package selection list is displayed. You can use patterns for the member name. See “Specifying selection criteria” on page 181 for details.

Package Member/Type Filter
Filtering is performed when a pattern is specified in the filter member and/or type fields. SCLM searches each package to see if it contains a matching member and type value. If no type field is entered then the filtering will match the member only. If no member is specified the filtering will match the type only.

Note: The status of the package on the subsequent package list will be the status of the first matching member if the member status is in error.

Match backed up members only
Use the match option to restrict the matching of members within a package to only backed up members. Set the match option to blank to match on all members referenced within a package.

Figure 120 on page 222 shows the packages available for backout at a given level.

```
Menu  SCLM  Functions  Utilities  Help
------------------------------------------------------------------------
 Package List  SLMTEST7.BACKUP.PACKAGE                      Row 1 to 12 of 12

 S=View      D=Delete     R=Restore

     Package  Status   Member Date/Time    Restored Date/Time
 _   JTEST01  BACKEDUP 2002/10/17 12:50:15
 _   TSTPACK5 BACKEDUP 2002/10/10 22:18:37 2002/10/10 22:18:41
 _   TSTPETE8 NOBACKUP 2002/10/10 22:09:17 2002/10/10 22:09:20
 _   TSTPETE7 BACKEDUP 2002/10/10 22:05:20 2002/10/10 22:05:24
 _   TSTPETE4 BACKEDUP 2002/10/10 02:09:12 2002/10/10 02:09:15
 _   TSTPACK6 BACKEDUP 2002/10/10 00:32:30 2002/10/10 00:32:36
 _   TESTSLC3 NOBACKUP 2002/08/28 04:35:23 2002/09/11 23:12:18
 _   TESTSLC2 RESTORED 2002/08/27 04:06:32 2002/09/05 02:10:28
 _   TESTSLC  BACKEDUP 2002/08/27 02:42:21
 _   TESTPKG3 BACKEDUP 2002/08/19 00:58:58
 _   TESTPKG2 RESTORED 2002/08/14 03:46:53 2002/09/16 00:41:28
 _   TSTPETE1 BACKEDUP 2002/08/13 23:38:20
 ******************************* Bottom of data
********************************

 Command ===> ____________________________________________________________
  F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
 F10=Actions  F12=Cancel
```

Figure 120. SCLM Package List Panel (FLMPFL#P)

To the left of each package listed is a space for entering a line command:

S
Display the list of members in the package. For more information about the options available from the SCLM Package Member Details panel, see “Package Member Details” on page 223

R
Restore the selected package.

When you enter the R line command beside a member name, SCLM attempts to restore the selected package. If the operation completes successfully, the Status field displays the word "Restored" and the current date and time appears in the Restored Date/Time field.

Note: Only packages whose status is BACKEDUP can be restored with this command. The target members and backup members associated with this package are validated before the restore process is performed.

If an error occurs the status changes to indicate one of the following values:

INVTARG
At least one of the target members to be restored has a different date to the target member at the time the package was created.

OBSOLETE
At least one package member has been superseded by another package.

D
Delete the package and its associated backup members.

When you enter the D line command beside a package name, SCLM deletes the package and its corresponding backup members. If the operation completes successfully, the Status field displays the word "Deleted".

Package Member Details

When you enter the S line command beside a package name, the SCLM Package Member Details panel lists the members contained in the package.

```
Menu  SCLM  Functions  Utilities  Help
_____________________________________________________________________
 Member List : SLMTEST7.BACKUP.PACKAGE(JTEST01)                Row 1 to 5 of 5

 Enter primary command R to perform member level restore
 Enter line command S to toggle member selection
 Sel  Member   Rec Status   Type     Member Date/Time     Restored Date/Time
 _    JTEST01      INITIAL  ARCHDEF
 _    JJMSCPR               ASM      2002/04/12 12:32:15
 _    JJMSCPR      BACKEDUP ASMLIST  2002/04/12 12:32:43
 _    JJMSCPR      BACKEDUP NCAL     2002/04/12 12:32:00
 _    JJMSCPR      BACKEDUP OBJ      2002/04/12 12:32:42
 ******************************* Bottom of data ********************************









 Command ===> _____________________________________________________________
  F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
 F10=Actions  F12=Cancel
```

Figure 121. SCLM Package Member Details Panel (FLMPML#P)

The fields for the Package Member Details panel are:

Sel
Enter the S line command next to a member to select it for member level restore. The member status must be BACKEDUP, NEWBKUP, or MODBKUP. If the status is not one of these values, the line command is ignored. If a member is already selected, entering the S line command removes it from the selection.

Member
The name of the members to be processed.

Rec
An '*' in this column indicates members that are selected.

Status
Indicates the backup status of the member. Possible values are:

INITIAL
The target member did not exist before promotion. Therefore, this member can not be backed up or restored.

INVTARG
The target member has been changed since this package backout member was created. Therefore, this member cannot be restored.

OBSOLETE
The member in the backup library is NOT the member referenced in this package. Therefore, this member cannot be restored.

RESTORED
This package member has been restored. It cannot be restored again.

BACKEDUP
The member has been backed up and if member level restore is available it can be restored, or alternatively if the package status is also BACKEDUP then it can be restored with a package restore.

NEWBKUP
The package has been reused and this member has not previously been backed up for this package.

MODBKUP
The package has been reused and this member has previously been backed up for this package.

blank
Indicates no backup has been made of this member and it cannot be restored.

Type
The type of the member.

Member Date/Time
The date and time this member was last changed before being packaged.

Restored Date/Time
The date and time this member was restored.

Enter the R primary command in the Command field next to invoke member level restore. If no members have been selected or member level restore is unavailable then this command is ignored.

Unit of Work Utility

The Unit of Work utility allows you to use an ARCHDEF member as a member list, from which you can use the standard SCLM utilities such as edit, build, view build map, and promote. Unlike the SCLM Library utility, which constrains you to working with one Type at a time, the Unit of Work utility provides access to all of the members associated with an architecture definition, regardless of Type.

In this way, the SCLM administrator can neatly organize all members of one language into separate libraries and a programmer can manage all the components for one "unit of work" (UOW) from a single point of control, without having to go back and forth to multiple member lists.

A Unit of Work member must be in standard ARCHDEF format and must contain an INCLD, INCL, COPY, SINC, or PROM statement for each editable member-type that is to be worked on for the programmer's current task. In principle, any architecture definition is eligible to be a Unit of Work, however the usefulness of the current architecture definitions in this regard will be determined by their contents.

When an architecture definition is selected in the SCLM Unit of Work processing - Entry Panel (or a new one is created), SCLM reads the member and creates a member list of the contents. Any embedded architecture definitions can also be selected and this provides a drill-down facility until the final non-ARCHDEF component is selected. This member is presented to the user in edit mode. All normal SCLM member list functions are available from this list, as well as some special "User" options that can facilitate local implementations.

The architecture definition that creates the member list is referred to as the Unit Of Work. The list of members generated from the Unit of Work is called a Work Element List. A member from this list is called a Work Element.

When an ARCHDEF member is selected, if the ARCHDEF member has been built SCLM uses the Build map associated with the architecture definition to build a list of members. If the ARCHDEF has been saved but not built, SCLM parses the ARCHDEF member to generate the list of members.

```
  Menu  SCLM  Utilities  Options  Help
_______________________________________________________________________________
                SCLM Unit Of Work processing - Entry Panel


SCLM Library:
  Project  . : SCLMTEST
  Group  . . . DEV1
  Type . . . . ARCHDEF     (Must contain Architecture Definitions only)
  Member . . . ________    (Blank or pattern for member selection list)


Enter "/" to select option
_  Hierarchy view
/  Confirm delete
_  Show Member Description
/  View processing options for Edit
/  View processing options for Build
/  View processing options for Promote
_  List include members


Option ===> ___________________________________________________________________
 F1=Help      F3=Exit     F10=Actions  F12=Cancel
```

Figure 122. SCLM Unit of Work processing - Entry Panel (FLMUW#P)

When you enter your choices from this panel, the UOW Member List panel is displayed. From this panel, you can choose to select, edit, build, promote, and otherwise manipulate the members. See “UOW Member List panel” on page 228 for details.

The fields on the SCLM Unit of Work Processing - Entry Panel are:

Project
The project that you specified on the SCLM Main Menu. An Alternate field also appears if you specified an alternate project definition. You cannot change the Project or Alternate fields on this panel.

Group
The group that you specified in the Group field on the SCLM Main Menu. The group field can be modified to specify other groups defined to the project.

Type
The identifier for the type of information in the ISPF library. While this field does not prevent you from using other Types, only a member constructed as an ARCHDEF will generate the appropriate member list.

Member
The name of an SCLM library member. You can display a member list by leaving the Command field blank and the Member field blank or by leaving the Command field blank and entering a pattern as the member name. See “Specifying selection criteria” on page 181 for details. Valid pattern characters are the asterisk (*) and the logical NOT symbol (¬).

Hierarchy view
Selects as input the library entered on the panel, as well as all the libraries in its hierarchy view. The hierarchy is searched from the bottom up for the first occurrence of the specified member. If you do not select "Hierarchy view", only the library entered on the panel is used as input. This option is valid with all UOW Member List commands except Delete, which defaults to NO.

Confirm delete
Allows you to specify whether you want a confirmation panel to appear when attempting to delete objects (text, accounting information, or build map information) in the UOW Member List panel. If you select this field, the Confirm Delete panel appears every time you request a delete. As well as confirming the delete request, this panel enables you to choose which information you want to delete

for the member. If you do not select this field, the Confirm Delete panel does not appear for deletions and all data is deleted without any additional user interaction.

Show member description
Allows you to display the member list panel FLMUSM#P, which contains an extra line displaying the description associated with a member. The Description is entered via the SPROF command.

View processing options for Edit
Allows you to indicate whether you want to verify or update edit processing options or allow them to default to the values that last appeared on the Edit Data Entry panel. When you select this option and then attempt to edit a member in the UOW Member List, the SCLM Edit Data Entry panel is displayed so that you can verify or update edit processing options. If you do not select it, Edit options default to those values that last appeared on the Edit Data Entry panel and the panel does not appear.

View processing options for Build
Displays the SCLM Build Data Entry panel so that you can verify or update Build processing options before Build is run.

View processing options for Promote
Displays the SCLM Promote Data Entry panel so that you can verify or update Promote processing options before Promote is run.

List include members
Allows you to indicate whether include members that are associated with members listed in the architecture definition are to be added to the member list that is generated by SCLM.

Unit of Work options

The SCLM Unit of Work processing - Entry Panel contains a unique set of Action Bar choices, under the "Options" menu.

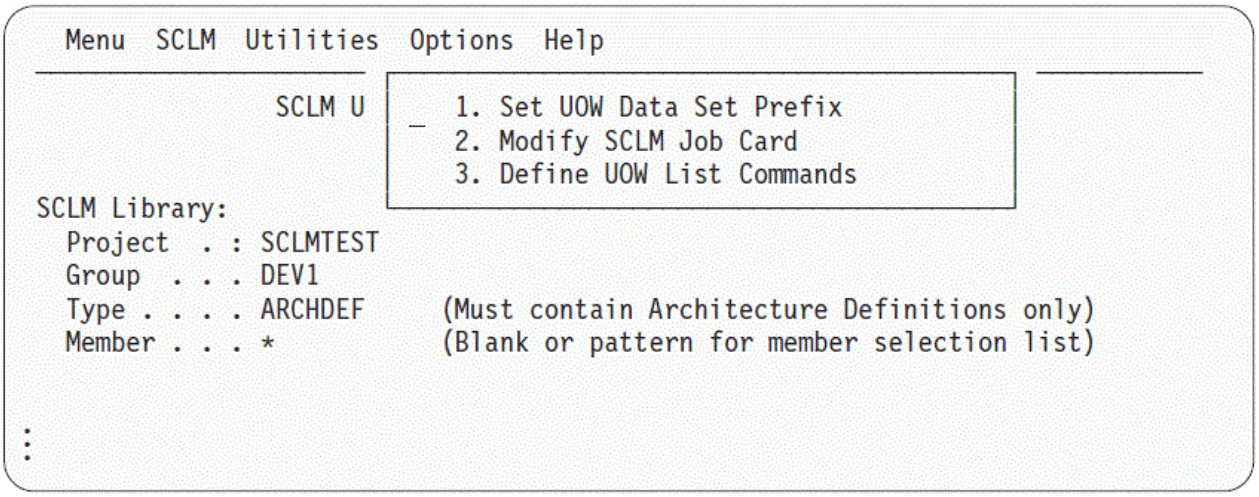


Figure 123. SCLM Unit of Work Options Action Bar choices

Set UOW Data Set Prefix
Displays a panel in which you can set the default prefix for all Unit of Work output data sets.

Modify SCLM Job Card
Displays the standard SCLM Verify Batch Job Information panel. See “Batch Processing” on page 252 for details.

Define UOW List Commands
Displays a panel in which you can create a customized list of commands that display on the UOW Member List panel.

SCLM Unit of Work Data Set Specification panel

In the SCLM Unit of Work Data Set Specification panel, you can specify the default prefix for all Unit of Work output data sets.

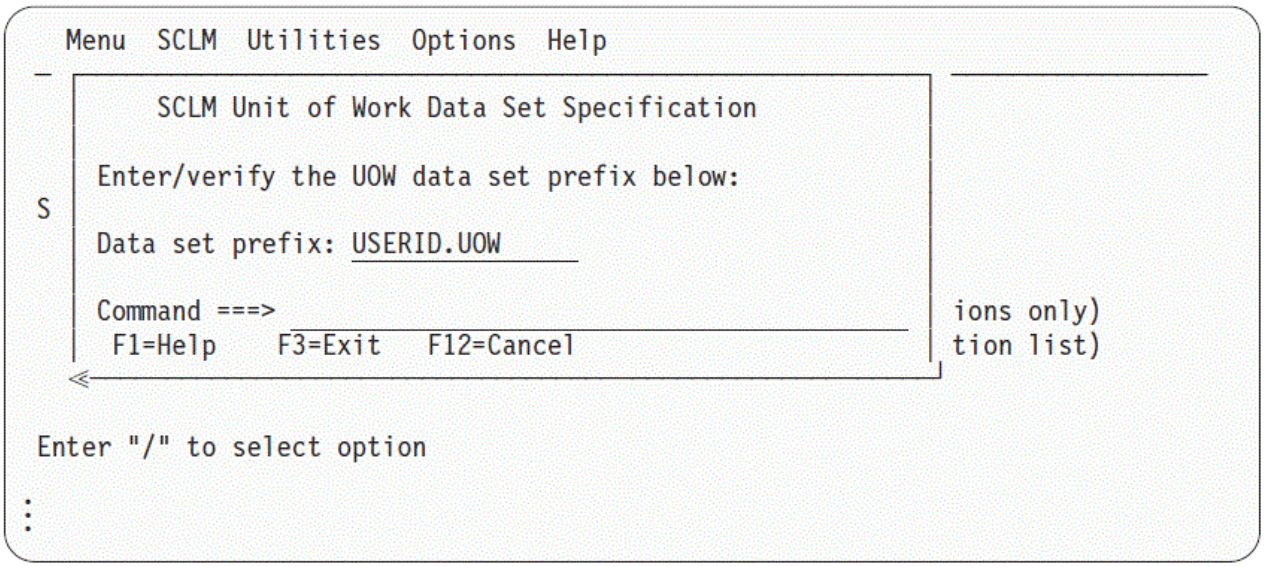


Figure 124. Set Work Data Set Prefix

The "Data set prefix" field defaults to your user ID. You can specify any prefix, provided that the first delimiter is RACF-authorised.

Define Unit of Work list commands

In the SCLM Unit of Work List Commands panel, you can specify up to eight user-defined line commands that will appear on the UOW Member List panel for the current project.

There are 3 levels of Unit of Work list commands. These are:

User-defined
When you create your own Unit of Work list commands, they are saved as a member in your ISPF user profile data set. The member name is derived from the current project qualifier. If the project name is 7 characters or less, a "Y" is added to the beginning of the member name. If the project name is 8 characters, the first letter of the project is changed to "Y". If the project name is 8 characters and it already starts with a "Y", the second letter is changed to a "Y". For example:

```
Project qualifier = HLASM, Member = YHLASM
Project qualifier = HLASMKIT, Member = YLASMKIT
Project qualifier = YEAR2000, Member = YYAR2000
```

In this way, each user can create a set of Unit of Work list commands that are specific to each project.

Project-defined
Your project administrator can create a project-wide set of Unit of Work List Commands by using these options to create a user list, then copying the project member from their ISPPROF DDNAME, to a library that is allocated to ISPTLIB ahead of the ISPF libraries.

ISPF-supplied
This currently contains a single default entry, Versions. The member is stored in the ISPF-supplied library allocated to ISPTLIB.

The order of precedence is:

```
User -> Project -> ISPF
```

When the SCLM Unit of Work List Commands panel is displayed, SCLM looks for a member in the user's profile data set that matches the naming convention for the current project. If it does not find this member, it will look in the ISPTLIB project library, and then in the ISPF library.

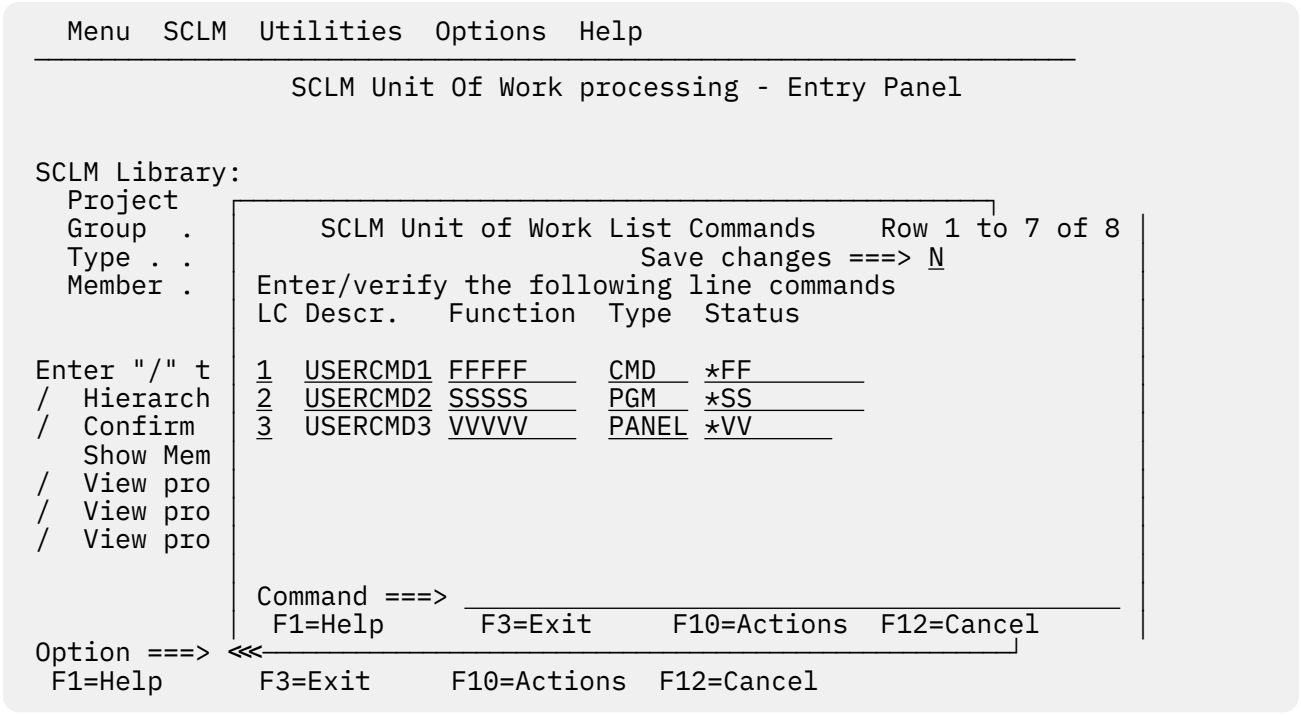


Figure 125. SCLM Unit of Work List Commands panel

The fields on the SCLM Unit of Work List Commands panel are:

LC
The character that is to be entered to select this command.

Descr
The keyword that will be displayed to represent this command.

Function
The name of the function that will be invoked.

Type
The type of function. This can be either CMD, PANEL, or PGM.

Status
The comment that will be placed in the status field when the command has successfully executed.

When you first open this panel, SCLM displays any project-defined list of commands. If there is no project-defined list, the ISPF default list is displayed. If you make and save any changes to either of these lists, a copy of the displayed list is saved into your user profile data set.

You can enter a maximum of 8 lines (commands). To define a valid command, all the fields must be filled in. If you overtype a line with blanks, the line is deleted. Only completed lines are saved.

If you delete all the lines in your user-defined Unit of Work list commands data set, the member is deleted and the project-defined list becomes the default list. This is displayed when you reopen the panel. If there is no project-defined list, the ISPF default list is used.

UOW Member List panel

The UOW Member List panel displays the list of ARCHDEFs that match the member name pattern entered on the previous panel. You can apply the standard SCLM line commands or your own user-defined UOW Member List commands to each member in this list.

```
  Menu  SCLM  Functions  Utilities  Help
_____________________________________________________________________
UOW Member List: SLMTEST.DEV1.ARCHDEF - HIERARCHY VIEW -       Member 1 of 12

S=Sel/Edit  A=Account   M=Map       B=Browse    D=Delete    E=Edit      V=View
 C=Build     P=Promote   U=Update    T=Transfer  N=NOPROM
 Z=Versions

 Member   Status      Account  Language   Text       Chg Date   Chg Time
_    FLM01AP1            DEV1     ARCHDEF    DEV1       2006/12/11 12:54:25
_    FLM01ARH            DEV1     ARCHDEF    DEV1       2006/12/11 12:54:39
_    FLM01CMD            DEV1     ARCHDEF    DEV1       2006/12/11 12:54:52
_    FLM01CM9            DEV1     ARCHDEF    DEV1       2006/12/11 12:54:59
_    FLM01LD1            DEV1     ARCHDEF    DEV1       2006/12/11 12:55:19
_    FLM01LD2            DEV1     ARCHDEF    DEV1       2006/12/11 12:55:29
_    FLM01LD3            DEV1     ARCHDEF    DEV1       2006/12/11 12:55:35
_    FLM01LD4            DEV1     ARCHDEF    DEV1       2006/12/11 12:55:41
_    FLM01LD7            DEV1     ARCHDEF    DEV1       2006/12/11 12:55:49
_    FLM01LD9            DEV1     ARCHDEF    DEV1       2006/12/11 12:55:56
_    FLM01SB1            DEV1     ARCHDEF    DEV1       2006/12/11 12:55:03
_    FLM01SB2            DEV1     ARCHDEF    DEV1       2006/12/11 12:55:09
******************************Bottom of data********************************

Command ===>_____________________________________________ Scroll ===> PAGE
  F1=Help     F3=Exit     F10=Actions  F12=Cancel
F10=Actions  F12=Cancel
```

Figure 126. UOW Member List panel

The default commands available from this panel are:

S=Select
Selects the ARCHDEF member and displays the contents as another list of members (the Work Element List).

A=Acct
Displays the Accounting Record for the specified member.

M=Map
Displays the Build Map Record for the specified member.

B=Browse
Displays the specified member in an ISPF Browse session.

D=Del
Deletes the specified member. If the "Confirm delete" option was selected on the previous panel, the Confirm Delete panel is displayed, otherwise, the member is deleted without confirmation.

E=Edit
Displays the specified member in an ISPF Edit session. If the "View processing options for Edit" option was selected on the previous panel, the SCLM Edit - Entry Panel is displayed, otherwise, the member is opened for editing immediately, using the Edit options most recently specified in the Edit Entry panel.

V=View
Displays the specified member in an ISPF View session.

C=Build
Builds the specified member. If the "View processing options for Build" option was selected on the previous panel, the SCLM Build - Entry Panel is displayed, otherwise, the member is built immediately, using the Build options most recently specified in the Build Entry panel.

P=Promote
Promotes the specified member. If the "View processing options for Promote" option was selected on the previous panel, the SCLM Promote - Entry Panel is displayed, otherwise, the member is promoted immediately, using the Promote options most recently specified in the Promote Entry panel.

U=Upd
Displays the SCLM Authorization Code Update panel for the selected member.

T=Transfer
SCLM modifies the Change User ID field in the accounting record to transfer ownership of the member to another user.

N=NOPROM
Allows you to modify how the SCLM member is promoted. You can use this option to leave the member behind on promotion.

W=WhereUsed
Shows what members include this member.

One additional command is provided as a sample of a user-defined Unit of Work list command:

Z=Versions
Lists versions of the selected member.

Work Element List panel

The Work Element List panel displays the contents of the selected Unit of Work (ARCHDEF) as a list of members. You can apply the standard SCLM line commands or your own user-defined Work Element List commands to each member in this list.

The first line on this panel contains the current UOW ARCHDEF member that was used to generate the member list. This allows the current Unit of Work ARCHDEF member to be maintained or built without exiting the member list.

If you use the current UOW ARCHDEF member to add or delete members, the member list can be refreshed by entering REFRESH on the command line.

```
  Menu  SCLM  Functions  Utilities  Options  Help
 ──────────────────────────────────────────────────────────────

Work Element List for UOW  FLM01AP1 in SLMTEST6                 Member 1 of 2
Command ===> _____________________________________________ Scroll ===> CSR

S=Sel/Edit  A=Account   M=Map       B=Browse    D=Delete    E=Edit      V=View
C=Build     P=Promote   U=Update    T=Transfer  N=NOPROM
Z=Versions
    Member   Type     Status      Account  Language Chg Date         User
    FLM01AP1 ARCHDEF              (Current UOW ARCHDEF)
    FLM01SB1 ARCHDEF              RELEASE  ARCHDEF  2001/10/30 16:59 JPHILP
    FLM01SB2 ARCHDEF              RELEASE  ARCHDEF  2001/10/30 16:59 JPHILP
****************************** Bottom of data ********************************


 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 127. Work Element List panel

The commands available are the same as those on the UOW Member List Panel. See Figure 126 on page 229

SCLM Explorer

The SCLM Explorer utility provides an interactive facility for viewing the relationships between components of a project. You can select as a starting point any architecture definition or part member and then navigate up or down the hierarchy of related ARCHDEFs or parts.

With SCLM Explorer, you can:

- Display a list of all components, or selected components.

- Select a component and identify its immediate relations (parent or child).
- Select a component and identify its related executable components.

In addition, the utility provides some impact analysis capability by identifying the "buildable" or "linkable" components that would be affected by planned changes to a lower-level component.

Component relationships are generally hierarchical. For example:

ARCHDEFs can include source parts
Source parts include other source parts
ARCHDEFs can include other ARCHDEFs

Figure 128 on page 231 shows the panel that is displayed when you select Option 12, SCLM Explorer, from the SCLM Utilities panel.

```
  Menu  Utilities  Help
_______________________________________________________________________
                              SCLM Explorer
Option ===> ___________________________________________________________________

Browse the relationships between entities within the project hierarchy.

0  Settings             Set SCLM Explorer table library.
1  Build                Define SCLM project dependency information tables.
2  Parts                Start with a list of parts.
3  Architecture         Start with a list of architecture definitions.









 F1=Help      F2=Split     F3=Exit      F6=Nretriev  F7=Backward  F8=Forward
 F9=Swap     F10=Actions  F12=Cancel
```

Figure 128. SCLM Explorer panel (FLMUDEP0)

Select option 2 to list the components in the project. Select option 3 to list the architecture definitions in the project.

The following commands are available:

U (up)
Show the parents of the selected component

D (down)
Show the child components

L (LMOD)
Show related executable components (load modules)

Parent or child relations can be followed until no further relations remain. Position the cursor on one of the displayed components and press Enter to show its parent or child components. Parent relationships will generally terminate at a high level architecture definition (HLMAP), while child relationships usually terminate with a low-level copybook source or macro part.

As the relationship hierarchy is navigated, a 'path' description is maintained, identifying the chain of selected parts.

The relationship information is extracted from the accounting files in the project database and stored in a set of ISPF tables. The tables are populated by the FLMUEXTR utility (see “FLMUEXTR—the SCLM Explorer batch utility” on page 232). The data displayed in SCLM Explorer therefore reflects the status of the project at the time the batch utility was last run.

The extraction process is controlled by the following options on the SCLM Explorer panel:

Option 0 is used to specify the name of the ISPF table library. Your SCLM Administrator can provide you with the library name.
Option 1 is used to build the JCL for a batch job to extract data and populate the ISPF tables.

FLMUEXTR—the SCLM Explorer batch utility

This program reads the project accounting files and populates a set of ISPF tables used by SCLM Explorer. To keep the data current, this batch job should be run regularly, for example daily during overnight processing. This process would typically be managed by the SCLM administrator.

Use the Build option from the SCLM Explorer panel to create the JCL for running the batch utility. The fields on the SCLM Explorer Batch JCL panel are:

Project Id
The project name.

Table Library
The name of the output ISPF table library.

This library must already exist. Note that the JCL build process creates some tables in this library.

HLQ for accounting file copy
Projects can have multiple accounting files. To simplify the data extraction process a single input file is used. All project accounting files are therefore copied into a single file. This parameter specifies the high-level qualifier to use for this temporary file.

Ensure that the user ID to run the batch job has authority to delete, allocate, and update files with this prefix.

HLQ of ISPF libraries
The extraction program runs under ISPF in batch.

Jobclass
Run job in this class.

Msgclass
Output messages to this class.

After all the fields have been specified, press Enter to build the JCL. You can modify the JCL before submitting it if required.

SCLM Search

The SCLM Search utility provides a single SCLM search interface where you can search complete SCLM hierarchies or group/type/member patterns for a string or a number of strings using SuperC without having to use the ISPF Search-For panels. All of the standard functions of Search-For are available through the use of the SCLM interface. Additionally, the SCLM Search utility performs decoding of members if they are encoded, and security checks if member security is activated.

To display the SCLM Search utility panel shown in Figure 129 on page 233:

1. Select option 3 (Utilities) from the SCLM Main Menu.
2. Select option 13 (SCLM Search) from the SCLM Utilities Menu.

```
  Menu  SCLM  Utilities  Jobcard  Help
_____________________________________________________________________
                          SCLM Search - Entry Panel
Command ===> _____________________________________________________________

Search string . . _____________________________________________________________

Search input:
Project  . : PDFTDEV
Group  . . . ________  (Pattern may be used)
Type . . . . ________  (Pattern may be used)
Member . . . *_______  (Pattern may be used)

Enter "/" to select option
_  Search hierarchy              _  Mixed mode
_  Additional search strings     _  Bypass selection list
_  Case sensitive                _  Editable types only

Statements Dsn . . . _____________________________________________________________

Output control:
                Ex Sub                  Process . . 1  1. Execute
  Messages . . 3  3    1. Terminal                     2. Submit
  Report . . . 3  3    2. Printer
                       3. Data set     Printer . . _
                       4. None         Volume  . . ______
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 129. SCLM Search Entry panel (FLMUC#P)

The fields for the SCLM Search Entry panel are:

Search string
A string to be searched for. If you need to distinguish between uppercase and lowercase characters, then select the **Case sensitive** option.

Four keywords (PREFIX, SUFFIX, WORD, and C) can help you narrow the scope of a search. See "Search-For strings and keywords" on page 235 for information about these keywords and the rules that govern search string entry.

Project
The project that you specified on the SCLM Main Menu. If you specified an alternate project, SCLM also displays an Alternate field.

Group
You can specify the group as a single group or pattern. For a pattern, specify * for all groups or, for example, DEV* to search all groups beginning with "DEV". A group with a pattern is considered invalid if the Search hierarchy option is specified. If you specify a specific group, then you can also set the **Search hierarchy** option to allocate the hierarchy for the search.

Type
You can specify the type as a single type or pattern. Specify * for all types or, for example, C0* to search all types beginning with "CO".

Member
You can specify the member as a single member or pattern. Specify * for all members or, for example, *ABC* to search all members that contain "ABC" in their name.

Search hierarchy
If you have specified a single group (that is, you have not included a "*" in the **Group** field), then you can optionally search the hierarchy. The hierarchy is searched from the entered group upwards.

Mixed mode
SuperC option. Select this option to instruct the Search-For utility to scan and parse the input data set lines for DBCS text strings.

Note: The WORD, PREFIX, and SUFFIX Search-For qualifiers have no effect on DBCS strings.

Additional search strings
Select this option to instruct the SCLM Search utility to search for more than one string. The SCLM Search utility displays the SCLM Search strings panel shown in Figure 130 on page 235, on which you can specify additional search strings. This panel precedes a member list request.

If you do not select this option, SuperC searches only for the string entered in the **Search string** field.

Bypass selection list
If you have entered a member pattern in the **Member** field, selecting this option causes SuperC to process all members matching that pattern without displaying a member selection list.

If you do not select this option, the member list is displayed.

Case sensitive
Select this option to search for occurrences where the result matches exactly the entered string or strings.

Editable types only
If you have specified a type pattern by using an "*" in the **Type** field, only search those types that contain editable parts. For example, do not search OBJ if a pattern of O* is entered in type.

Statements Dsn
The name of the data set that contains your SuperC process statements. If specified, the data set must already exist. SuperC reads these process statements before conducting the search. All statements data sets must be fixed-block with 80-byte records (FB 80). For information about SuperC statements, see Appendix A ("SuperC reference") in the *ISPF User's Guide Volume II*.

Output control
Specify the destination for messages and reports when they are executed (Ex) or submitted (Sub) by entering the corresponding destination number:

1
Terminal

2
Printer

3
Dataset

4
None

The data sets that are created are not deleted. If you specify a volume that already contains a report or message data set, it may result in JCL errors when the job is submitted.

Process
You can call the processing part of the SCLM Search utility from the interactive or batch environment by selecting **Execute** or **Submit**, respectively. If you request batch processing by selecting **Submit**, you must specify the job statement information that is used in the JCL generated for batch processing.

For information about using a unique jobname on the jobcard in batch processing, see “Batch Processing” on page 252.

Printer
Specify the printer output class.

Volume
Specify the volume on which you want SCLM to save data sets.

Specifying additional search strings

The panel shown in Figure 130 on page 235 is displayed if you select the **Additional search strings** option on the SCLM Search utility panel panel (see Figure 129 on page 233). You can specify:

- Additional strings to be searched for
- Optional scan-type and continuation keywords

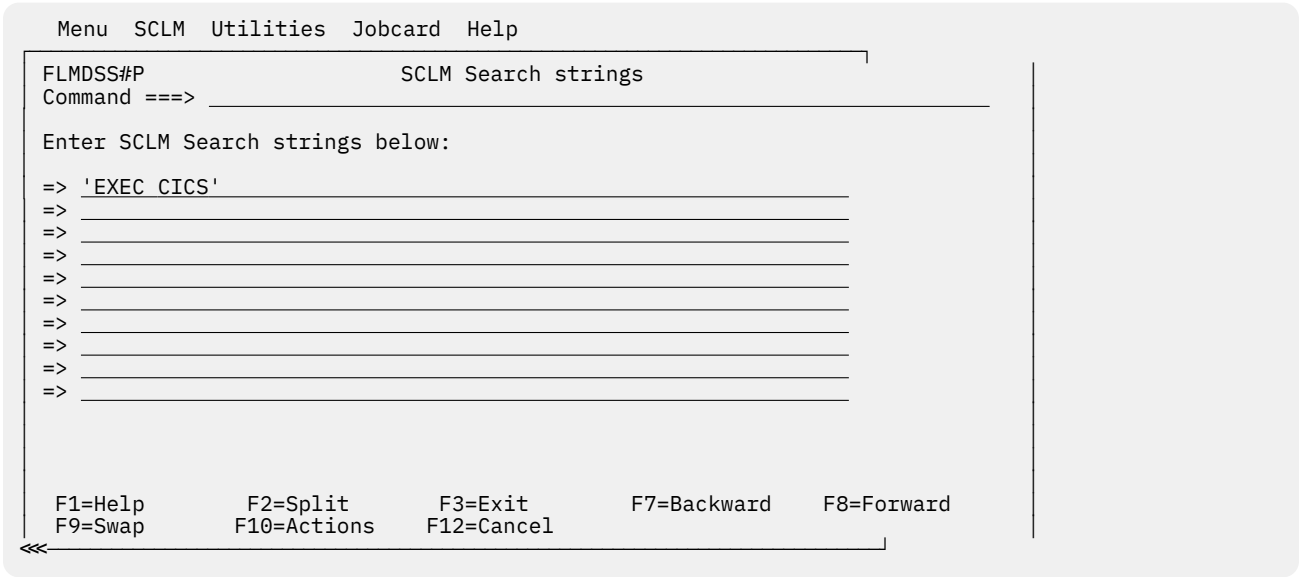


Figure 130. SCLM Search strings panel (FLMDSS#P)

From the SCLM Search strings panel, pressing Enter results in SCLM performing one of these actions:

- Displaying the member lists, if requested
- Running the search if no member lists are needed

Entering the END command returns you to the SCLM Search utility panel.

Search-For strings and keywords

Enter the strings that you want SuperC to find. When you press Enter, SuperC looks for the strings without regard to whether they appear in uppercase or lowercase in the original data sets, unless you have selected the **Case sensitive** option, in which case, only strings matching the exact string are found.

If you are searching the SCLM hierarchy, SCLM informs you (both with the "hier" keyword on the header page and in the CONCAT# field in the listing) of the group in which it first found the string. To search all the groups for each member, use the pattern character (*) in the group field. If any of the strings are found (string-1 OR string-2 OR string-3) on the line, SuperC considers the condition met. You can further restrict SuperC searches by using one of the SuperC Search-For keywords discussed in “Using keywords” on page 235.

Entering search strings

Enclose the string in single quotation marks if it contains embedded blanks or apostrophes. Two consecutive apostrophes must be entered to specify a single apostrophe within a search string. If you need to specify a DBCS string that contains a hexadecimal '7D' (X'7D', the hexadecimal representation of a single quotation mark) as half of a DBCS pair, you must use the **Mixed mode** option.

Using keywords

The keywords shown here can help you narrow the range of the search. If you do not use a keyword, SuperC finds the string wherever it exists, even if that happens to be in the middle of a word.

PREFIX
Shows the string is preceded by a non-alphanumeric character, such as a blank space. It cannot be used on the same line with SUFFIX or WORD. For example, you can specify this:

```
==> ELSE PREFIX
==> ELSE SUFFIX
```

but not this:

```
==> ELSE PREFIX SUFFIX
```

SUFFIX
Shows the string is followed by a non-alphanumeric character. It cannot be used on the same line with PREFIX or WORD. See the examples under PREFIX.

WORD
Shows the string is both preceded and followed by a non-alphanumeric character. It cannot be used on the same line as PREFIX or SUFFIX. See the examples under PREFIX.

C
Continuation. Shows continuation of the previous line. Continuation lines generate additional strings, all of which must be found in the same line of an input data set. Also, the C keyword can be entered on the same line as one of the other keywords. The example shown here instructs SuperC to find ELSE and to also find IF, but only when IF is on the same line as ELSE.

```
==> ELSE WORD
==> IF WORD C
```

SCLM Search member lists

A panel similar to the one shown in Figure 131 on page 236 is displayed only if the **Bypass selection list** option is *not* checked.

```
  Menu  SCLM  Functions  Utilities  Test  Help
──────────────────────────────────────────────────────────────────────
Member List : SCLM07.DEV1.COBOL                          Member 1 of 15
Command ===>  ____________________________________________  Scroll ===> CSR

 S=Select

S   Member   Status      Text     Chg Date   Chg Time
_   BENKE01              DEV1     2008/03/04 04:19:44
_   BZZSMEP1             DEV1     2007/09/21 11:18:51
_   CBLDBGEX             DEV1     2008/02/26 02:03:19
_   ENTCOB1              DEV1     2007/03/26 17:59:34
_   ENTCOB2              DEV1     2007/03/26 18:03:53
_   PETER1               DEV1     2007/01/11 21:49:40
_   PETER2               DEV1     2007/01/03 01:20:39
_   PRINTAPP             DEV1     2008/01/08 21:56:21
_   RDBKC02              DEV1     2008/01/05 18:46:32
_   RDBKC03              DEV1     2008/02/28 03:43:17
_   SCLMTST1             DEV1     2007/01/11 05:08:30
_   SCLMTST2             DEV1     2007/01/31 22:48:13
_   SCTRAN01             DEV1     2007/10/03 16:31:20
_   STARTAPP             DEV1     2008/01/08 21:56:21
_   VCMSCLM              DEV1     2006/12/23 06:01:59
****************************** Bottom of data ********************************
```

Figure 131. SCLM Search member list panel

To start the search, select any members and enter the END command. To cancel your selections, enter either:

- The RESET command to remove all unprocessed selections without ending the member list display.
- The CANCEL command to end the member list display without processing selections still on the screen.

Note: Both the jump function (=) and the RETURN command cause an implied cancellation of selections before they are carried out. For more information about member lists, see the "Using member selection lists" section of the "ISPF libraries and data sets" chapter of the *z/OS ISPF User's Guide Vol I*.

SCLM Search example

Figure 132 on page 237 shows an example of an SCLM Search report. By using Edit highlighting and entering HILITE SUPERC, the report is colored as for a normal SuperC report.

```
1  ISRSUPC   -   SCLM
   - : SCLM07 : DEV1  (hier) Type: COBOL Member: RDBK*
 *
  LINE-#  SOURCE SECTION                   SRCH DSN: SCLM07.PROD.COBOL


  RDBKC01  CONCAT#(3)       ----------- STRING(S) FOUND -------------------

      65                     EXEC CICS
      78             EXEC CICS
     110             EXEC CICS
     156             EXEC CICS
     164             EXEC CICS

  RDBKC02  CONCAT#(1)       ----------- STRING(S) FOUND -------------------

      65                     EXEC CICS
      78             EXEC CICS
     110             EXEC CICS
     156             EXEC CICS
     164             EXEC CICS

  RDBKC03  CONCAT#(1)       ----------- STRING(S) FOUND -------------------

      65                     EXEC CICS
      78             EXEC CICS
     110             EXEC CICS
     156             EXEC CICS
     164             EXEC CICS

1  ISRSUPC   -   SCLM
   - : SCLM07 : DEV1  (hier) Type: COBOL Member: RDBK*

*
     SEARCH-FOR SUMMARY SECTION            SRCH DSN: SCLM07.PROD.COBOL

LINES-FOUND  LINES-PROC  MEMBERS-W/LNS  MEMBERS-WO/LNS  COMPARE-COLS  LONGEST-LINE
       15          507            3              0          1:80           80

PROCESS OPTIONS USED: ANYC NOPRTCC

THE FOLLOWING PROCESS STATEMENTS (USING COLUMNS 1:72) WERE PROCESSED:
   SRCHFOR  'EXEC CICS'
   SELECT RDBKC01,RDBKC02,RDBKC03

THE FOLLOWING "SRH" FILE CONCATENATED DATA SETS WERE SPECIFIED:
   (1) SCLM07.DEV1.COBOL
   (2) SCLM07.TEST.COBOL
   (3) SCLM07.PROD.COBOL
```

Figure 132. Example of an SCLM search report

Build (option 4)

The build processor automatically compiles, links, or deletes output to make build outputs match build inputs. The build function:

- Ensures total project integrity by verifying that all components defined to the architecture being built are present and complete
- Performs necessary translations such as compiles and links
- Conditionally saves translator output in the database
- Generates a build report

Build compiles, links, and integrates software components according to the architecture. For any group in the hierarchy, the build function uses the software components within the hierarchy of that group to update the out-of-date members. Use build to compile and link individual components as well as to integrate the smaller components into larger components.

For each component that it processes, the build function takes one of the following actions:

- Does nothing if the component has not changed since the previous build
- Deletes out-of-date outputs if that will leave the component in an up-to-date state
- Compiles or links changed components.

At the completion of the build, SCLM, when requested, produces a report identifying the members that were generated or deleted by the build function.

You also can specify that a Build Report be generated without actually invoking any translators. The Build Report identifies those components in the hierarchy that would change if translators were to be invoked.

With the PTFs for APAR OA21104 applied, a build can also be run in information mode. When this is done the options passed to the translators, and the data sets used in the concatenations for resolving the include dependencies, are written to the build report output in XML format. The options that would be passed to a compiler will be shown, after FLMTOPTS and Compilation Control architecture member overrides have been taken into consideration. In addition, all DDs in the SYSLIB, or any other IOTYPE=I allocation, will be shown as they would be passed to the compiler with all FLMCPYLB, FLMINCLS and FLMSYSLB overrides taken into consideration.

Before build begins processing the member, it tries to open the VSAM accounting and cross-reference data sets for the group where the build is taking place. If you do not have UPDATE authority to the data sets or if there is an error opening one of the data sets, the build will fail. See "BUILD—Build a Member" on page 388 for more information about the processing done by the build processor.

The panel shown in Figure 133 on page 238 appears when you select Option 4, Build, from the SCLM Main Menu.

```
   Menu  SCLM  Utilities  Jobcard  Test  Workstation  Build  Help
 ──────────────────────────────────────────────────────────────────
                            SCLM Build - Entry Panel
Command ===>

Build input:
  Project  . : PDF42SVT
  Group  . . . DEV1
  Type . . . . ARCHDEF            Enter "/" to select option
  Member . . . SAMPLE                    /_ Error Listings only
                                          Workstation Build

  Mode  . . 1  1. Conditional          Scope . . . 2  1. Limited
               2. Unconditional                       2. Normal
               3. Forced                              3. Subunit
               4. Report                              4. Extended
               5. Information
Output control:
               Ex Sub                  Process . . 1  1. Execute
  Messages . . 3  3    1. Terminal                    2. Submit
  Report . . . 3  3    2. Printer
  Listings . . 3  3    3. Data set     Printer . . H
                       4. None         Volume  . . ________
```

Figure 133. SCLM Build (FLMB#P)

Note: The NRETRIEV command key is enabled to work with this option. See "Name retrieval with the NRETRIEV command" on page 143 for more information.

The fields for the SCLM Build - Entry panel are:

Project
The project that you specified on the SCLM Main Menu. An Alternate field also appears if you specified an alternate project.

Group
The group in which the build is to occur.

Type
The type of the member to build.

Member
The name of the member to build.

Scope
You must specify a scope equal to or greater than the scope specified with the SCOPE keyword in the FLMLANGL macro.

Limited
To process those components that the architecture members directly reference. If you use a source member, the build function processes only that member.

Normal
To process the components and members referenced by the specified architecture member. In addition, this scope processes upward dependencies for all Ada-type source members referenced directly by the architecture member and all source members referenced as upward dependencies.

Subunit
To process the components and members processed in normal scope as well as downward dependencies for all Ada-type source members referenced directly by the architecture members.

Extended
To process the components and members processed in normal scope as well as downward dependencies for all source members within the normal scope and the source to all outputs referenced. In addition, extended scope processes any outputs referenced via LINK architecture definition statements or parsed includes. Extended scope also includes anything that Promote verifies that is related to the member built. For example if the architecture definition statement LINK is used to reference a load module, the architecture definition that created the referenced load module is included in the extended scope.

Because SCLM uses information from the most recent build map to determine what should be included in extended scope, extended scope may include members that are no longer relevant to the architecture. If you receive error messages about members that are no longer relevant to the architecture definition, try building in normal scope before using extended scope.

Mode

Conditional
To check for unacceptable translator return codes (for example, compiler or linker return codes). Processing stops immediately if build detects any translation errors.

SCLM saves build maps and translator output only for translations that complete successfully. However, the translator listings (if desired) for all components processed, and the build report, are saved and reflect the final results of the build.

Unconditional
To continue processing of all members despite translation errors of other members.

Use this mode when you need to update complete applications or large subapplications. You can also use this mode initially to detect translation errors in several components.

As with the conditional mode, BUILD will stop when verification errors occur and not continue on to execute the BUILD translators. After a successful verification of the members, SCLM will pass control to the BUILD translators, regardless of the return code value from each translator. This will provide information as to the extent of any errors that may have been introduced by changing the members. A conditional BUILD would stop after the first translator return code that exceeds the GOODRC value for the related FLMTRNSL macro.

Build does not attempt a translation unless all of its dependencies that were in scope were completed successfully. For example, a link-edit is not attempted if the compilation of a source member failed.

Forced
To force all requested components to be translated again regardless of the previous status of the modules.

Report
To generate a complete build report without performing an actual build. The report reflects the potential results of an unconditional build.

Information
With the PTFs for APAR OA21104 applied, the information Build returns an XML document describing the input that would have been passed to the BUILD translators on a Forced Build. The information is returned in the build report and contains the options passed to the translators, along with the data set allocations made by SCLM for IOTYPE=I allocations, which are include dependency allocations.

Output control
Specify the destination for messages, report, and listings when they are executed (Ex) or submitted (Sub), by entering the corresponding destination number: 1 for Terminal, 2 for Printer, 3 for Dataset, or 4 for None.

When executing a build in the foreground, the build listing is browsed if a translation error occurs; otherwise, the build report is browsed. The translator is responsible for providing the build listing.

Note: If no output is specified for Report, no build user exit information is produced. That is because SCLM provides the build user exit with information from the build report.

The data sets that are created are not deleted. Specifying a volume that already contains a report, message or listing data set could result in JCL errors when the job is submitted.

Error listings only
The build service allows you to generate a temporary listings file. If you do not select Error listings only, all translator listings are copied to the temporary listings file. If you select it, only those members receiving a translator error are copied to the temporary listings file. An empty file indicates that no errors occurred. The file is temporary in the sense that the contents are not under SCLM control and may be purged by the user.

Workstation Build
Specify whether the build will invoke any workstation translators. For a foreground build which invokes a workstation translator, SCLM will verify that an ISPF workstation connection exists before executing the build. For a batch build which invokes a workstation translator, SCLM will verify that the information required to initiate an ISPF workstation connection has been set by a previous build or the workstation build pull-down. If not, SCLM will prompt the user to enter this information before the build job is submitted. If the build does not invoke a workstation translator, do not specify this field.

Process
You can call the processing part of the build utility from the interactive or batch environment by selecting Execute or Submit, respectively. If you request batch processing by selecting Submit, you must specify the job statement information that is used in the JCL generated for batch processing.

For information about using a unique jobname on the jobcard in batch processing, see "Batch Processing" on page 252.

Printer
Specify the printer output class.

Volume
Specify the volume on which SCLM should save data sets.

Build Report example

The build report provides a synopsis of the build. It includes:

- The date and time of the build
- The mode used
- The name of the component that was requested to be built

- The last change date and time of the component
- The project definition used
- The software components that were successfully translated
- The build maps that required regeneration
- The out-of-date software components that caused the regeneration
- The software components and build maps that were deleted from the build group.

This report provides a synopsis of the Build. The title page identifies the date and time of the build, as well as the scope and mode used. It also lists the member you specified on the Build panel and the project definition specified on the SCLM Main Menu.

The report lists the components that were built and saved in the database; that is, those components that passed the compilation or linkage edit phase. It also shows the build maps that required (re) generation, along with a list of software components that build used to determine that (re)generation of the build map was necessary. After the section for items generated, the report contains a section for items deleted. It lists the build outputs that were deleted from the build group. Finally, it lists the build maps that were deleted.

Note: Intermediate information is in the report if it is valid and useful. The following example is an Ada build report, so the sections on Intermediate Code Generated and Intermediate Code Deleted have been included. These two sections are omitted from the report for builds that do not affect intermediate code.

If you enter REP0RT in the Mode field, the report indicates what would be rebuilt or deleted if you requested an unconditional build.

Figure 134 on page 242 shows an example of a build report.

```
**************************************************************************
**                                                                      **
**                                                                      **
**            SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)          **
**                                                                      **
**                         B U I L D    R E P O R T                    **
**                                                                      **
**                          2000/11/18   08:41:19                       **
**                                                                      **
**                          PROJECT:      SCLM69                        **
**                          GROUP:        USER                          **
**                          TYPE:         MVS2ADA                       **
**                          MEMBER:       GSPEC                         **
**                          ALTERNATE:    SCLM69                        **
**                          SCOPE:        NORMAL                        **
**                          MODE:         CONDITIONAL                   **
**                                                                      **
**                                                                      **
**************************************************************************


    ******* B U I L D   O U T P U T S   G E N E R A T E D *******  Page 1

MEMBER     TYPE       VERSION      KEYWORD
------     ----       -------      -------
FLM01MD3   OBJ              6      OBJ
FLM01MD5   OBJ              6
FLM01MD6   OBJ              6
FLM01MD3   LIST             6      LIST
FLM01MD5   LIST             6
FLM01MD6   LIST             6
FLM01LD3   LOAD             6      LOAD
FLM01LD3   LMAP             6      LMAP

           ******* B U I L D   M A P S   G E N E R A T E D *******  Page 2

                                 (REASON FOR REBUILD)
MEMBER     TYPE       VERSION      MEMBER     TYPE
------     ----       -------      -------    ----
FLM01LD3   ARCHDEF          3      FLM01MD3   SOURCE
FLM01MD3   SOURCE           6      FLM01MD3   SOURCE
                                   FLM01MD5   SOURCE
                                   FLM01MD6   SOURCE
FLM01MD5   SOURCE           5      FLM01MD5   SOURCE
FLM01MD6   SOURCE           4      FLM01MD6   SOURCE
```

Figure 134. Build Report (Part 1 of 2)

```
    ******* B U I L D   O U T P U T S    D E L E T E D   *******  Page 3

MEMBER     TYPE       VERSION      KEYWORD
------     ----       -------      -------
FLM2M01    OBJ              4      OBJ
FLM2M02    OBJ              4
FLM2M03    OBJ              4
FLM2M01    LIST             4      LIST
FLM2M02    LIST             4
FLM2M03    LIST             4
FLM2LD     LOAD             5      LOAD
FLM2LD     LMAP             5      LMAP

           ******* B U I L D   M A P S    D E L E T E D   *******  Page 4

                                 (REASON FOR DELETE)
MEMBER     TYPE       VERSION      MEMBER     TYPE
------     ----       -------      -------    ----
FLM02LD    ARCHDEF          6      FLM02LD    LOAD
                                   FLM02LD    LMAP
FLM02MD1   SOURCE           6      FLM02MD1   OBJ
                                   FLM02MD1   LIST
FLM02MD2   SOURCE           6      FLM02MD2   OBJ
                                   FLM02MD2   LIST
FLM02MD3   SOURCE           6      FLM02MD3   OBJ
```

Figure 135. Build Report (Part 2 of 2)

Build Information Example

With the PTFs for APAR OA21104 applied, a build information is returned in the build report data set. Just like the normal report mode build, the title page identifies the date and time of the build, as well as the scope and mode used. It also lists the member you specified on the Build panel and the project definition specified on the SCLM Main Menu.

The report lists, in XML format, all members in the scope of the build that would have been built if the build had been performed in "Forced" mode. For each member, the language is listed and each translator within the language definition is shown. For each translator, the options passed to the translator, along with any include dependency DD allocations, are shown.

Figure 136 on page 243 shows an example of a build information report.

```
********************************************************************************
********************************************************************************
**                                                                            **
**                                                                            **
**             SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)               **
**                                                                            **
**                          B U I L D    R E P O R T                          **
**                                                                            **
**                          2007/06/18   16:33:02                             **
**                                                                            **
**                           PROJECT:      ADMIN10                            **
**                           GROUP:        DEV1                               **
**                           TYPE:         COBOL                              **
**                           MEMBER:       RDBKC01                            **
**                           ALTERNATE:    ADMIN10                            **
**                           SCOPE:        NORMAL                             **
**                           MODE:         INFORMATION                        **
**                                                                            **
**                                                                            **
********************************************************************************
********************************************************************************

<?xml version="1.0"?>
<build-information>
  <member name="RDBKC01" lang="COBCICD2"
          langdesc="ENTERPRISE COBOL WITH CICS AND DB2">
    <translator name="IGYCRCTL">
      <option-set>
        <option-string>NONUM,LIB,XREF(FULL),MAP,OFFSET,</option-string>
        <option-string>NOOPTIMIZE,CICS(&amp;quot;COBOL3&amp;quot;),SQL</option-string>
      </option-set>
      <data-definition name="SYSLIB">
        <dsn name="ADMIN10.DEV1.COBOL"/>
        <dsn name="ADMIN10.TEST.COBOL"/>
        <dsn name="ADMIN10.PROD.COBOL"/>
        <dsn name="ADMIN10.DEV1.COPYBOOK"/>
        <dsn name="ADMIN10.TEST.COPYBOOK"/>
        <dsn name="ADMIN10.PROD.COPYBOOK"/>
        <dsn name="ADMIN10.DEV1.DCLGEN"/>
        <dsn name="ADMIN10.TEST.DCLGEN"/>
        <dsn name="ADMIN10.PROD.DCLGEN"/>
        <dsn name="CICS.TS31.CICS.SDFHCOB"/>
        <dsn name="CICS.TS31.CICS.SDFHMAC"/>
      </data-definition>
      <data-definition name="DCLGENC">
        <dsn name="ADMIN10.DEV1.DCLGEN"/>
        <dsn name="ADMIN10.TEST.DCLGEN"/>
        <dsn name="ADMIN10.PROD.DCLGEN"/>
      </data-definition>
    </translator>
  </member>
</build-information>
```

Figure 136. Build Information Report

Note that standard XML rules apply to certain special characters, called pre-defined entities. If there are any of the pre-defined entities in the translator options they will be converted to the required XML format name as specified in Table 17 on page 244.

Table 17. Pre-defined entities

Character value	Pre-defined entity value	Description
<	<	left angle bracket
>	>	right angle bracket
&	&	ampersand
'	'	single quote character
"	"	double quote character

The following table contains a description of the elements and attributes that make up the build information XML document.

Table 18. Description of the Elements and Attributes in the Build information XML

Name	Description
build-information	Root element. There is a single *build information* element. **Sub-elements** member **Attributes** None
member	There is one *member* element for each member in the scope of the build information report. For example; if build information is requested on an LEC architecture definition, then there will be a *member* element for each member in the LEC architecture definition. **Sub-elements** translator **Attributes** **name** Name of the member **lang** SCLM language of the member **langdesc** SCLM language description
translator	There is one *translator* element for each translator step in the SCLM language definition. For example; if a COBOL compile has a separate DB2 pre-compile, then there will be a *translator* element for the DB2 pre-compile and a *translator* element for the COBOL compile step. **Sub-elements** option-set data-definition **Attributes** **name** Name of the translator invoked. For example, IGYCRCTL for the COBOL compiler

Table 18. Description of the Elements and Attributes in the Build information XML (continued)

Name	Description
option-set	There is one *option-set* element for each translator. This will contain a number of *option-string* elements depending on how many options there are. **Sub-elements** option-string **Attributes** None
option-string	There are one or more *option-string* elements depending on the number of options specified in the translator. This element contains as many options as will fit within an 80 byte record. **Sub-elements** None **Attributes** None
data definition	There is one *data-definition* element for each IOTYPE=I (Include dependency) allocation in the translator. This will contain a number of *dsn* elements depending on how many data sets are allocated to the specified DD. **Sub-elements** dsn **Attributes** **name** Name of the DD specified in the FLMALLOC macro for an IOTYPE=I allocation. For example; SYSLIB and DCLGEN.
dsn	There is one *dsn* element for each data set that is allocated as part of the concatenation of DDs for the DDNAME specified in the *name* attribute of the *data-definition* element. **Sub-elements** None **Attributes** **name** Data set name

Promote (option 5)

The promote function copies members from any group to the next higher group.

Note: SCLM promote only copies a member over a member at the next level if it has changed. Two members with the same name are considered to be changed if the accounting data and the member statistics are different. If you retrieve the most recent version of a member into the hierarchy, the recovered member at the development group is considered the same as the member residing in the hierarchy. If the member in the hierarchy has been corrupted, but the statistics are still valid, SCLM will not overwrite the existing member during promotion. The promote report indicates that the member was purged but not copied. If you recover the most recent version of a member in order to replace a corrupted member, you must save the member at the development group to refresh the accounting data. You can save the member using SCLM edit, migrate in forced mode, or the SAVE service. Then build and promote the member as usual.

The promote function:

- Determines which components are eligible for promotion
- Verifies that the application is complete and current
- Promotes the components that are at the current group and within the scope of the promote
- Potentially purges the components from the current group (and possibly lower key groups)
- Generates a promote report
- Rebuilds the promoted member at the 'to group', if requested in the language definition

Promote gives you an easy and efficient method to move data through a hierarchy. As you build software components, they become eligible for promotion to the next group in the hierarchy. Promote is based on architecture or source members; thus you must build software components successfully before you can promote them to the next group. Using architecture members, you can promote individual software components or sets of software components during one promote. SCLM processes all data types associated with a component as a unit.

When the promote is complete, the promote function generates a report identifying the components promoted.

The Build function is invoked when the members are copied successfully and any language definitions of members promoted into this group require rebuilding. The promoted member is conditionally rebuilt at the to-group level, as well as any components with the given languages. Other components are not rebuilt. Build messages, listings, and reports are generated based on the values on the SCLM Build - Entry Panel.

You also can specify that only a Promote Report be generated. The Promote Report identifies those components in the hierarchy that would be copied or moved if the promote function were to be invoked.

The panel shown in Figure 137 on page 246 appears when you select Option 5, Promote, from the SCLM Main Menu.

```
  Menu  SCLM  Utilities  Jobcard  Workstation Promote  Help

                          SCLM Promote - Entry Panel

Promote input:
  Project . . . : PDFTDEV
  From group  . : MOS
  Type  . . . . : SOURCE              Enter "/" to select option
  Member  . . . : PROG01              _  Workstation Promote

  Mode  . . 1 1. Conditional          Scope . . . 1 1. Normal
              2. Unconditional                      2. Subunit
              3. Report                             3. Extended

Output control:
               Ex Sub                Process . . _ 1. Execute
  Messages . . 3  3   1. Terminal                  2. Submit
  Report . . . 3  3   2. Printer
                      3. Data set    Printer . . H
                      4. None        Volume  . . ______

Command ===> ____________________________________________________________
 F1=Help      F2=Split    F3=Exit      F7=Backward F8=Forward   F9=Swap
F10=Actions  F12=Cancel
```

Figure 137. SCLM Promote (FLMP#P)

Note: The NRETRIEV command key is enabled to work with this option. See “Name retrieval with the NRETRIEV command” on page 143 for more information.

The fields on the SCLM Promote - Entry panel are:

Project
The project that you specified on the SCLM Main Menu. An Alternate field also appears if you specified an alternate project.

From group
The group from which to promote the member

Type
The type of the member

Member
The name of the member to be promoted

Scope
Select one of the following possibilities:

Normal
To process the components and members directly referenced by the specified architecture member. In addition, this scope processes upward dependencies for all Ada-type source members referenced directly by the architecture member and all source members referenced as upward dependencies.

Subunit
To process the components and members processed in normal scope as well as downward dependencies for all Ada-type source members referenced directly by the architecture members.

Extended
To process the components and members processed in normal scope as well as downward dependencies for all source members within the normal scope.

Note: You must specify a scope equal to or greater than the scope specified with the SCOPE keyword in the FLMLANGL macro.

Mode
Select one of the following possibilities:

Conditional
To bypass the copy and purge steps if promote discovers a verification error.

Promote compares dates in the build maps against dates in the database for all software components taking part in the promote. Software components are not promoted if they are deemed out of date. Use this mode to guarantee complete project integrity.

Unconditional
To perform copy and purge processing of all members despite verification errors of other members and to promote only those members with correct build map information.

Use this mode to promote software components for incomplete or partial applications. For example, if some software components referenced by an architecture member are not complete but are required in the next group of the hierarchy, you can use this mode to promote those software components.

The use of the unconditional mode does not guarantee application integrity, so use it with caution. It is, however, an effective method of promoting dependent software components that you plan to integrate at a later date. The Unconditional mode field is not retained on the Promote panel. If Unconditional is used, the panel is changed to Conditional when the promote returns to the panel.

Report
To perform verification and report generation processing. The report contains a list of members eligible for promotion.

Output control
Specify the destination for messages and the report when they are executed (Ex) or submitted (Sub), by entering the corresponding destination number: 1 for Terminal, 2 for Printer, 3 for Dataset, or 4 for None.

Workstation Promote
Specify whether the promote needs a workstation connection. For Foreground, SCLM verifies that an ISPF workstation connection exists before executing the promote. For Batch, SCLM verifies that the

information required to initiate an ISPF workstation connection has been set by a previous build or promote or from the workstation build pull-down. If not, SCLM prompts the user to enter this information before the build job is submitted. If the promote does not require a workstation connection, do not use this field.

Process
You can call the processing part of the Promote - Entry Utility from the interactive or batch environment by selecting Execute or Submit, respectively. If you request batch processing by selecting Submit, you must specify the job statement information which is used in the JCL generated for batch processing.

For information about using a unique jobname on the jobcard in batch processing, see "Batch Processing" on page 252.

Printer
Specify the printer output class.

Volume
Specify the volume on which SCLM should save data sets.

Promote Report

Figure 138 on page 249 shows an example of the promote report.

The promote report provides an accurate account of the promote. It lists all members promoted to the next group and all members purged from lower groups. It also marks "out-of-scope" software components with an asterisk (*).

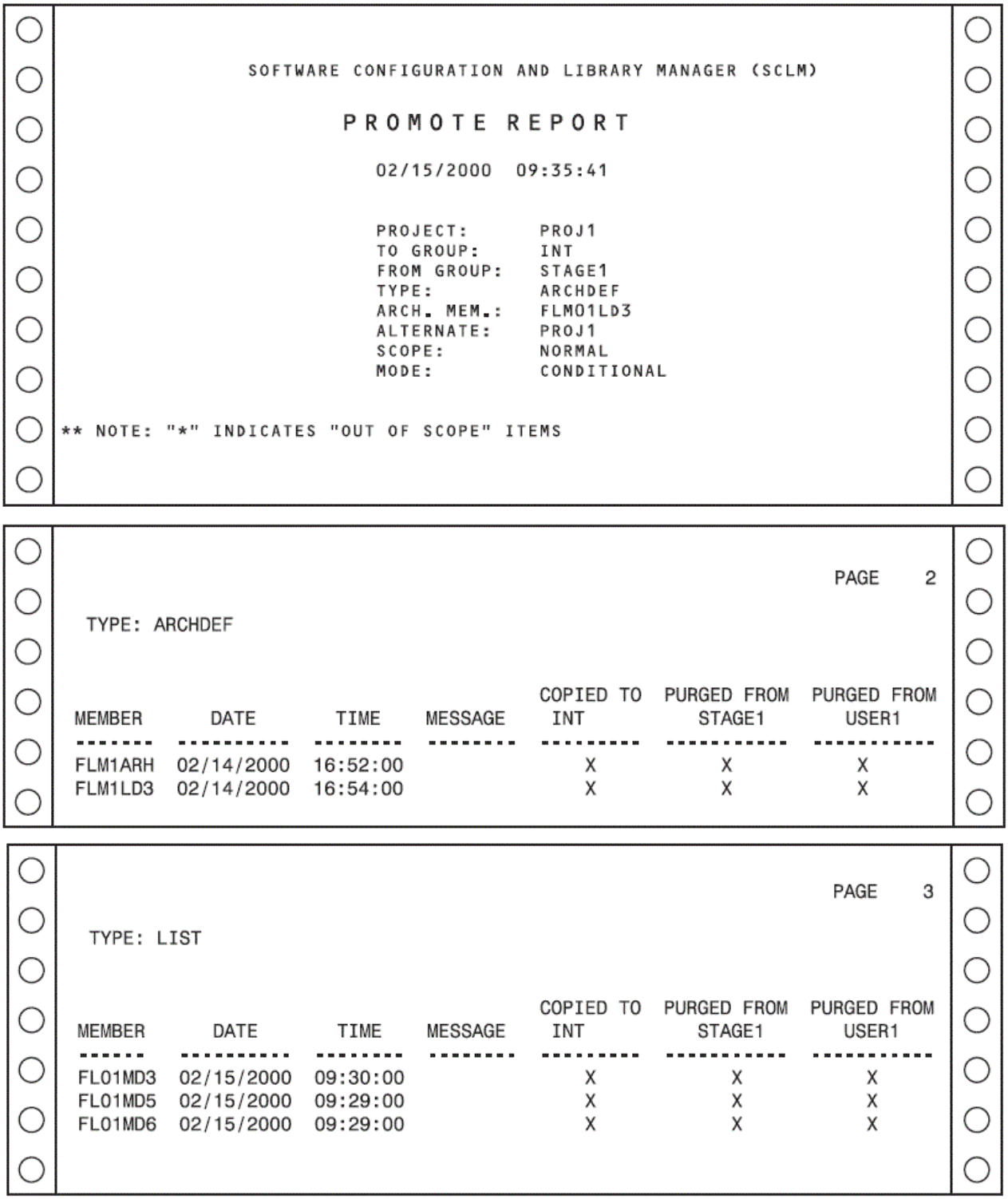
Note: An *out-of-scope* software component is an architecture that is referenced with a LINK statement but not with an INCL statement. It is not within the domain of the architecture specified.

The report displays specific information according to the promote modes and scopes you select.

- For a promote of a member from a non-key group to a key group, the report indicates that the member was:
  - Copied to the next group
  - Purged from the "from" group
  - Purged from the last key group.
- For a promote of a member in a key group to a non-key group, it indicates that a copy was made.
- For a promote of a member in a key group to a key group, it indicates that a copy was made and a purge was performed on the source key group.
- For a second promote that follows a failed promote, it indicates the work completed by that promote only.

For more information about key and non-key groups, see "Key/non-key groups" on page 141.

If a verification error occurs for a member, the report displays the message number that identifies the error in the Message field.



```
                SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)

                          PROMOTE REPORT

                             02/15/2000  09:35:41


                             PROJECT:       PROJ1
                             TO GROUP:      INT
                             FROM GROUP:    STAGE1
                             TYPE:          ARCHDEF
                             ARCH. MEM.:    FLM01LD3
                             ALTERNATE:     PROJ1
                             SCOPE:         NORMAL
                             MODE:          CONDITIONAL

** NOTE: "*" INDICATES "OUT OF SCOPE" ITEMS
```

```
                                                                  PAGE     2

  TYPE: ARCHDEF

                                                    COPIED TO  PURGED FROM  PURGED FROM
MEMBER      DATE        TIME      MESSAGE           INT        STAGE1       USER1
-------     ----------  --------  --------  ---------  ---------  ----------
FLM1ARH     02/14/2000  16:52:00                    X          X            X
FLM1LD3     02/14/2000  16:54:00                    X          X            X
```

```
                                                                  PAGE     3

  TYPE: LIST

                                                    COPIED TO  PURGED FROM  PURGED FROM
MEMBER      DATE        TIME      MESSAGE           INT        STAGE1       USER1
------      ----------  --------  --------  ---------  ---------  ----------
FL01MD3     02/15/2000  09:30:00                    X          X            X
FL01MD5     02/15/2000  09:29:00                    X          X            X
FL01MD6     02/15/2000  09:29:00                    X          X            X
```

Figure 138. Promote Report (Part 1 of 3)

PAGE 4

TYPE: LMAP

MEMBER	DATE	TIME	MESSAGE	COPIED TO INT	PURGED FROM STAGE1	PURGED FROM USER1
FL01LD3	02/15/2000	09:30:00		X	X	X

PAGE 5

TYPE: LOAD

MEMBER	DATE	TIME	MESSAGE	COPIED TO INT	PURGED FROM STAGE1	PURGED FROM USER1
FL01LD3	02/15/2000	09:31:00		X	X	X

PAGE 6

TYPE: OBJ

MEMBER	DATE	TIME	MESSAGE	COPIED TO INT	PURGED FROM STAGE1	PURGED FROM USER1
FL01MD3	02/15/2000	09:30:00		X	X	X
FL01MD5	02/15/2000	09:29:00		X	X	X
FL01MD6	02/15/2000	09:29:00		X	X	X

PAGE 7

TYPE: SOURCE

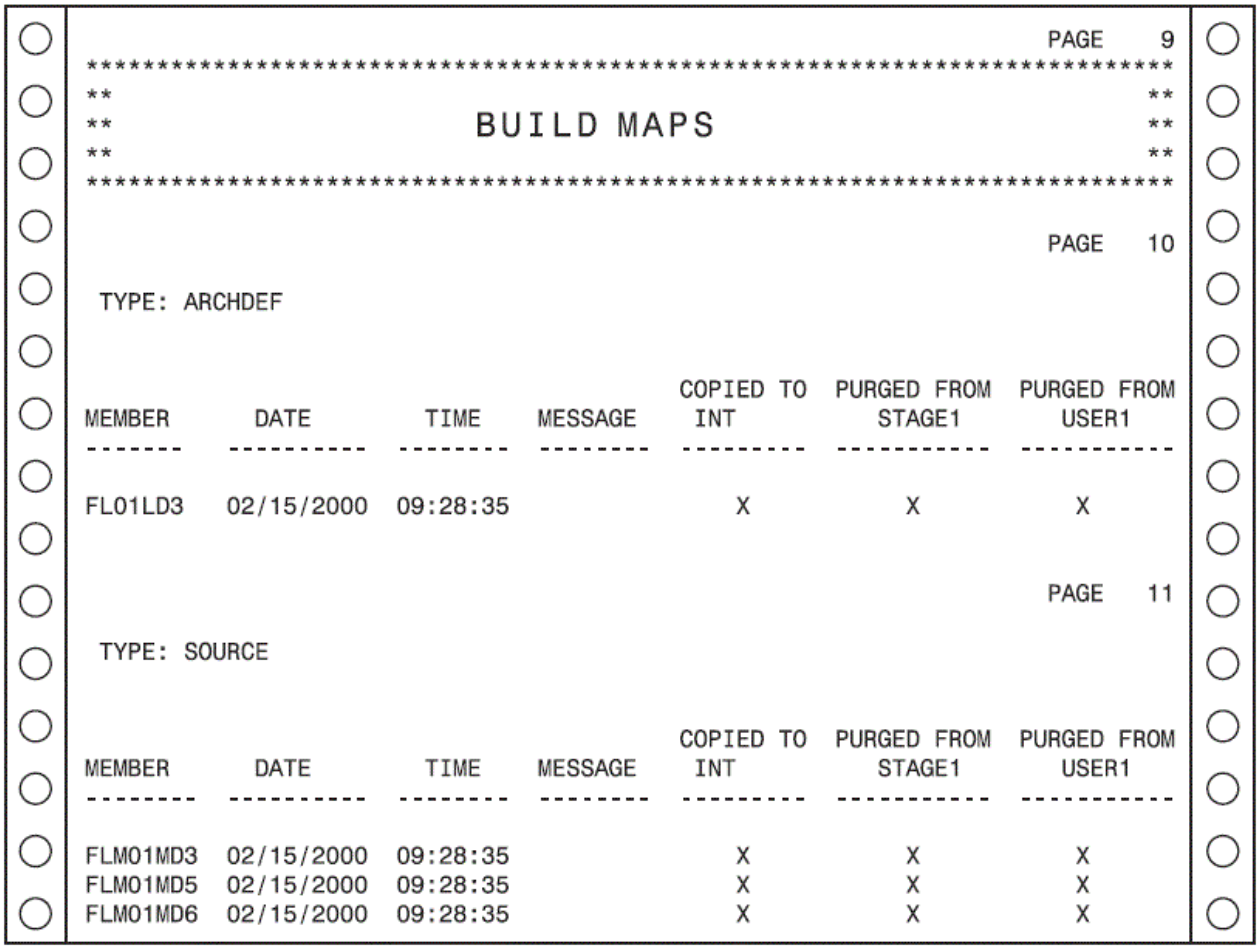
MEMBER	DATE	TIME	MESSAGE	COPIED TO INT	PURGED FROM STAGE1	PURGED FROM USER1
FL01LD3	02/14/2000	16:33:00		X	X	X
FL01LD3	02/14/2000	17:03:00		X	X	X
FL01LD3	02/14/2000	16:48:00		X	X	X

PAGE 8

TYPE: SOURCE2

MEMBER	DATE	TIME	MESSAGE	COPIED TO INT	PURGED FROM STAGE1	PURGED FROM USER1
INCLUDE2	02/14/2000	19:49:00		X	X	X
INCLUDE3	02/14/2000	16:50:00		X	X	X

Figure 139. Promote Report (Part 2 of 3)



```
                                                                        PAGE     9
************************************************************************************
**                                                                                **
**                                BUILD MAPS                                      **
**                                                                                **
************************************************************************************


                                                                        PAGE    10

 TYPE: ARCHDEF


                                                  COPIED TO  PURGED FROM  PURGED FROM
MEMBER      DATE        TIME     MESSAGE           INT        STAGE1       USER1
-------   ----------  --------  --------  ---------  -----------  -----------

FL01LD3   02/15/2000  09:28:35                X          X            X


                                                                        PAGE    11

 TYPE: SOURCE


                                                  COPIED TO  PURGED FROM  PURGED FROM
MEMBER      DATE        TIME     MESSAGE           INT        STAGE1       USER1
--------  ----------  --------  --------  ---------  -----------  -----------

FLM01MD3  02/15/2000  09:28:35                X          X            X
FLM01MD5  02/15/2000  09:28:35                X          X            X
FLM01MD6  02/15/2000  09:28:35                X          X            X
```

Figure 140. Promote Report (Part 3 of 3)

Processing errors

The Promote function can recover from most SCLM environment errors. However, data set overflow and data contention, as described as follows, can occur during a promote.

Data set overflow

Partitioned data sets tend to become full and require compression. When a target data set runs out of space during a promote, promote attempts to recover and continue the promote. Although you get system ABEND messages, the promote ignores the ABEND and continues. However, processing bypasses making a copy to this data set and it also bypasses the subsequent purge step for members that were not copied.

If data set overflow occurs, follow these steps:

1. Compress or reallocate the data set with larger space allocations.
2. Increase the directory block allocation, if necessary.
3. Promote again.

The second promote copies only the members that did not copy in the original promote. If successful, the purge step is normal. The resulting promote report identifies only the copied and purged members in the second promote.

Data contention

Be careful when you process certain combinations of SCLM builds and promotes simultaneously. Do not promote or build members that have not completed processing for another promote. Compiler errors or

promote verification errors in one or more of the concurrent jobs can occur. You can normally recover from most errors by running the failed function again.

Command (option 6)

To use the SCLM command shell, select Command (option 6) from the SCLM Main Menu. The panel shown in Figure 141 on page 252 appears.

```
                              SCLM Command Shell
 Enter TSO or SCLM commands below.

 ===> _____________________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________
_______




  F1=Help    F2=Split   F3=Exit    F9=Swap    F12=Cancel
```

Figure 141. SCLM Command Shell (FLMTSO)

Use this panel to execute TSO, CLIST, REXX execs, or SCLM commands from within SCLM.

Easy Cmds (option 6A)

The Easy Cmds option provides a menu that lists the available FLMCMD services. When you select an option from this menu, ISPF displays a panel that provides data entry fields for the parameters associated with the selected service.

For details about the specific service panels, see the description of the relevant service in Chapter 19, “SCLM services,” on page 379.

Batch Processing

The Verify Batch Job Information panel shown in Figure 142 on page 253 is the standard panel for the SCLM functions that allow you to select batch processing. When you enter SUBMIT and when the JOB statement is not on the submittal panel, this panel appears. SCLM requires JCL job statements when you process in batch mode.

Note: SCLM can automatically generate unique jobnames. If you use the jobname USERIDx, where x is a letter of the alphabet or a digit, SCLM increments this letter or number by one for the next job. For example, if your USERID is SMITH, and your jobcard is submitted with the jobname SMITH3, the jobname is updated to SMITH4.

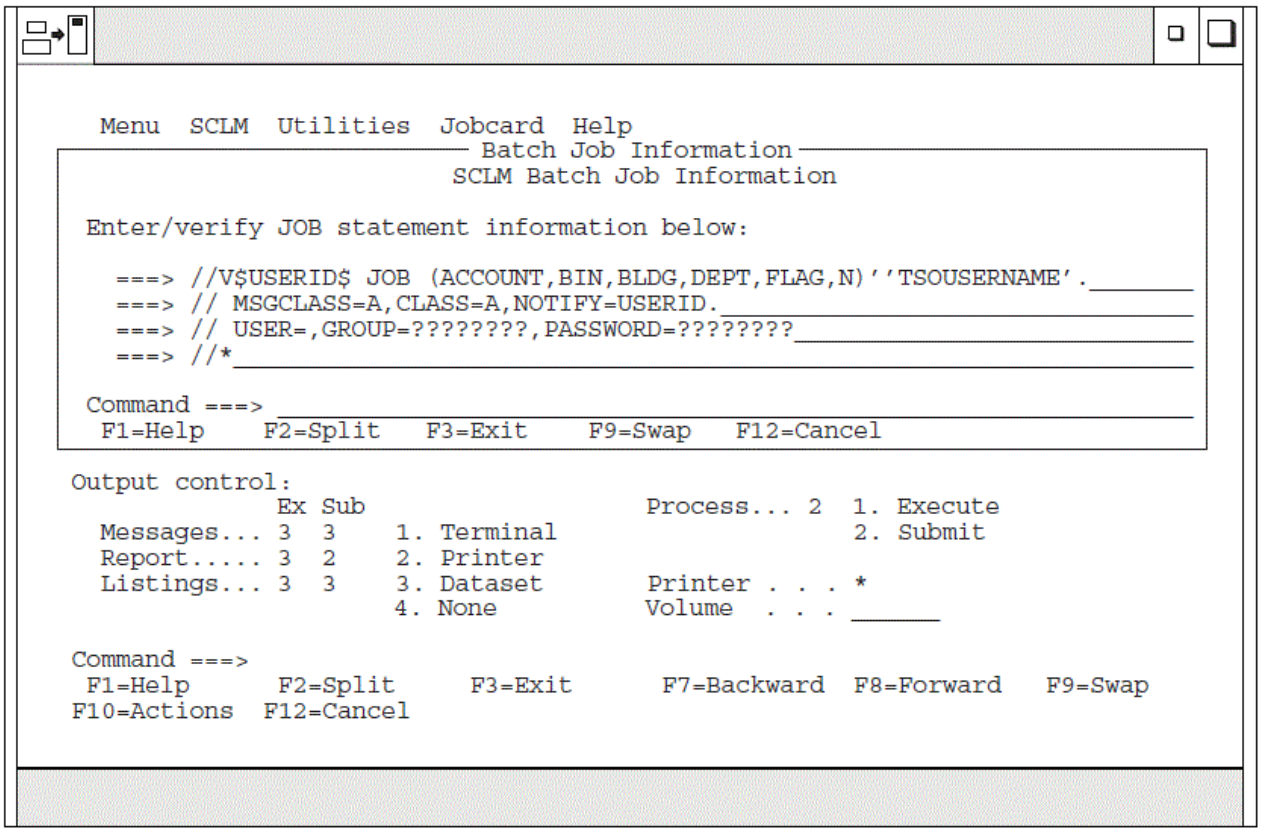


Figure 142. Verify Batch Job Information (FLMDSU#P)

Output disposition

The Output Disposition panel shown in Figure 143 on page 254 is the standard end panel for many SCLM functions when you have sent output to a data set. It allows you to determine the disposition of the report or messages data set previously displayed. You can choose between keeping the data set, deleting the data set, printing and keeping the data set, or printing and deleting the data set.

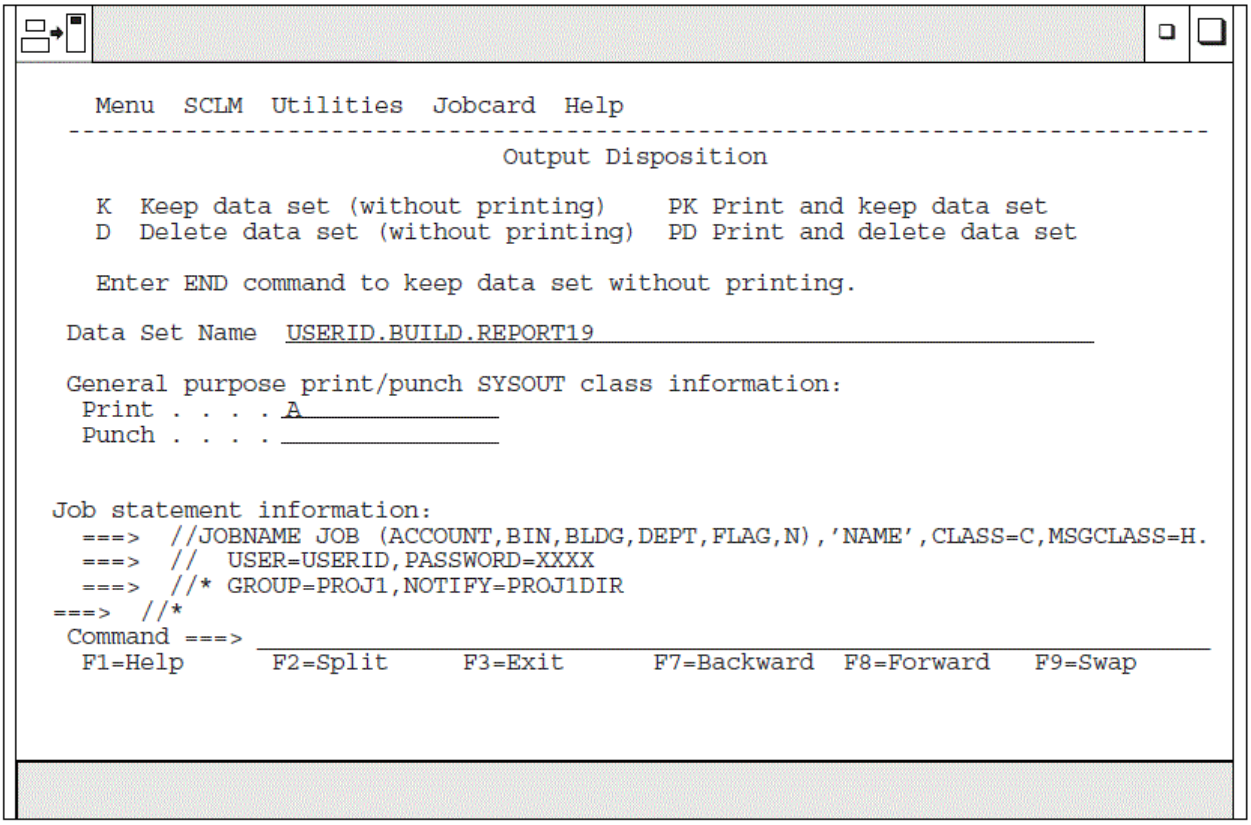


Figure 143. Output Disposition (FLMDEXT)

When you send output to a data set, the database contents, architecture, build, and promote functions display a report data set if they complete with an acceptable return code. The migration utility displays a message data set because its report is a set of messages.

If you allocate the output to a data set and 99 data sets have already been allocated, SCLM either overlays a new data set over an old one or concatenates a new data set with an old one. To avoid this problem, delete old data sets to allow allocation of new data sets.

If error conditions occur in any of these functions (except build translator errors) and SCLM routes messages to a data set, SCLM displays the message data set, not the report data set. In either case, the Output Disposition panel appears after you finish browsing the displayed data set.

The view, edit, library, sublibrary management, and audit and version utility functions do not create report or message data sets and, consequently, do not display the Output Disposition panel.

Sample Project Utility (option 7)

The SCLM Sample Project Utility makes it easier to create a sample SCLM project to use in learning the functions of SCLM, or as the basis for building a project for production use. In addition, you can use the Sample Project Utility to delete a project that was built using the utility.

The SCLM Sample Project Create function, Option 10.7.1, creates the data sets required for a simple SCLM project (including the VSAM accounting data base). It also creates a data set listing information about the project.

You must provide the names of several existing data sets on your system (such as, the ISPF macros data set), and the location of the High Level Assembler on your system.

- If you have the Enterprise PL/I Compiler installed on your system, you have a choice of including a PL/I sample.

- If you have the Enterprise COBOL Compiler installed on your system, you have a choice of including a COBOL sample.
- If Fault Analyzer is installed on your system, you have a choice of generating a side file for member FLM01AD9.

You do not need knowledge of assembler or link-editing. The utility customizes, assembles, and link-edits the project definition for you. The architecture definitions are then imported from the ISPF sample library and the sample application is built and promoted to the top level of the hierarchy. The project is then ready to use for the scenario described in Chapter 10, “Development scenario,” on page 257. Use this scenario to learn the capabilities of SCLM.

The SCLM Sample Project Delete function, Option 10.7.2, deletes a project that was created with the Create utility. This function uses the information data set created by the Create utility to identify the data sets to delete.

Maintaining SCLM administrators (option A)

This option is used to add and delete SCLM administrators for a project. SCLM administrators are allowed to maintain any locked members and to transfer ownership of a member to another user ID.

The SCLM Admin option is only available to users who are already SCLM administrators. When member level locking is first enabled, the only user who can access this option is the user whose ID is specified in the ADMINID parameter on the FLMCNTRL macro in the project definition.

To maintain the list of administrators for a project, specify the project high-level qualifier on the SCLM Main Menu and select option A.

Chapter 10. Development scenario

This chapter uses a sample application to describe the basic tasks you typically perform using SCLM. The sample data sets referred to in the example are included with ISPF.

Chapter 1, “Defining the project environment,” on page 3 provides step-by-step instructions for the project manager to define the sample project for this scenario. You can also define the sample project using Option 10.7, the SCLM Sample Project utility. No knowledge of SCLM is required to use the utility. You can use this hierarchy to gain some basic experience using SCLM. After examining some of the project data sets and performing some SCLM operations, you will have a better understanding of how SCLM can help you in your project activities.

This chapter walks you through the functions from the SCLM Main Menu. For a complete description of the SCLM Main Menu options, see Chapter 9, “Using SCLM functions,” on page 143.

Understanding the hierarchy and the SCLM main menu

This section provides an overview of the sample hierarchy and briefly describes the functions available from the SCLM Main Menu.

The sample project uses a three-layer hierarchy composed of four groups. This figure shows the SCLM hierarchy in this sample.

Figure 144. Sample Project Hierarchy

Throughout the remainder of this chapter, this sample project is called PROJ1. If the name established by your project manager is different, or you used a different name to define the project using the SCLM Sample Project utility (Option 10.7), use that name instead.

The sample application is composed of 9 programs that are used to build an application called FLM01AP1, as shown in Figure 145 on page 258. The programs are linked into 6 load modules. The 6 load modules are organized as 2 subapplications, which in turn are components of FLM01AP1.

Note: If neither the Enterprise PLI Compiler nor Enterprise COBOL Compiler are included as a language in the sample project, the application consists of 6 programs linked into 4 load modules.

The sample that follows assumes that the SCLM project setup activities have been completed as described in Chapter 1, “Defining the project environment,” on page 3 or that you have defined the sample project using the SCLM Sample Project utility (Option 10.7).

© Copyright IBM Corp. 1990, 2019

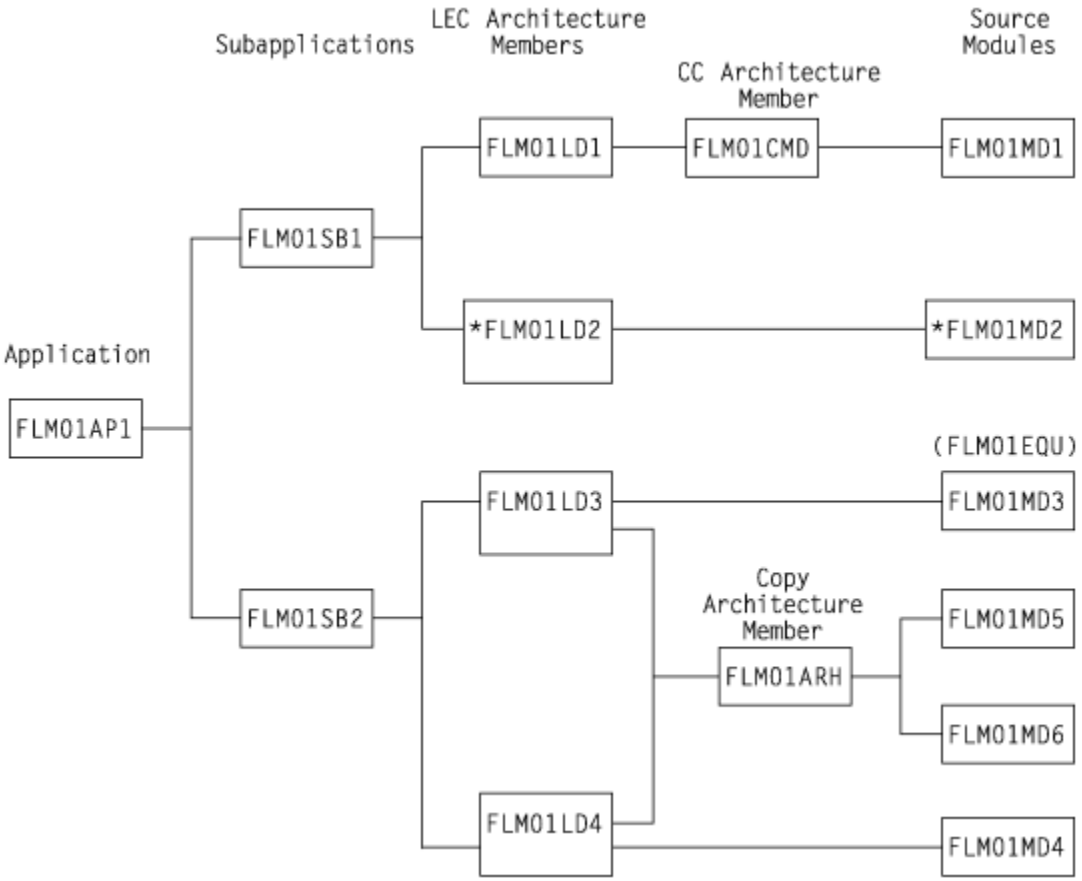


Figure 145. Application FLM01AP1

Note: Source module FLM01MD2 and architecture member FLM01LD2 are included only if Enterprise PL/I Compiler is included as a language if the sample is defined using the SCLM Sample Project utility (Option 10.7).

Source modules FLM01CD7,FLM01CD8, and architecture member FLM01LD7 are included only if Enterprise COBOL Compiler is included as a language if the sample is defined using the SCLM Sample Project utility (Option 10.7).

After the sample project has been defined, you can take the following steps to begin using SCLM.

1. Log on to MVS.
2. Start ISPF to display the ISPF Primary Option Menu.
3. Select SCLM and press Enter. The SCLM Main Menu is displayed.

Understanding the architecture definition

This section describes the architecture definition and its importance in an SCLM project. The architecture definition describes to SCLM how the components of an application fit together. For more information about architecture definitions, see Chapter 11, “Architecture definition,” on page 271.

There are four types of architecture members:

HL (high level)
HL architecture members reference application and subapplication components.

CC (compilation control)
CC architecture members contain the information to produce and track software components with object module output.

LEC (link-edit control)
LEC architecture members contain the information to produce a complete load module.

Generic
Generic architecture members identify the source member or groups of source members to be processed by a processor other than a standard compiler. The sample project does not contain examples of generic architecture members.

If you have several architecture definition statements that are used together in many places, you can put them into a member and reference the member using the COPY statement wherever you need the statements. When you use the COPY statement, the contents of the specified member are inserted directly into the respective architecture members.

1. Select View from the SCLM Main Menu. Specify PROJ1 in the Project field and specify DEV2 in the Group field. Press Enter.
2. Specify ARCHDEF in the Type field and leave the Member field blank. Press Enter. The architecture members are shown in the following table.

Table 19. The architecture members

Member	Type	Comments
FLM01AP1	HL	References FLM01SB1 and FLM01SB2 with the INCL statement. A build performed on FLM01AP1 results in a complete build for all the code in the project, if necessary.
FLM01SB1	HL	References FLM01LD1,FLM01LD2,FLM01LD7, and FLM01LD9 with the INCL statement. A build performed on FLM01SB1 results in a complete build of the FLM01SB1 subapplication, if necessary. If the Enterprise PL/I Compiler is not included as a language in the sample project, FLM01SB1 does not reference FLM01LD2. If the Enterprise COBOL Compiler is not included a language in the sample project, FLM01SB1 does not reference FLM01LD7.
FLM01SB2	HL	References FLM01LD3 and FLM01LD4 with the INCL statement. A build performed on FLM01SB2 results in a complete build of the FLM01SB2 subapplication, if necessary.
FLM01LD1	LEC	Directs SCLM to produce the load module and load map for FLM01LD1. The INCL statement references architecture member FLM01CMD. The PARM statements pass parameters to the SCLM BUILD translators.
FLM01LD2	LEC	Directs SCLM to build load module FLM01LD2 from the source FLM01MD2. The INCLD architecture statement is used to identify FLM01MD2 as the source. Note that LOAD, LMAP, and SOURCE are types identified by the FLMTYPE macro in the project definition. If the Enterprise PL/I Compiler is not included as a language in the sample project, FLM01LD2 is not included.
FLM01LD7	LEC	Directs SCLM to build load module FLM01LD7 from the source FLM01CD7 and FLM01CD8. The INCLD architecture statement is used to identify FLM01CD7 and FLM01CD8 as the source. Note that LOAD, LMAP, and SOURCE are types identified by the FLMTYPE macro in the project definition. If the Enterprise COBOL Compiler is not included as a language in the sample project, FLM01LD27 is not included.
FLM01LD9	LEC	Directs SCLM to produce the load module and load map for FLM01LD9. The INCL statement references architecture member FLM01CM9. The PARM statements pass parameters to the SCLM BUILD translators.

Table 19. The architecture members (continued)

Member	Type	Comments
FLM01CMD	CC	Directs SCLM to produce object code from FLM01MD1. SINC identifies FLM01MD1 as the source member. Note that in addition to object code (OBJ), there is also source listing (LIST). OBJ and LIST are identified in the project definition with the FLMTYPE macro.
FLM01CM9	CC	Directs SCLM to produce object code from FLM01AD9. SINC identifies FLM01AD9 as the source member. Note that in addition to object code (OBJ), there is also source listing (LIST). OBJ and LIST are identified in the project definition with the FLMTYPE macro.
FLM01LD3	LEC	References FLM01MD3 with the INCLD statement. Other modules are referenced with the copy of FLM01ARH. In this example, FLM01ARH references FLM01MD5 and FLM01MD6. FLM01LD3 indirectly references FLM01MD5 and FLM01MD6 via the COPY statement in FLM01ARH.
FLM01LD4	LEC	References FLM01MD4 with the INCLD statement. Other modules are referenced with the copy of FLM01ARH. In this example, FLM01ARH references FLM01MD5 and FLM01MD6. FLM01LD4 indirectly references FLM01MD5 and FLM01MD6 via the COPY statement in FLM01ARH.
FLM01ARH	CC	References modules FLM01MD5 and FLM01MD6 with the INCLD statement. The LEC architecture members FLM01LD3 and FLM01LD4 use the COPY directive to copy the contents of FLM01ARH into their members for a build.

To create an architecture report:

1. Select Architecture Report (option 3.5) from the SCLM Main Menu, and press Enter.
2. Type:

   ARCHDEF
   in the Type field

   FLM01AP1
   in the Member field

   6
   in the "Report cutoff" field

   1
   in the Process field

   1
   in the Messages field

   1
   in the Report field

   Press Enter.

The output shows the hierarchy, the kinds of architecture members (HL, CC, and LEC), and various cross-references. See “Architecture Report example” on page 190 for an example of the architecture report.

Sample SCLM development cycle

Your typical daily operations using SCLM might flow like this: edit (SCLM editor), compile (Build), and test, repeating this cycle until testing is complete, and then promote. After the promote is performed, you or other developers can use the SCLM editor to automatically draw members down to a development group for modification.

The following list includes steps that you might perform in the development cycle of a software component or any type of data that is under SCLM control. Figure 146 on page 262 illustrates the project flow of the following steps. The hierarchy used for this example is shown in Figure 144 on page 257.

1. The developer draws down a source member from group RELEASE to group DEV1 and modifies it. The data at group RELEASE is the current release of the project. Changes are now being made for the next release. When the developer has made the modifications to the member, SCLM parses the member and registers it with SCLM. The successful registering of the update makes this member available for use by other SCLM functions.
2. The Build function is initiated against an architecture definition that includes this parsed and stored source member. This build creates object modules reflecting the changes that were made to the source member. The source, architecture definition, and object module members used here have been given the same member names. Thus, you can easily see how these members are related, although their types are different. These naming conventions, however, are not required by SCLM.

   If the Build function does not complete successfully because of errors in the modified members, you must use the SCLM editor again to correct the errors, and try to build again.
3. The developer can now test the effect the changes have made to the application.
4. The developer then moves all the changed data to the group TEST by invoking PROMOTE using the same architecture definition that was previously built. The data changes are now available to all developers because they have reached a common group. If any changes in data made by the developer conflict with changes other developers are making in their development groups, these changes are found when the other developers build their changes at their development group.

   Alternately, the person appointed as SCLM project manager can do the promote. The SCLM project manager is the person who has UPDATE authority to TEST and promote changes to this group. The SCLM project manager can guarantee all changes promoted to the group TEST have been unit tested (because the project manager can control the promotes).
5. When all changes scheduled for the next release have been promoted to the group TEST, testing the application can occur at this group while other programmers are still developing software in the development groups.
6. Finally, after system testing is complete in the TEST group, the new release of the project can be promoted to the RELEASE group.

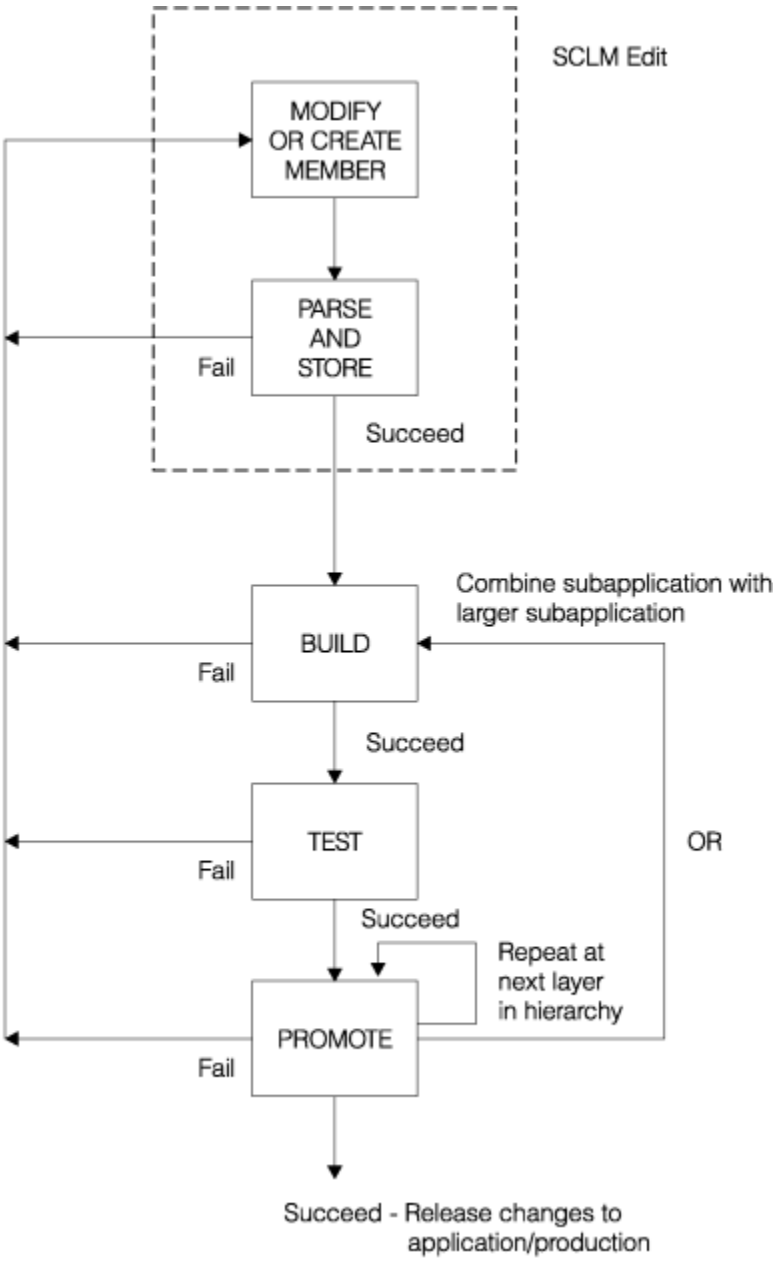


Figure 146. Development Cycle

Using the SCLM editor

This section describes how to alter code using the SCLM editor. To illustrate how SCLM protects project members from unintentional updates, you will change the FLM01EQU member and create an error situation. This error causes the BUILD to fail and prevents a PROMOTE until you correct the error.

FLM01EQU is an included member in FLM01MD3. SCLM automatically tracks included members, so you do not have to specify their relationship in your architecture definition.

1. Return to the SCLM Main Menu, and specify DEV2 in the Group field. Select the Edit option and press Enter.
2. Select SOURCE in the Type field and FIX01 in the "Change code" field. Press Enter to open the Edit Member list.
3. Select FLM01EQU from the Edit Member list. Note that FLM01EQU is in the RELEASE group and a draw down from the RELEASE group to the DEV2 group takes place.

4. From the command line, issue the SETUNDO ON command. Different system installations will have different profile defaults set, so issuing this command will ensure that you have PDF Edit UNDO set On.
5. Duplicate the line R4 EQU 4 and change WORK REGISTER in the comment to DEV2 ERROR. Press Enter.
6. From the command line, issue UNDO: type Undo on the command line and press Enter. The change to the comment is removed. The duplicate line remains. Note that UNDO works only if your profile has UNDO set to ON.
7. Reenter the change to create the error situation for this example from step "4" on page 263.
8. Use the split screen option. Select SCLM from the ISPF Primary Option Menu. Select Edit, specify PROJ1 in the Project field, and specify DEV1 (DEV1 is another development group in this SCLM project) in the Group field.

   Attempt to edit FLM01EQU by typing FLM01EQU in the Member field and pressing Enter. Press the Help key twice to retrieve the long message describing the error condition. SCLM locked FLM01EQU for DEV2 at the time of the draw down. FLM01EQU cannot be updated by another group until a PROMOTE is issued from DEV2 or FLM01EQU (member and accounting record) is deleted from DEV2. End split screen.
9. Return to the DEV2 edit screen and issue the SPROF edit command: type SPROF on the command line and press Enter. Note that the language is ASM and the change code is FIX01. SCLM prompts you for a language when a member is created. You can use SPROF to change the language SCLM associates with the member. Press Enter to return from the SCLM Edit Profile Panel to the SCLM Edit panel.
10. Press the End key to save the member and end the edit session. Use the Help key to display the long message, which indicates that SCLM parsed and stored the member.

    Press the End key twice to return to the SCLM Main Menu.

Understanding the library utility

This section describes the library utility functions typically used by developers. You can use the library utility to browse and delete components and the accounting information that is generated with edit/save, build, and promote activities.

1. Select Utilities from the SCLM Main Menu, and press Enter.
2. Select Library, and press Enter.
3. To browse the accounting record for PROJ1.DEV2.SOURCE(FLM01EQU), type:

   A
   on the command line

   DEV2
   in the Group field

   SOURCE
   in the Type field

   FLM01EQU
   in the Member field

   Press Enter.

   Notice the date and time of the last update ("Change date" and "Change time" fields) for FLM01EQU.
4. To display the statistics, select the "Display statistics" field and press Enter.
5. Return to the accounting record by pressing the End key once. Note that the FLM01EQU has one change code. To display the change code, select the "Number of change codes" field and press Enter. The change code FIX01 appears along with the Change date and Change time.
6. Return to the Library Utility panel by pressing the End key twice.
7. To browse the member PROJ1.RELEASE.SOURCE(FLM01MD3), type:

B
on the command line

RELEASE
in the Group field

FLM01MD3
in the Member field

Press Enter.

Notice that FLM01MD3 contains a COPY statement for FLM01EQU.

8. Press the End key until you are back at the SCLM Main Menu.

Using Build

This section illustrates how to use the SCLM build processor when one of the members has an error. The SCLM build processor translates all members and all modules that have been affected by alterations. A build operation prepares the member for a promote operation.

1. Select the Build option from the SCLM Main Menu, and press Enter.
2. Execute a Build operation by typing:

   DEV2
   in the Group field

   ARCHDEF
   in the Type field

   FLM01AP1
   in the Member field

   /
   in the "Error listings only" field

   1
   in the Mode field

   2
   in the Scope field

   1
   in the Messages field

   1
   in the Report field

   3
   in the Listings field

   Press Enter.

   Notice that you did not have to type EX on the command line or re-enter a value in the Process field. You set this value when you created the Architecture Report. The value is carried from panel to panel and is maintained as is until you change it.
3. Note the return code of 8 from the assembler. There is also an error from the translator for FLM01MD5, which contains FLM01EQU. The assembler listing is contained in *userid*.BUILD.LIST*nn*.

   Because of the assembler error, SCLM Build will place you in Browse of the LISTING data set (*userid*.BUILD.LIST*nn*). Note that the error is the duplicate symbol R4.

   If you are using a tso-prefix that is not your user ID, the data set name will be **tso-prefix.userid.BUILD.LISTnn**.
4. When you are finished browsing the LISTING data set, press the End key. The Output Disposition panel appears. Type D to delete the LISTING data set, or type K to keep the LISTING data set. After pressing Enter, the Build panel appears.

Because the FLM01EQU member has changed and because FLM01MD5 contains the FLM01EQU member, Build attempts to assemble and link FLM01MD5. However, FLM01EQU contains the error you previously entered (a duplicate symbol for R4) so nothing is assembled or linked.

Editing the member to correct errors

This section describes how to re-edit the FLM01EQU member to correct the error you introduced previously.

1. Select Edit from the SCLM Main Menu, leave PROJ1 in the Project field and DEV2 in the Group field. Press Enter.
2. Specify FLM01EQU to edit the FLM01EQU member in PROJ1.DEV2.SOURCE.
3. Remove the duplicate R4 equate line.
4. Save the changes by pressing the End key.

Attempting to promote a member before performing a build

This section describes how SCLM protects the integrity of your project hierarchy by not allowing you to promote a member that has not been successfully built. The promote operation copies changed members up into the next group in the library structure.

The build operation you attempted previously was unsuccessful. Therefore, the promote you attempt in this section will also be unsuccessful. SCLM maintains synchronization between source and object by ensuring that only successfully built members can be promoted. This safety feature addresses the common problem of forgetting to recompile changed modules.

1. Select Promote from the SCLM Main Menu.
2. On the Promote panel, type:

   DEV2
   in the "From group" field

   ARCHDEF
   in the Type field

   FLM01AP1
   in the Member field

   1
   in the Mode field

   1
   in the Scope field

   1
   in the Messages field

   1
   in the Report field

   Press Enter.

SCLM issues date and time mismatch error messages because the FLM01EQU source has been updated and the modules that use it have not been recompiled by the build operation. Promote sends a return code of 8 because the date and time mismatch prevented it from copying anything to the next group.

Rebuilding the changed member

This section illustrates a successful build operation. Because all members are not affected by the change to the FLM01EQU member, only the members containing FLM01EQU are recompiled and linked. SCLM

processes project components efficiently by recompiling and relinking only those modules that were altered since the last build operation.

1. Select Build from the SCLM Main Menu and press Enter.
2. On the Build panel, type:

   DEV2
   in the Group field

   ARCHDEF
   in the Type field

   FLM01AP1
   in the Member field

   1
   in the Mode field

   2
   in the Scope field

   1
   in the Messages field

   1
   in the Report field

   3
   in the Listings field
3. Press Enter.
4. Note that:
   - The traversal of the architecture FLM01MD2 was not affected by the change to the FLM01EQU member and will not be recompiled. FLM01LD2, which contains only FLM01MD2, will not be relinked.
   - FLM01CD7 was not affected by the change to the FLM01EQU member and will not be recompiled. FLM01LD7, which contains only FLM01CD7 and FLM01CD8, will not be relinked.
5. Verify that the build completed successfully (RETURN CODE = 0). If the return code is not zero, check the listing, correct the errors, and try again.

Using the Database Contents Utility

This section illustrates use of the database contents utility to verify that the compilations and links were performed.

1. Select the Utilities option from the SCLM Main Menu.

   Select the Database Contents Utility option from the SCLM Utilities Menu.
2. On the Database Contents Utility panel, type:

   DEV2 TEST RELEASE
   in the Group fields

   SOURCE
   in the Type field

   *****
   in the Member field

   /
   in the "Change additional selection criteria" field

   1
   in the Messages field

1
in the Report field

3
in the "Tailored output" field

Press Enter. The Additional Selection Criteria panel appears.

3. On the SCLM Database Contents - Additional Selection Criteria panel, type * for the "Authorization code", "Change code", "Change group", "Change user id", and Language fields. Do not select the "First occurrence only" field.

   Type:

   1
   in the "Data type" field

   3
   in the "Architecture control" field

   1
   in the Scope field

   These are the default values.

   Press Enter. The Customization Parameters panel appears.

4. On the Customization Parameters panel, select the "Page headers" and "Show totals" fields, and enter `Statistics Report` for the "Report name" field. Type `@@FLMMBR @@FLMLAN @@FLMCML @@FLMNCL @@FLMBLL @@FLMTLS @@FLMCMS @@FLMNCS` for the "Report line format" field after the prompt.

   Put at least 2 spaces between each @@FLMxxx variable. This can wrap to the next line; this field accepts up to 160 characters. These are the default values. Press Enter to execute the database contents utility report.

   Note that only FLM01EQU is in the DEV2 group. The Database Contents Utility panel reappears.

5. On the Database Contents Utility panel, type:

   DEV2 TEST RELEASE
   in the Group fields

   OBJ
   in the Type field

   Do not select the "Change additional selection criteria" field.

   Press Enter. Press Enter again on the Customization Parameters panel.

   Note that FLM01LD2,FLM01LD7,and FLM01LD8 do not appear in the DEV2 group. These modules were not affected by the changes to FLM01EQU.

6. On the Database Contents Utility panel, type:

   DEV2 TEST RELEASE
   in the Group fields

   LMAP
   in the Type field

   Press Enter. Press Enter again on the Customization Parameters panel.

   Note that FLM01LD2,FLM01LD7,and FLM01LD8 do not appear in the DEV2 group. These modules were not affected by the changes to FLM01EQU.

7. On the Database Contents Utility panel, type:

   DEV2 TEST RELEASE
   in the Group fields

   LOAD
   in the Type field

Press Enter. Press Enter again on the Customization Parameters panel.

Note that FLM01LD2,FLM01LD7,and FLM01LD8 do not appear in the DEV2 group. These modules were not affected by the changes to FLM01EQU.

Promoting a member successfully

This section illustrates a successful promote operation. The FLM01EQU member is moved from the DEV2 group to the TEST group.

1. Select the Promote option from the SCLM Main Menu, and press Enter.
2. On the Promote panel, type:

   DEV2
   in the "From group" field

   ARCHDEF
   in the Type field

   FLM01AP1
   in the Member field

   1
   in the Mode field

   1
   in the Scope field

   1
   in the Messages field

   1
   in the Report field

   Press Enter.
3. Verify that the promote completed successfully (RETURN CODE = 0). If the return code is not zero, check the messages, correct the errors, and try again.

   When the Promote panel reappears, press the End key to return to the SCLM Main Menu.
4. Select the Utilities option from the SCLM Main Menu.

   Select Database Contents Utility from the SCLM Utilities Menu. On the Database Contents Utility panel, type:

   DEV2 TEST RELEASE
   in the Group fields

   *****
   in the Type field

   FLM01EQU
   in the Member field

   1
   in the Messages field

   1
   in the Report field

   4
   in the "Tailored output" field

   Do not select the "Change additional selection criteria" field.

   Press Enter. The Database Contents Utility panel reappears.
5. On the Database Contents Utility panel, type:

DEV2 TEST RELEASE
in the Group fields

SOURCE
in the Type field

*****
in the Member field

/
in the "Change additional selection criteria" field

1
in the Messages field

1
in the Report field

4
in the "Tailored output" field

Press Enter. The Additional Selection Criteria panel is displayed.

Type FIX01 in the "Change code" field. Press Enter again. Only FLM01EQU should be found, and it should only be found at TEST. The Database Contents Utility panel reappears.

6. Return to the SCLM Main Menu by pressing the End key twice.

Drawing down a promoted member

This section illustrates that a promoted member is available and can be edited by other developers.

1. Specify Edit from the SCLM Main Menu, PROJ1 in the Project field, and DEV1 in the Group field.
2. Edit the FLM01EQU member, by specifying SOURCE in the Type field and FLM01EQU in the Member field. However, do not make any changes to the member. Note that FLM01EQU is no longer locked by SCLM.

Performing project housekeeping activities

After you complete the development activities described in this chapter, be sure to perform any cleanup or housekeeping activities in preparation for the next project operations. You can clean up the sample project hierarchy by performing a promote operation using group TEST, type ARCHDEF, and member FLM01AP1. This restores the hierarchy to its original state so that others can use it to execute this scenario. If you made other changes (such as a change to the FLM01EQU member in the last activity), you might need to perform additional build and promote operations.

You can also delete the *tso-prefix*.BUILD.LIST*nn* and *tso-prefix*.DBUTIL.CMD*nn* data sets created during the preceding SCLM Build process.

Chapter 11. Architecture definition

An architecture definition describes the configuration of an application under SCLM control and how it is to be constructed and integrated. Architecture definitions are created and updated by the developers and describe the architecture of an application. They provide specifications to the Build function for data generation, and to the Promote function for the movement of data from one group to another. Architecture definitions can reference other architecture definitions, thus providing a simple building block tool for complex application definitions.

- Data Generation

  Architecture definitions can specify the following information to the build function:

  – Where inputs to translators (for example, compilers) are to come from
  – Where outputs from translators are to be stored
  – What parameters are needed by a translator

  A single architecture definition can specify all the data generation to occur for a large, complex application simply by referencing other architecture definitions.

- Data Movement

  All data that is directly or indirectly referenced by an architecture definition is promoted when that architecture definition is promoted. This encompasses included architecture definitions, along with the system components they describe. Thus, specifying a single high-level architecture definition for promotion can cause an entire application to be promoted.

This chapter discusses the methods you can use to define the architecture, provides several different examples of architecture members, and explains the use of architecture member statements.

Architecture members

Architecture members define the application at a high level by referencing lower level architecture members. You can generate them top down or bottom up, using an iterative approach. Create architecture members by using the edit function.

The capability to define an architecture allows you to control and track any discrete division of an application from the most encompassing definition down to the individual component. You can maintain the architecture members in a separate type in the project data base. Use the architecture members to describe the different versions or variations of a project or application.

Kinds of architecture members

SCLM provides four kinds of architecture member that you can use to generate an architecture definition for an application. Each kind of architecture member controls a different kind of component that SCLM processes.

Table 20. Uses of Architecture Members

Architecture Member	Use
Compilation Control (CC)	Define compiler processed components.
Linkedit Control (LEC)	Define link-edit processed components.
High-Level (HL)	Define application and subapplication components.
Generic	Define specially processed components.

© Copyright IBM Corp. 1990, 2019

Each of these uses is described in the following pages. See "Sample application using architecture definitions" on page 283 for an example of an application consisting of architecture members.

Defining compiler processed components

Standard compilers produce object modules as output. SCLM can be used to create object modules by using either a Compilation Control (CC) architecture member or a compilable source member as input to the build function. This topic discusses both methods for producing object modules.

Compilation control architecture members

One method of creating object modules is through a Compilation Control (CC) architecture definition.

CC architecture definitions contain all the information necessary to produce and track software components with object module output. Use CC architecture definitions to provide the following information:

- The inputs to the compiler and other translators
- The outputs of the compiler and other translators
- Compiler options.

To directly identify an input to the compiler, use the SINC statement. If the input is generated from another member in the project, use the INCL and INCLD statements along with the KREF statement. The INCL and INCLD statements identify members built before compiling this member. The KREF statement identifies which outputs of the members on the INCL and INCLD statements are inputs.

CC architecture members must have at least one SINC statement and one OBJ statement. See "Architecture statements" on page 277 for more information.

Members included by compiler include statements such as COPY are not identified in architecture members. SCLM obtains the list of included members from a parser that is run when a member is stored into SCLM and when members are updated. The information about the parser, the compiler, and include libraries outside the project is specified in a language definition. The language of a member must be identified to SCLM when a member is added to an SCLM project. The language of a member can be changed.

The ddnames used by the compiler are specified in the language definition by FLMALLOC macros. The types of ddnames are identified by different IOTYPEs. An IOTYPE of S identifies the input stream for the compiler. The input stream has two formats. One, identified by KEYREF=SINC, is a sequential work file that contains all of the inputs to the compiler concatenated together. The other, identified by KEYREF=INCL, is a sequential work file that contains INCLUDE statements for each of the input members. The format of the INCLUDE statement is `INCLUDE DDNAME(MEMBER)`. The DDNAME will be a ddname dynamically allocated by SCLM. If multiple inputs are identified, they are concatenated in the order specified in the architecture member.

You can add information to the input stream passed to the compiler by using the CMD statement. The CMD statement can be used to add compiler directives, force titles, or control listings based on the commands supported by the compiler in the input stream.

You can append translator options to the options specified in the language definition by using the PARM statement. Use the statement as many times as necessary to specify all options you want (up to a string length of 512 characters).

You can pass parameters directly to specific build translators defined in the language definition by using the PARM*x* statement, coupled with the use of the PARMKWD parameter of the FLMTRNSL macro.

SCLM orders compiles to ensure that outputs (such as DB2 DBRMs) are produced before compiling the member that references them. SCLM orders compiles that are within the scope of the build. (See "Build (option 4)" on page 237 for more information.)

SCLM generates compiler listings to temporary listing data sets so that you can view them online during the build. You can save these listings to members in the database by using the optional LIST architecture

member statement. If you do not specify the LIST statement, the online compiler listings are not saved. You might choose to discard the listings because of disk space limitations. Another option is to compress the listings. The ISPF sample library ISP.SISPSAMP contains two sample members, FLM03ASM and FLM03LMC, that demonstrate how to compress SCLM listings using the PACK option on the LMCOPY service. Comments explaining how to use these members are included in the code.

Specifying source members

Specifying a compilable source member to the build function is the alternate method of creating object modules. The language definition of the source member from the project definition determines which translators are called and where outputs are saved during the build. Compiler parameters can only be overridden by creating a CC architecture member.

Defining link-edit processed components

Standard linkage editors produce load modules as output. To define software components with load module outputs from standard linkage editors, use Linkedit Control (LEC) architecture members. LEC architecture members contain all the information necessary to produce a complete load module. Use the LEC architecture member to identify the following information:

- The load module name and the type in which you want it saved
- The linkage editor listing name and the type in which you want it saved
- All object and other load modules the load module is to contain
- Linkedit control statements and linkage editor options.

LEC architecture members must have at least one LINK, INCL, INCLD, or SINC statement and one LOAD statement.

Linkedit Control (LEC) architecture members can be constructed by referencing any combination of source members, CC architecture members, generic architecture members or LEC architecture members. Inputs to LEC architecture members are identified in the same way that inputs to CC architecture members are identified. The one difference is that by default LEC architecture members include object and load modules generated by the OBJ and LOAD statements in the input stream to the linkage editor. SINC statements can be used in LEC architecture members to identify object modules or load modules which are generated outside of the project. If SINC statements are being used to include load modules, the input ddname for the build translator must specify KEYREF=INCL. One additional statement can be used in LEC architecture members to identify an input to the linkage editor. That statement is the LINK statement. It identifies an output in the project that does not need to be rebuilt before being included in the input stream.

SCLM verifies that the inputs to the LEC architecture member are up-to-date before link-editing the inputs. SCLM will rebuild any inputs that are outputs of building other members in the project when those outputs are out-of-date. The inputs specified on LINK statements are an exception. These inputs will not be rebuilt.

You can override default linkage editor options by using the PARM statement. Use the statement as many times as necessary to specify all options you want. SCLM uses the standard S/370 linkage editor as defined by the LE370 language definition unless an LKED statement is used to override the default. See LKED within “Statement uses” on page 278 for more information.

You can specify in the LEC that SCLM pass linkage edit control statements directly to the linkage editor by using the CMD statement. Insert the control statements along with the object and load modules by careful positioning in the LEC architecture member.

The CMD statement can be used to include object modules and load modules that are in data sets outside of the project. The language definition for the linkage editor must include a ddname referencing the data set containing the members to include.

SCLM generates linkage editor listings to temporary listing data sets so that you can view them online during the build. You can save these listings to members in the database by using the optional LMAP

architecture member statement. If you do not specify the LMAP statement, the online linkage editor listings are not saved. You might choose to discard the listings because of disk space limitations. Another option is to compress the listings. The ISPF sample library ISP.SISPSAMP contains two sample members, FLM03ASM and FLM03LMC, that demonstrate how to compress SCLM listings using the PACK option on the LMCOPY service. Comments explaining how to use these members are included in the code.

You cannot use the SETSSI linkage editor command in an LEC architecture member. If SCLM finds a CMD SETSSI statement in an LEC architecture member during a build, the build function overrides the statement with its own SETSSI command.

SCLM build and control timestamps

SCLM uses the System Status Index (SSI) field to signify that the last update of a load module was made through SCLM. The SSI field data that SCLM generates consists of the following bits: the most significant bit is defined as a flag; the next most significant 11 bits specify hour and minute in binary form; and the least significant 20 bits specify Julian date in packed decimal form. SCLM sets the flag bit and writes these items into the SSI field during build processing when it generates a load module.

Table 21. SCLM System Status Index Field Data

Bit	Definition	Form
0	flag	bit
1-5	hour	binary
6-11	minute	binary
12-31	Julian date	packed decimal

Note: To update the SSI information for a PDSE, SCLM uses the IEWBIND macro. The IEWBIND macro provides the invoking program with access to binder services. Not-editable PM2 and higher format program objects cannot be reprocessed by the binder. The advantage of PM2 or higher format program object created with the not-editable option is that they may require much less space on DASD.

In some cases SCLM Build fails when EDIT=NO is specified. For example, the output from the C or C++ compiler is an extended object file (XOBJ). The lowest compatibility of the binder with XOBJ format input without going through the Language Environment® prelinker is PM3. In addition there are some restrictions on the usage of the not-editable attribute for PM2 or higher format program objects. Therefore we recommend not to use EDIT=NO binder option with SCLM-controlled members. See *z/OS MVS Program Management: Advanced Facilities* and *z/OS MVS Program Management: User's Guide and Reference* for more details.

Defining application and subapplication components

You can define applications and subapplications by using High-Level (HL) architecture members. HL architecture members allow you to categorize groups of related load modules, object modules, and other software.

You can maintain one HL architecture member to define an entire application for a project. This HL architecture member references other architecture members that eventually reference every component in the application. It can also reference the source directly, with the language of the source defining the outputs to be produced. By using this HL architecture definition as input to the build or Promote functions you can ensure that the entire application is up to date or is promoted to the next group in the project hierarchy. A build or promote of an HL architecture member results in the building or promotion of every software component referenced. In this way, you can guarantee the integrity of an entire application.

You can also use an HL architecture member to define subapplication software components. Subapplications can be a combination of load modules or merely a list of internal data items to be

controlled. Subapplications can, in turn, reference other subapplications to any depth. Conscientious use of HL architecture members contributes to application modularity.

SCLM can control and track ISPF panels, skeletons, and messages that are not processed by a compiler or linkage editor or used to invoke processors. Because these unique forms of software are not processed by compilers, linkage editors, or other processors, they are considered data dependencies and, therefore, can be controlled by using the PROM statement.

In most cases, you do not want panel, skeleton, and message dependencies in LEC, CC, and generic architecture members. Use HL architecture members to control all dialog software. For example, you can use one HL architecture member for panels, one for skeletons, one for messages, and one for the entire dialog that references the three previous HL architecture members.

The PROM statement date_check parameter allows SCLM to bypass date checking for the referenced member, thereby eliminating the need to build before promoting when that member is modified. Careful use of the PROM statement in this manner can eliminate unnecessary SCLM processing and improve efficiency.

Generic architecture members

Generic architecture members are used to process members that do not generate object modules. Examples of the outputs that might be produced are documentation and panels. Generic architecture members are almost the same as Compilation Control (CC) architecture members. The difference is that generic architecture members cannot generate object modules using the OBJ statement. If an OBJ statement is added to a Generic architecture member it becomes a CC architecture member. Other output statements such as LIST and OUT1 are used in generic architecture members to identify the listings, documentation, panels or other outputs produced.

Build and promote by change code

You can also use architecture definitions to identify the parts associated with a specific change or group of changes. This can be done in any architecture member using the CCODE statement. In addition to the normal contents of an architecture definition, such an architecture member contains a list of CCODE keywords followed by a change code and include flag. An example of such an architecture definition follows:

```
* ARCHDEF FOR PACKAGE PKG00001
CCODE  POY66045 INCLUDE   * Include changes for problem POY66045
CCODE  POY66615 INCL      * Include changes for problem POY66615
INCL   SCLM     ARCHDEF   * SCLM ARCHDEF
```

There are no SCLM-enforced conventions for change codes. The only restriction is that it be a maximum of 8 characters. For SCLM to determine the change code, any change code that contains an embedded blank or whose first character is other than A-Z, 0-9, @, # or $ must be enclosed in delimiters. A delimiter can be any character not specified above. Here are some examples:

```
CCODE A                 * this includes change code A
CCODE ,A B C,  E        * this excludes change code A B C
CCODE /AB/     IN       * this includes change code AB
CCODE 'A B'    EX       * this excludes change code A B
CCODE 1"       EXCLUDE  * this excludes change code 1"
```

Valid values for the include flag are INCLUDE or EXCLUDE. The default value is INCLUDE. A value of INCLUDE indicates that *only* the changes specified are included. A value of EXCLUDE indicates that everything *except* the specified changes are included. The following table illustrates the conditions under which SCLM will build and promote by change code.

Table 22. The conditions under which SLCM builds and promotes by change code

MEMBER CHANGE CODE	CCODE CCODEX INCLUDE	CCODE CCODEX EXCLUDE
CCODEX	Yes	No
CCODEY	No	Yes
no change code	No	Yes

Multiple CCODE statements can be specified in an architecture definition. An error message is issued when the include flag value is not the same on all statements. Duplicate CCODE statements are ignored. Any CCODE statements whose change code and include flag resolve to the same value are considered duplicates. For example, the following CCODE statements are duplicates:

```
CCODE 1
CCODE '1 ' INCLUDE
```

CCODE and COPY keywords cannot be used in the same architecture definition. Because the COPY keyword causes an actual copy of an architecture definition to be inserted into the first, the architecture definition referenced by the COPY statement must also be free of CCODE statements. To build an architecture definition containing COPY statements by change code, create a new architecture definition that contains the CCODE statement and an include (INCL) of the original architecture definition.

The concept of a package (group of changes) is supported through the ability to specify multiple CCODE keywords in an architecture definition. To more easily identify and maintain these architecture definitions, you can define a TYPE called PACKAGE with a language of ARCHDEF and use the package identifier or change code as the name for each member name. Or you can define a single architecture member and update the change code values in that member for each new build or promote by change code.

Only those CCODE statements that appear in the architecture definition specified as input to the build or promote will be processed. All other CCODE statements will be ignored. For example, assume that you have architecture definitions ISPF, PDF, SCLM and ISPFSUB. The architecture definitions contain the following statements:

```
* ARCHITECTURE DEFINITION MEMBER ISPF
INCL  ISPFSUB ARCHDEF
INCL  PDF     ARCHDEF
INCL  SCLM    ARCHDEF
CCODE A       INCLUDE
```

```
* ARCHITECTURE DEFINITION MEMBER ISPFSUB
CCODE D       INCLUDE
```

```
* ARCHITECTURE DEFINITION MEMBER PDF
CCODE B       INCLUDE
```

```
* ARCHITECTURE DEFINITION MEMBER SCLM
CCODE C       INCLUDE
```

When the ISPF architecture definition is built, only members with the change code A will be included from the build group. The CCODE statements to include change codes B, C, and D will not be processed for this build because they were found in included architecture definitions.

During the verification phase of build and promote, SCLM will search the change code list for members in the build or promote scope at the specified group. If the member is in scope and the change code appears (or does not appear in the case where EXCLUDE is specified) on the change code list, it will be included. Otherwise, SCLM will continue to search for the member beginning at the next group. Change codes will be processed for all editable members stored in PDS data sets under SCLM control, including architecture definitions. Change codes will be processed on included members when their data sets are allocated with IOTYPE=I, KEYREF=SINC. Included members whose data sets are allocated with a KEYREF of SREF or

CREF will not be processed by change code. To process includes referenced by SREF and CREF allocations:

1. Add FLMINCLS macros to reference the desired types.
2. Change the FLMALLOC macros to use KEYREF=SINC.
3. Add an INCLS parameter to the FLMALLOC macros to reference the FLMINCLS macros.

The architecture definition specified as input to the build or promote will always be processed, regardless of its change codes. Change codes are only significant for the build or promote group. In scope members found above this group will be included regardless of change code. If the specified change appears on a member's change code list but is not the last change and INCLUDE is specified, a warning message will be issued.

We recommend you build and promote each change to a member before beginning another. In cases where this is not possible, multiple changes that affect a single member should be built or promoted together. For instance, assume that you have members A, B, and C. Change 1 affects members A and B while change 2 affects members A and C. As both changes affect member A, the inclusion of either change without the other will cause the changes to be unsynchronized. Change codes 1 and 2 should be built and promoted together.

To build an application containing dynamic includes by change code, a build without change codes must occur first. Otherwise, the build can fail because includes are missing.

A promote by change code must always be preceded by a successful build of the same architecture definition. At the completion of a promote by change code, rebuild the application at the higher group. Change codes are used to determine whether a member found at the report input group will be included in the Architecture Report when executing the Architecture Report Utility against an architecture definition containing CCODE statements. The Database Contents Utility, on the other hand, does not use change codes specified on CCODE statements to determine whether a member will appear in the report or tailored output.

Architecture statements

You must use a special SCLM architecture language when you create architecture members. This language consists of statements that identify necessary information. The following paragraphs discuss the statements and their formats.

Statement format

You must use a specific format for architecture members. Architecture definition data sets must be fixed block (FB) with a length of 80 bytes or characters. Only one statement can appear in each 80-byte record. A record ranges from columns 1 through 72, and the records cannot be continued. SCLM ignores information that appears after column 72.

Write the statements in either upper- or lowercase. You can write all statements, except for CMD, PARM, and PARMx statements, in a free format as long as the items within the statements are in the correct order. The number of blank spaces between each item is not significant (except in the CMD statement).

The order of statements is generally not significant. For example, you can place OBJ statements before or after SINC statements. The only statements for which the order is significant are those keywords that cause data to be concatenated into the input stream (INCL, INCLD, CMD and LINK for LEC architecture members; SINC and CMD for CC and generic architecture members); or into the translator options (PARM and PARMx).

Member and type names must follow MVS naming conventions. SCLM does not check parameters and control statements for validity. They can continue up to and including column 72.

All members explicitly referenced by an architecture statement MUST exist in the type specified in the architecture statement. However, SCLM uses extended types and include sets to resolve the parsed dependencies of members referenced by a SINC statement if necessary.

Statement uses

SCLM distinguishes architecture members from one another by their content. SCLM assumes, for example, that a member containing both an OBJ statement and a SINC statement is a CC architecture member, and that a member containing a LOAD statement is an LEC architecture member.

Architecture statements provide information about the design of applications in the project database. Here are the valid statements for each type of member.

HL

```
*  CCODE  COPY  INCL  INCLD  PROM
```

LEC

```
*     ALIAS  CCODE  CMD   COPY  INCL[2]  INCLD[2]  KREF  LINK[2]
LKED  LMAP   LOAD[1]  OUTx  PARM  PARMx  PROM    SINC[2]
```

CC

```
*     CCODE  CMD   COPY  INCL   INCLD  KREF   LINK  LIST
LKED  OBJ[1]   OUTx  PARM  PARMx  PROM   SINC[1]  SREF
```

Generic

```
*     CCODE  CMD    COPY  INCL   INCLD  KREF   LINK
LIST  LKED   OUTx   PARM  PARMx  PROM   SINC[1]  SREF
```

Footnotes:

1. Each of the following statements must be present in the architecture definition member:
   - An LEC member must contain exactly one LOAD statement
   - A CC member must contain exactly one OBJ statement and at least one SINC statement
   - A Generic member must contain at least one SINC statement.
2. An LEC member must contain at least one of the following statements: INCL, INCLD, LINK, or SINC.

Figure 147. Valid keywords for architecture member statements

Each architecture statement is composed of a keyword followed by one or more operands. For those keywords that allow you to specify either a member name or an asterisk (*), specify an asterisk if you expect multiple outputs per DD statement. Otherwise, specify the member name if only a single output is expected. The following list shows the valid statements, their usage, and their format:

*****

Identifies an architecture comment statement on a line by itself.

```
* <comment>
```

ALIAS

Identifies load module aliases to be generated. Use it only in LEC architecture members. The type_name specified on the ALIAS statement must be the same as the type_name on the LOAD statement of the LEC architecture member.

```
ALIAS <member_name> <type_name> <optional_comment>
```

CCODE

Identifies a change code to be included or excluded from a build or promote.

Any change code that contains an embedded blank or whose first character is other than A-Z, 0-9, @, # or $ must be enclosed in delimiters. A delimiter can be any character not specified before.

Valid values for the include flag are INCLUDE and EXCLUDE. The flag can be abbreviated but must be followed by a space. If no value is specified, the default is INCLUDE. Examples of valid flags are I, E, IN, EX, INCL, and EXCL.

```
CCODE change_code <optional_include_flag> <optional_comment>
```

CMD

Identifies command statements to be included with inputs to the compiler, linkage editor, or other processors. SCLM strips off the CMD keyword and the first blank of this statement, then passes the remaining columns (4-80) as columns 1-77 directly to the processor's input stream. No further formatting, substitution, or interpretation is performed on the statement. Thus, when coding a CMD statement you must consider how the input statements will be interpreted by the invoked processor.

For example, the linkage editor expects at least one blank at the beginning of a statement (but not more than 15) before the operation code. The linkage editor also expects at least one blank between that and the operand and everything following the first blank after an operand is a comment. The exception is column 72, which is the statement continuation character. Therefore, CMD statements coded in an LE ARCHDEF must have at least 2 (but no more than 16) blanks between the CMD keyword and the operation code, and that column 75 must be blank, unless a continuation to the next CMD statement is desired.

Do not include the optional_comment with the CMD statement because it will be part of the control statement.The CMD statement is not valid in HL architectural members.

```
CMD <control_statement>
CMD PARMS /Ss /DIPF
CMD ACTION IPFCP
```

The FLMLTWST translator reads the build map for ACTION and PARMS control statements. ACTION may be used for additional workstation commands. PARMS may be used to identify strings to be added to the workstation command. These control statements are different than the ACTION and PARMS keywords that may be used in the OPTIONS list for FLMLTWST. The PARMS value in the OPTIONS list is added to all workstation commands whereas the string following the PARMS control statement in the build map is appended to the workstation command being created at that time. See “FLMLTWST Workstation Build translator” on page 608 for more information.

Note: CMD statements in an architecture definition will be placed in the build map with the control statement. The control statement will only be passed to the build translator in the controlling language definition if there is also an FLMALLOC macro with IOTYPE=S. Translators used for workstation build may read the control statement from the build map to create a workstation command.

COPY

Identifies another architecture member to be inserted into this architecture member.

The COPY statement of the architecture language provides you with the ability to simplify related, complex architecture members. To simplify architecture members with similar contents, use the COPY statement to isolate identical statements into a separate member and reference the member. Referenced members must follow all formatting rules for architecture members.

The COPY statement results in a direct insert of the contents of the specified member into the respective architecture members. Therefore, using a copy architecture member is an efficient way to group sets of commonly used architecture statements into a single area. Additions to and deletions from the common architecture member affect all the architecture members referencing the member.

```
COPY <member_name> <type_name> <optional_comment>
```

Note: Use the COPY statement rather than the INCL statement (see the following description) when the specified member cannot be processed independently from the architecture definition in which it appears.

INCL
Identifies another architecture member that this architecture member references. The referenced architecture member will be processed before this architecture member.

Additionally, if INCL is used in an LEC architecture member, the output from the INCL is used to create the load module for the LEC.

Only CC and LEC architecture members should be referenced by an INCL statement in another LEC architecture member. For CC architecture members, the output referenced by the OBJ keyword is used to create the load module; for LEC architecture members, the output referenced by the LOAD is used.

```
INCL <member_name> <type_name> <optional_comment>
```

Note: Use the INCL statement rather than the COPY statement (see the previous description) when the specified member can be processed independently from the architecture definition in which it appears.

INCLD
Identifies a source member that this architecture member references. The referenced member will be processed before this architecture member.

Additionally, if INCLD is used in an LEC architecture member, the output from the INCLD is used to create the load module for the LEC. The language definition for the member referenced by the INCLD statement must have a build output with KEYREF=OBJ.

```
INCLD <member_name> <type_name> <optional_comment>
```

KREF
Identifies the output keywords from other members that will become inputs to the member containing the KREF statement. The keywords identified by the KREF statement must be architecture statements that identify outputs of a build. Examples are OBJ, LOAD and OUT1. Only those outputs of members referenced by INCL or INCLD statements in the architecture member containing the KREF statement will be considered for inclusion.

If the KREF statement is omitted, the outputs that are included depend on the type of architecture definition. For LEC architecture definitions, the default is to include OBJ and LOAD outputs. For all other types of architecture definitions, the default is not to include any outputs produced by referenced members.

If a KREF statement is specified in an LEC architecture definition, the defaults of OBJ and LOAD will be lost. To include another output type in addition to OBJ and LOAD, three KREF statements must be specified: one for OBJ, one for LOAD, and one for the additional output type (OUT1 for example).

Valid reference keywords are: COMP, LIST, LMAP, LOAD, OBJ, and OUTx.

```
KREF <reference_keyword>
```

Note: Although multiple KREF statements can be coded in a single LEC architecture member, duplicate KREF statements will result in an error.

LINK
Identifies an output that must be produced before this ARCHDEF is processed. The build function only verifies the contents of the output referenced if extended scope is specified. You can substitute the INCL statement to cause this verification to always be performed.

Additionally, if LINK is used in an LEC architecture member, the output referenced is used to create the load module for the LEC.

```
LINK <member_name> <type_name> <optional_comment>
```

LIST
Identifies the members and type in which the compiler listing is to reside. The LIST statement is not valid in HL or LEC architecture members.

```
LIST <member_name | *> <type_name> <optional_comment>
```

LKED
Identifies the language to be used to process the contents of the architecture member.

Language_id is an 8-character language identifier for a translator. The language ID specified must correspond to a valid language identifier defined in the project definition.

If the LKED keyword is omitted, SCLM uses the default language to process the architecture member. For LEC architecture members the default language is LE370. For CC and Generic architecture members the default language is the language of the member on the first SINC statement.

```
LKED <language_id> <optional_comment>
```

LMAP
Identifies the members and type in which the linkage editor listing (load map) is to reside. Use it only in LEC architecture members.

```
LMAP <member_name | *> <type_name> <optional_comment>
```

LOAD
Identifies the load modules to be created and the type in which the load modules reside. Use it only in LEC architecture members.

```
LOAD <member_name> <type_name> <optional_comment>
```

OBJ
Identifies the name of the object modules to be created and the type in which the modules reside. Use it only in CC architecture members.

```
OBJ <member_name | *> <type_name> <optional_comment>
```

OUT*x*
Identifies the output members to be created and the type in which the members reside. Replace the *x* with an integer to identify the specific statement. Valid integer replacements are 0 through 9. You can use these statements to track additional outputs other than the standard outputs described by the statements OBJ, COMP, LIST, LOAD, and LMAP. Use the OUT*x* statement in an LEC, CC, or generic architecture member.

```
OUTx <member_name | *> <type_name> <optional_comment>
```

PARM
Identifies parameters (options) to be passed to all build translators of a compiler, linkage editor, or other processor. Use it in generic, CC, or LEC architecture members. Do not use this keyword to pass parameters to non-build translators such as VERIFY, PURGE, and COPY.

SCLM offers a set of variables that you can use to dynamically provide information to compilers, linkage editors, and other processors. Use these variables with the PARM statement.

Do not use the optional_comment with the PARM statement because it will be passed to the build translators.

```
PARM <parameters>
```

PARM*x*
Identifies parameters (options) to be passed to build translators of an SCLM language. Replace the *x* with an integer to identify the specific statement. Valid integer replacements are 0 through 9. You can use the SCLM variables, mentioned previously, with the PARM*x* statement. You can use the PARM*x*

statement in generic, CC, and LEC architecture members. Do not use this keyword to pass parameters to non-build translators such as VERIFY, PURGE, and COPY.

Do not use the optional_comment with the PARM*x* statement because it will be passed to the build translators.

If the PARM*x* keyword used in the architecture member is not specified in one of the FLMTRNSL macros (using the PARMKWD parameter), SCLM ignores the PARM*x* statement.

```
PARMx <parameters>
```

Note:

1. The complete options list passed to the build translator has a maximum length of 512 characters and has the following format:

   ```
    string1
   ,string2
   ,string3
   ```

   where

   string1
   contains the options from the OPTIONS parameter on the FLMTRNSL macro.

   string2
   contains the options from the PARM statements in the architecture definition. No commas are inserted between PARM statements.

   string3
   contains the options from the PARMx statements in the architecture definition. Commas are inserted between PARM*x* statements.

   Leading and trailing blanks are removed by SCLM.

   For example, suppose that the FLMTRNSL macro specifies that the following options are to be passed to a translator:

   ```
   OPTIONS=(NOXREF)
   ```

   Further suppose that there is an architecture definition for the translator with the following parameters defined:

   ```
   PARM  PARAMETER1
   PARM  PARAMETER2
   PARM  PARAMETER3
   PARM1 PARAMETER4
   PARM2 PARAMETER5
   PARM3 PARAMETER6
   ```

   The options passed to the translator would look like this:

   ```
   NOXREF,PARAMETER1PARAMETER2PARAMETER3,PARAMETER4,PARAMETER5,PARAMETER6
   ```

2. Parameters specified on the PARM and PARMx statements in an LEC architecture member are passed to the linkage edit translator but not to any of the compilations needed to produce object or load modules for the linkage edit operation.
3. You should review the documentation of each build translator for unique handling requirements of passed parameters (for example, case and handling of special characters).

PROM
Identifies a text member, such as design, data, or test plans, to be promoted along with the modules processed in this architecture member. The member specified is not processed by build (for example, compiled or linked) but is tracked during promotions. You can specify an additional parameter to indicate whether date checking is to be performed for the member.

Date_check is a special optional parameter for the PROM statement to bypass date checking for noncompilable/nonlinkable members. A nonblank character, such as N, as a third parameter on the PROM statement indicates to the build and promote functions to bypass date checking for that member (thereby eliminating the need to build before promoting) when you modify the member.

Note: Do not use the optional_comment with the PROM statement because it can cause build and promote to bypass date checking.

```
PROM <member_name> <type_name> <date_check>
```

SINC

When used in generic and CC architecture members, the SINC statement identifies the source member. When used in an LEC architecture member, the SINC statement identifies the member or group of members to pass to the linkage edit translator. Use it only in generic, CC, and LEC architecture members.

```
SINC <member_name> <type_name> <optional_comment>
```

You can specify multiple SINC statements in an architecture definition. SCLM copies each statement, in the order they appear, into the temporary file allocated with FLMALLOC IOTYPE=S.

Note:

1. The input list feature of the Build function is designed to work with direct translations of source members only (source members referenced with an INCLD statement). Using the input list feature with source members controlled by CC or Generic architecture definitions produces undefined results (source members referenced with a SINC statement). For more information about Input List languages and translators, see Part 1, “Project Manager's Guide,” on page 1.
2. If there is a SINC statement, but no FLMALLOC with IOTYPE=S, in the language definition for the language of the member referenced by the SINC statement, the referenced member is not placed on the SYSIN input stream for the build.

SREF

Identifies a type to be allocated during processing. Specifically, use the SREF keyword to allocate a specific type for translators. You can use it in generic, CC, and LEC architecture members.

SREF is a function that identifies an additional type to be allocated during processing. Do not use this function unless you have extremely complex hierarchical concatenation needs.

```
SREF <type_name> <optional_comment>
```

Sample application using architecture definitions

The following application is composed of two subapplications. The first subapplication consists of four load modules and the second subapplication consists of two load modules, that are composed of a series of object modules. Load module FLM01LD1,FLM01LD2, and FLM01LD9 contain one object module each, while FLM01LD3,FLM01LD4,and FLM01LD7 contain multiple object modules. Figure 148 on page 284 shows a diagram of the design of this application (FLM01AP1) and Figure 149 on page 285 shows the architecture members for the FLM01AP1 application.

Note: SCLM tracks the included members; therefore, there is no need to mention FLM01EQU in the architecture definition.

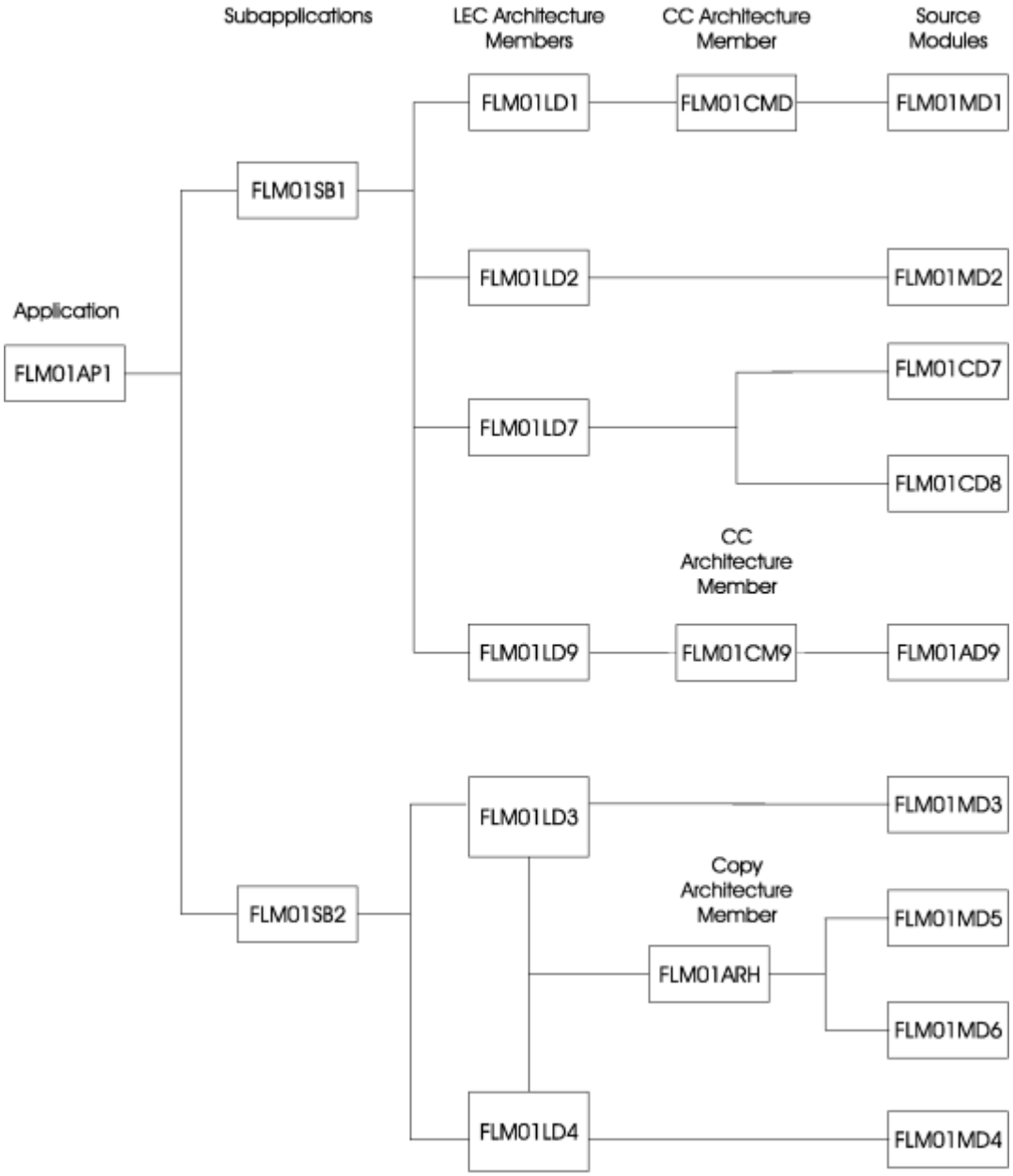


Figure 148. Application FLM01AP1

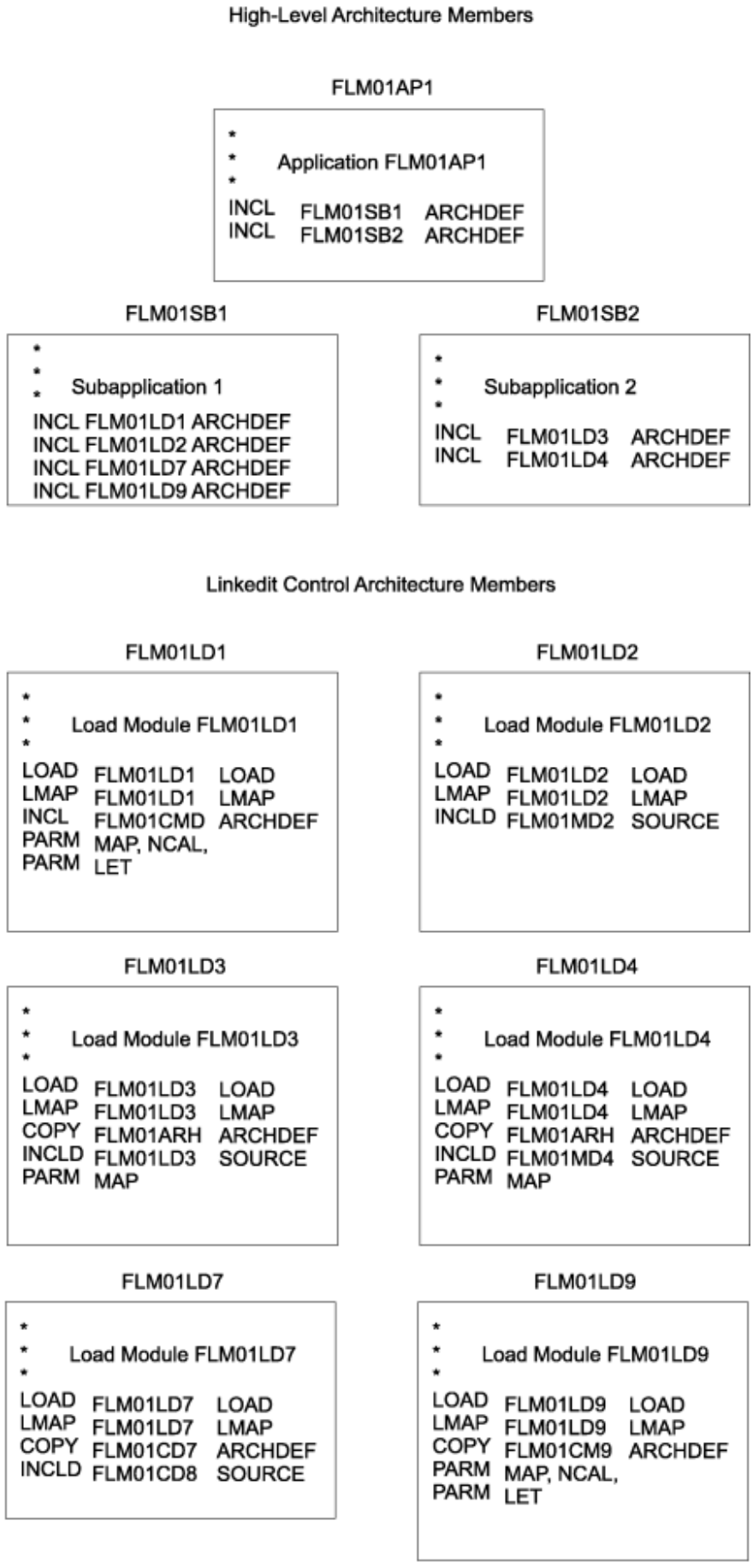


Figure 149. Architecture Members for Application Sample FLM01AP1 (Part 1 of 2)

Compilation Control Architecture Members

FLM01CMD

```
*
*      Object Module FLM01MD1
*
OBJ     FLM01MD1   OBJ
LIST    FLM01MD1   LIST
SINC    FLM01MD1   SOURCE
PARM    NOXREF, LC(75)
```

FLM01CM9
with Fault Analyzer

```
*
*      Object Module FLM01AD9
*
OBJ     FLM01AD9   OBJ
OUT2    FLM01AD9   IDILANGX
LIST    FLM01AD9   LIST
SINC    FLM01AD9   SOURCE
PARM1   FOLD,NORENT, FLAG (4)
```

FLM01CM9
without Fault Analyzer

```
*
*      Object Module FLM01AD9
*
OBJ     FLM01AD9   OBJ
LIST    FLM01AD9   LIST
SINC    FLM01AD9   SOURCE
PARM1   FOLD,NORENT, FLAG (4)
```

Copy Architecture Members

FLM01ARH

```
*
*      COPY ARCHITECTURE
*
INCLD   FLM01MD5   SOURCE
INCLD   FLM01MD6   SOURCE
```

Figure 150. Architecture Members for Application Sample FLM01AP1 (Part 2 of 2)

The HL architecture member in part 1 of this figure includes references to two subapplications: (FLM01SB1 and FLM01SB2). The subapplication HL architecture members reference the LEC architecture members that define the load modules they contain. Note that the referenced LEC architecture members have the same names as the load modules they produce.

The LEC architecture members contain all the information necessary to produce the load modules in the application. Two PARM statements in FLM01LD1 override the default linkage editor options.

Load modules FLM01LD3 and FLM01LD4 contain copy statements. These statements identify the architecture member FLM01ARH, that references two source modules for SCLM to insert into the FLM01LD3 and FLM01LD4 load modules.

Thus, copy architecture members are an efficient technique for grouping commonly used architecture statements into a single member. Additions to and deletions from FLM01ARH affect FLM01LD3 and FLM01LD4 and all the other architecture members that might reference FLM01ARH.

Ensuring synchronization with architecture definitions

SCLM ensures that all modules within the scope of a build are synchronized. If you build a source module, SCLM synchronizes the resulting object and listing with the source. If you build an architecture definition, SCLM synchronizes all members used as input to the builds and all members output from the builds. However, if there are object or load modules outside the scope of a particular build that are dependent on source modules within the scope of that build, those source, object, and load modules might no longer be synchronized.

In the following example, object modules OBJ1, OBJ2, and OBJ3 are produced by compiling source modules SOURCE1, SOURCE2, and SOURCE3, respectively. SOURCE2 might be the source module for an I/O routine many applications use. Load module LOAD1 is the result of linking OBJ1 and OBJ2, while LOAD2 results from the link-edit of OBJ2 and OBJ3. LOAD1 and LOAD2 might be two separate programs that run against the same kind of data and would therefore need to have a common I/O routine (SOURCE2). FLM01AP1 and FLM01AP2 are LEC architecture definitions that describe how to link-edit LOAD1 and LOAD2, respectively. Finally, TOPARCH is a high-level architecture definition that includes FLM01AP1 and FLM01AP2.

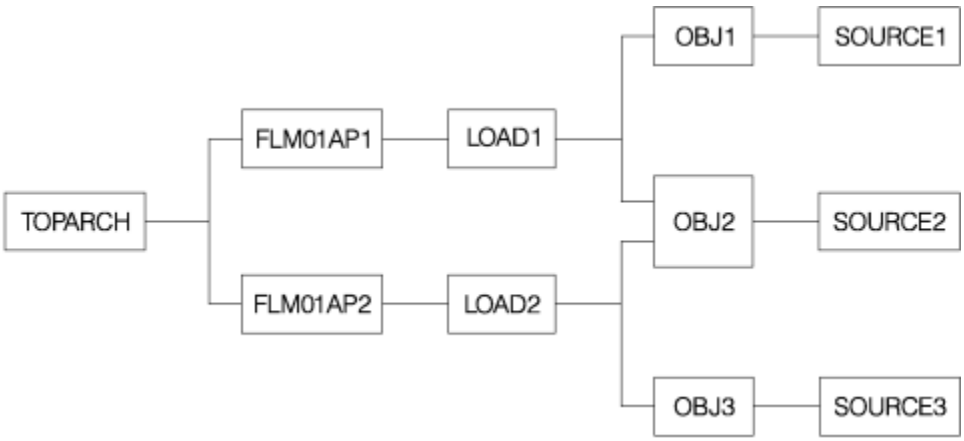


Figure 151. Example of Synchronization

In Figure 151 on page 287, all of the modules shown in the diagram exist only in the production layer of your SCLM-controlled hierarchy and all source, object and load modules are synchronized. In other words, for each load module, the hierarchy contains the exact version of the object modules that were used to link-edit that load module. For each object module, the hierarchy contains the exact version of the source that was compiled to create that object module. You can always recreate exactly (except for time stamps) the object and load modules for the applications.

With this structure, you must pay close attention to which architecture definitions you use to build and promote development changes. The following scenario describes the INCORRECT use of architecture definitions, which leads to a loss of synchronization between source and load.

A user puts in a request for a change to LOAD1 and you decide that the way to implement that change is to modify SOURCE2. Because you are making a change to LOAD1, you also decide (in error as it will turn out) to use FLM01AP1 to drive your builds and promotes. When your changes are made and you are ready to build, you cause SCLM to rebuild OBJ2 (because SOURCE2 changed) and LOAD1 (because OBJ2 changed), by specifying FLM01AP1 on the Build panel. LOAD2 is *not* rebuilt, even though OBJ2 changed, because LOAD2 is outside of the scope of architecture definition FLM01AP1. Herein lies the problem. When you promote FLM01AP1, SCLM checks that everything that needs to be rebuilt (within the scope of FLM01AP1) has been rebuilt. Unfortunately, modules outside the scope of FLM01AP1 should be rebuilt as well.

When complete, all modules within the scope of FLM01AP1 are synchronized and recreatable. However, LOAD2 was outside the scope of the architecture definition you used and is not recreatable. Therefore LOAD2 is not synchronized with its source.

To avoid this problem, you must analyze the architecture of the applications in your SCLM-controlled project and choose an architecture definition with a scope that contains all modules that need to be rebuilt. The correct architecture definition would have been TOPARCH in the example because only TOPARCH has both LOAD1 and LOAD2 within its scope. These modules have to be relinked because of a change to SOURCE2.

It is strongly suggested that you have one high-level architecture definition with a scope that includes every module controlled by an SCLM project. You can use architecture definitions with much smaller scopes in your day-to-day development work. However, if you do that, you should also check the synchronization of all modules in the project by performing a build on the top high-level architecture definition as part of your testing.

Build outputs

Several architecture definition statements are used to identify the outputs of a build. These statements are: ALIAS, COMP, LIST, LMAP, LOAD, OBJ, and OUTx. These statements have two parameters. The first is the member name of the output and the second is the type name of the output. The type name parameter must be a type name from the project definition. The member name parameter can be either a valid PDS member name or an "*". A PDS member name can be used when there is a single output with a predefined member name. PDS member names must be used for the ALIAS and LOAD architecture statements. An "*" must be used if there are multiple outputs or the output member name is not predefined.

Build allocates temporary data sets to hold the outputs generated by the build translators. If all the translators complete successfully the outputs from the temporary data sets are copied into the SCLM hierarchy. Because the copy does not take place until all translators have completed, the allocation of the output data sets must be retained without overwriting the output until after the last translator runs.

Multiple build outputs

Multiple output members may be generated for a single output keyword if the IOTYPE on the FLMALLOC for the translator output is "P". This allows the translator to store multiple members into a PDS data set. When a PDS member name is specified on the output architecture statement SCLM will copy a member with that name from the temporary data set into the SCLM hierarchy. The member name in the temporary data set must match the SCLM member name. When an "*" is specified in the member name parameter then SCLM will copy all outputs in the temporary data sets without changing the member names.

Sequential build outputs

A single build output may be generated into a sequential data set by using an FLMALLOC with IOTYPE=O. When the output architecture statement indicates a member name the output will be copied to an SCLM member of that name. When an "*" is specified for the output member the member name will be the name of the architecture definition.

Default output member names

When a source member is built directly, either as the input member to the build or by an INCLD statement, the output member name is determined from information in the project definition or by SCLM defaults. If the FLMALLOC statement for the output specifies a default member name using the DFLTMEM parameter then that member name will be used. When no default member name is specified, the output member name will be the same as the source member. Use an architecture definition when generating multiple outputs to be stored in a partitioned data set. See the description of “Multiple build outputs” on page 288.

Languages of output members

SCLM gets the language of the output member from one of two locations. The first place SCLM looks is on the FLMALLOC statement in the project definition for a LANG parameter. If it is found then it is used as the language of the output member. When no LANG parameter is found and a source member is being built the language of the source member is used as the language of the output member. If an architecture

definition is being built and no LANG parameter was found, then the language used to build the architecture definition is used as the language of the output member.

Part 3. Advanced Topics

© Copyright IBM Corp. 1990, 2019

Chapter 12. Managing complex projects

This chapter describes additional SCLM features that you can use to define and manage complex projects. Topics discussed in this chapter include:

- Impact assessment techniques
- Dependency processing implementation
- Propagating applications to other databases.

Impact assessment techniques

Making updates to a component of an application without full knowledge of their effect on the application can cause a large number of unexpected recompilations. Impact assessment is a technique you can use to assess the impacts of updates to an application *before* they occur. It allows developers to determine what effect changing a given component of the application has on the rest of the application or a given subapplication. Impact assessment enables you to avoid time-consuming recompilations.

Follow the procedure below to use SCLM Build to create an impact assessment:

1. Use the SCLM editor to save the members you want to change
   a. in an empty development group or
   b. save them with a change code.
2. Invoke the build function using the report mode on the top architecture definition for the application affected. If you saved with a change code, create a new top architecture definition that includes the old top architecture definition and uses the CCODE keyword to include the change.
3. Examine the resulting build report. This report reflects all output that regenerates when the build is performed. The build messages data set indicates which translators are invoked.
4. If the results are acceptable, you can proceed with your planned changes. Otherwise delete the members you saved in Step 1 using the SCLM Library utility or the Delete from Group utility.

You can perform a second method of assessing impacts by using an SCLM architecture report. Examine this report for the members that the developer wants to modify. Starting with the members to be modified, you can identify all architecture members that control the modified members. While this technique is more meticulous than the first, it does not require that the member be drawn down, modified, and built.

Either of the preceding techniques help identify costly recompilation impacts.

Dependency processing

This section explains how SCLM handles include dependencies. If SCLM does not provide a sample for a language you want to support, use this information to map the language dependencies to SCLM dependencies.

SCLM derives dependency information when a member is parsed. This information is stored as SCLM control data, and it allows SCLM to perform the following functions:

- Process members in the correct order
- Determine when members are out-of-date (changed) and need to be rebuilt
- Determine the scope for functions such as build and promote.

The following describes the processing involved for each include dependency.

© Copyright IBM Corp. 1990, 2019

A member is included if it is required for completion of a compile of the member that references it. Examples are members referenced by the %INCLUDE directive in Pascal, the COPY operand in Assembler, the COPY command in COBOL, and the imbed (.im) in Script. Assembler macros are also considered to be includes because they must be expanded when the referencing member is assembled.

The primary input to the compiler defines the SCLM-controlled data sets to search for includes. The primary input to the compiler is referenced directly on the build panel or via the SINC or INCLD architecture definition keywords in SCLM. If more than one SINC keyword is used in an architecture definition, the primary input is the member referenced by the first SINC.

Any member can have include dependencies. SCLM recursively searches for included members beginning with the primary input to find all of the dependencies that are needed for the compilation.

The language of the primary input defines which types are searched to find includes. The FLMINCLS macro is used to specify which types are searched and the order in which they are searched. For more information about how includes are found, see Part 1, “Project Manager's Guide,” on page 1.

Included members can be editable or non-editable.

Included members must exist and have valid accounting information when the member that references them is built. Build does not attempt to compile members that have missing include dependencies.

Build rebuilds the primary input member if any of its recursive includes have changed since it was last built.

Propagating applications to other databases

You can use EXPORT or IMPORT to propagate systems by moving code from a development group to a production group.

You can also use the EXPORT and IMPORT utilities to back up and restore data from an SCLM hierarchy. The steps necessary to back up and restore the project database are listed as follows:

1. Export the group to be backed up using the EXPORT service.
2. Save the member text in a PDS for later recovery if necessary.
3. To restore the data, create an alternate definition that specifies a new temporary development group into which you will import the previously exported data.
4. Specify the export data sets to be restored on the FLMCNTRL macro.
5. Copy the saved member text for the backed up group to the new temporary group.
6. Invoke the IMPORT service and specify the new temporary group. Note that after the IMPORT service has completed, the new group contains the same data that was originally exported.
7. If you use the new group, use the DELGROUP service to purge the data in the original group, delete the original data sets, and rename the temporary group to the original group name. Another way of accomplishing the same goal is to delete the accounting data out of the original group and then import directly into it.

Note: The IMPORT service erases the exported data after it successfully imports members. Therefore, you may want to make a copy of the export data sets before invoking the IMPORT service if you want to preserve the backup version of the data sets.

Chapter 13. SCLM support for DB2

In SCLM, you can have applications that support DB2 processing. Before SCLM can correctly interact with DB2, the DB2 system must be installed and fully operational.

DB2 support in SCLM can be split into two areas:

- Creation of the DBRM (Database Request Module)
- Binding the package or plans

The creation of the DBRM is handled in the language translator you have used to compile your program. Generally there will be a DSNHPC step that interprets the SQL and creates the DBRM and passes the modified source through to the compiler. The DBRM type will be an SCLM defined type. This means that the build map for the source member will contain outputs for DBRM as well as OBJ and LIST.

The key to the SCLM processing is how to let SCLM know that a DB2 bind is required when a module containing DB2 statements is built. This is handled through the DB2CLIST and DB2OUT language translators, which are shipped with SCLM. They are discussed in more detail in “Generating a project environment” on page 296.

To create the link between the compilable program, the load module, and the plan or package, we must create a DB2CLIST for each bindable object. The term "DB2CLIST" is used for historical reasons but this member is now more likely to be written as a REXX exec than a CLIST. This DB2CLIST member specifies the names of the DBRMs that will be bound. For example, if your site uses plans, the DB2CLIST would contain entries for all the DBRMs in the plan. If your site uses packages, the DB2CLIST will just contain one DBRM entry for the package DBRM. These scenarios are discussed with examples in “Create DB2 CLIST” on page 302.

The DB2CLIST is a REXX exec or CLIST. It sets up any variables required for the bind and then either performs the bind or calls an external bind processor to take the parameters and perform the bind. Because the DB2CLIST is controlled by SCLM, it contains accounting information and can be built. The DB2CLIST can be referenced from architecture definitions. By using High Level (HL) architecture definitions you can link the LEC archdefs that control the compilation and link to the generic archdef that controls the bind. See “Getting started” on page 300 for some examples.

The processing of a DB2CLIST in SCLM has the following stages. Assume at this time that you have a program containing SQL and a LEC archdef to build that program:

1. During the Editing stage, you must create a DB2CLIST as described in “Create DB2 CLIST” on page 302. When parsed, the DBRMs to be bound are identified by the %INCLUDE statement in the comment block and an entry is placed in the accounting information for the DB2CLIST. SCLM will then know that when that particular DBRM is recreated during compilation, it will need to rebuild the DB2CLIST member.
2. During the BUILD stage, the DB2CLIST member is executed to perform the appropriate BIND or FREE DB2 operation. An identical copy of the DB2CLIST, called a DB2OUT, is created and placed in the type that is used during the PROMOTE stage. The DB2OUT is an SCLM output generated by the DB2CLIST translator and as such is non-editable.

   The only difference between the original DB2CLIST and the new DB2OUT is the language value. The language for the original DB2CLIST is associated with a language definition that contains the parsing and build translators. This language definition is DB2CLIST. It can be found in the SCLM sample library (member FLM@BD2). The language for the new DB2OUT is associated with a language definition that contains the copy and purge translators. This language definition is DB2OUT. It can be found in the SCLM sample library (member FLM@BDO).
3. On promote, SCLM will not execute the DB2CLIST but only promote it. However, on promote the COPY and PURGE phases of the translators are executed and as such will run the contents of the DB2OUT member. This is why the DB2 translator creates a DB2OUT. The DB2OUT will be processed in exactly the same way that BUILD processed the DB2CLIST. On the COPY phase the DB2OUT will be executed

© Copyright IBM Corp. 1990, 2019

with the BIND option and on the PURGE phase the DB2OUT will be executed with the FREE option if you wish to free the plan or package at the from-group level.

In your high-level architecture definitions, always refer to the DB2CLIST used during the Build stage. Do not refer to the DB2OUT used during the Promote stage.

Restrictions

The included members (for example, COPYBOOKs, INCLUDEs, DCLGENs) that are processed by the DB2 precompiler must reside in the SCLM source library or its extended library for SCLM to track them as included dependencies. Otherwise, the library should be added to the FLMSYSLB macro in the language definitions to prevent SCLM from creating an Include dependency. Additionally, ALCSYSLB=Y should be specified in the language definition for the compiler, or an FLMCPYLB with the appropriate library specified should be added into the FLMALLOC that has DDNAME=SYSLIB in the COBOL compiler step.

Some of the SCLM parsers will check for SQL includes. The parser determines the SQL include dependencies in the source program by parsing the EXEC SQL INCLUDE statements.

See Chapter 22, “SCLM translators,” on page 565 for more information.

Information for project administrators

See:

- “The FLMCSPDB DB2 bind/free translator” on page 296
- “Generating a project environment” on page 296

The FLMCSPDB DB2 bind/free translator

This translator is supplied with SCLM and is used to process the DB2CLIST and DB2OUT members. It processes the DB2CLIST at build time and DB2OUT at promote time. When processing the DB2CLIST it runs as a BUILD translator. When processing the DB2OUT it runs as a COPY translator or PURGE translator or both. When running at build time it also copies the DB2CLIST to the DB2OUT for later processing.

For more information on invocation including parameters and return codes, refer to Chapter 22, “SCLM translators,” on page 565.

Generating a project environment

Chapter 1, “Defining the project environment,” on page 3 describes the steps to set up and maintain an SCLM project database. For DB2 support, additional actions within these steps may need to be must be performed. This section describes these considerations step-by-step.

Step 1: Determine the project's hierarchy

There are no additional considerations.

Step 2: Identify the types of data to be supported

If you are already running an existing SCLM project that has all the data types described in Chapter 1, “Defining the project environment,” on page 3, additional types must be created. The following types of data must be maintained and are the recommended naming conventions:

DBRM
Contains the source member input to a DB2 bind. It is generated by the DB2 preprocessing step.

DB2CLIST
A DB2CLIST that contains editable source members. These source members are used during SCLM Build to control bind and free functions for DB2.

For example, if you wish to use a DB2CLIST to do package binds as well as plan binds then you may want to give your DB2CLIST types more meaningful names such as PKGCLIST and PLNCLIST. The output member would then need to be named PKGOUT and PLNOUT.

To have DB2CLIST members and DBRM members with the same name, an FLMINCLS macro needs to be specified in the language definition for the DB2CLIST members. The FLMINCLS macro must list the DBRM type first on the TYPES parameter. Here is an example of an FLMINCLS macro to do this:

```
*
* SPECIFY TYPES TO SEARCH FOR DBRMS THAT ARE TRACKED AS
* INCLUDES TO THE DB2 CLIST MEMBERS
*
         FLMINCLS TYPES=(DBRM)
```

DB2OUT

This type contains non-editable build output that is used during SCLM Promote to control BIND and FREE functions for DB2. During a build of a DB2CLIST, a copy of the DB2CLIST is copied to the type DB2OUT into the group that is being built. During a promote, this member is called to bind the plan or package in the TO group and free the plan or package in the FROM group. Since the DB2OUT is the "output" from the DB2CLIST, SCLM will move both objects together during a Promote.

Note: The DB2OUT type must come after the DBRM type in the EBCDIC collating sequence. This is so that during the promote the bind is started after the DBRM has been promoted. SCLM promotes in EBCDIC collating sequence order.

Step 3: Establish authorization codes

There are no additional considerations.

Step 4: Allocate the PROJDEFS data sets

There are no additional considerations.

Step 5: Allocate the project partitioned data sets

The data set characteristics for the new types are described in Table 23 on page 297.

Table 23. SCLM Data Set Attributes for DB2 Types

Type	PS or PO	RECFM	LRECL	BLKSIZE
DBRM	PO	FB	80	3120
DB2CLIST	PO	FB	80	3120
DB2OUT	PO	FB	80	3120

You can browse the example project definition, FLM@EXM2, which provides an example of the macros used to support DB2.

Step 6: Allocate and create the control data sets

There are no additional considerations.

Step 7: Protect the project environment

There are no additional considerations.

Step 8: Create the project definition

Specify additional types to be supported with the FLMTYPE macro.

SCLM provides many language definitions as examples. The examples serve as a guide in the construction of language definitions for specific applications and environments. Use the COPY macro to include any of the following sample definitions that apply to your DB2 environment:

Table 24. Language definitions for DB2

Member	Language	Description
FLM@BD2	DB2CLIST	DB2 BIND/FREE
FLM@BDO	DB2OUT	DB2 BIND/FREE output
FLM@2ASM	DB2ASM	DB2 preprocessing + Assembler
FLM@2CO2	DB2COB2	DB2 preprocessing + COBOL II
FLM@2C	DB2C370	DB2 preprocessing + C/370
FLM@2FRT	FORTDB2	DB2 preprocessing + FORTRAN
FLM@2COB	DB2COB	OS COBOL with DB2
FLM@2PLO	PLIDB2	PL/I OPTIMIZER with DB2
FLM@EASM	CIDB2ASM	ASSEMBLER F with CICS V3R2M1 and DB2
FLM@ECOB	CIDB2COB	OS COBOL with DB2 and CICS
FLM@ECO2	CIDB2CO2	COBOL II with DB2 and CICS
FLM@EC	CIDBC370	C/370 with DB2 and CICS
FLM@EPLO	CIDB2PLO	PL/I OPTIMIZER with DB2 and CICS
FLM@2CBE	COB3DB2	Enterprise COBOL with DB2
FLM@2CBF	COB3DB2F	Enterprise COBOL with DB2 and Fault Analyzer
FLM@2CCE	CB3C2	Enterprise COBOL with DB2 and CICS
FLM@2PLE	PLEDB2	Enterprise PL/I with DB2
FLM@2PLF	PLEDB2F	DB2 and PL/I Enterprise compiler and NCAL linkedit to a sub-module library with Fault Analyzer side file generation

Define the language definitions

If you are not going to name the actual types and languages "DB2CLIST" and "DB2OUT", do the following steps:

1. Copy the sample language translators to your own source library with the names you choose.

   Modify the LANG values on the FLMLANGL macro for the DB2CLIST and DB2OUT language translators and the DFLTTYP and LANG values on the FLMALLOC macros in the DB2CLIST translators to reflect your naming conventions.

For example if you are using separate translators for binding plans and packages, and the DB2CLIST and DB2OUT members are stored in different SCLM types for plans and packages, the FLMALLOC statements might look like this:

```
FLMLANGL    LANG=PLNCLIST
   ⋮
FLMALLOC  IOTYPE=O,DDNAME=ISRDB2OT,KEYREF=OUT3,LANG=PLNOUT, C
    RECNUM=5000,LRECL=80,RECFM=FB,DFLTTYP=PLNOUT
```

Figure 152. DB2CLIST example for plans

```
FLMLANGL    LANG=PKGCLIST
   ⋮
FLMALLOC  IOTYPE=O,DDNAME=ISRDB2OT,KEYREF=OUT3,LANG=PKGOUT, C
    RECNUM=5000,LRECL=80,RECFM=FB,DFLTTYP=PKGOUT
```

Figure 153. DB2CLIST example for packages

Remember to change the LANG values on the DB2OUT translator, so that LANG=PLNOUT and LANG=PKGOUT.

2. Modify the DBRMTYPE values in the OPTIONS parameter on the FLMTRNSL macros in the language definitions to reflect your naming conventions. This tells SCLM which DBRM libraries to allocate during the bind step to the DBRMLIB DD. Be sure to make this change in both the DB2CLIST and DB2OUT translators.

```
* BUILD TRANSLATOR(S)
*
         FLMTRNSL   CALLNAM='DB2 BIND',                              C
              FUNCTN=BUILD,                                          C
              COMPILE=FLMCSPDB,                                      C
              PORDER=1,                                              C
              GOODRC=4,                                              C
              CALLMETH=LINK,                                         C
              OPTIONS=(FUNCTN=BUILD,                                 C
              OPTION=BIND,                                           C
              SCLMINFO=@@FLMINF,                                     C
              PROJECT=@@FLMPRJ,                                      C
              ALTPROJ=@@FLMALT,                                      C
              DBRMTYPE=DBRM,                                         C
              GROUP=@@FLMGRP,                                        C
              MEMBER=@@FLMMBR)
```

Figure 154. Defining DBRMTYPE in DB2CLIST translator

Step 9: Assemble and link the project definition

There are no additional considerations.

Information for developers

This diagram shows the flow of processing through the translators. It shows the inputs to and outputs from the various phases.

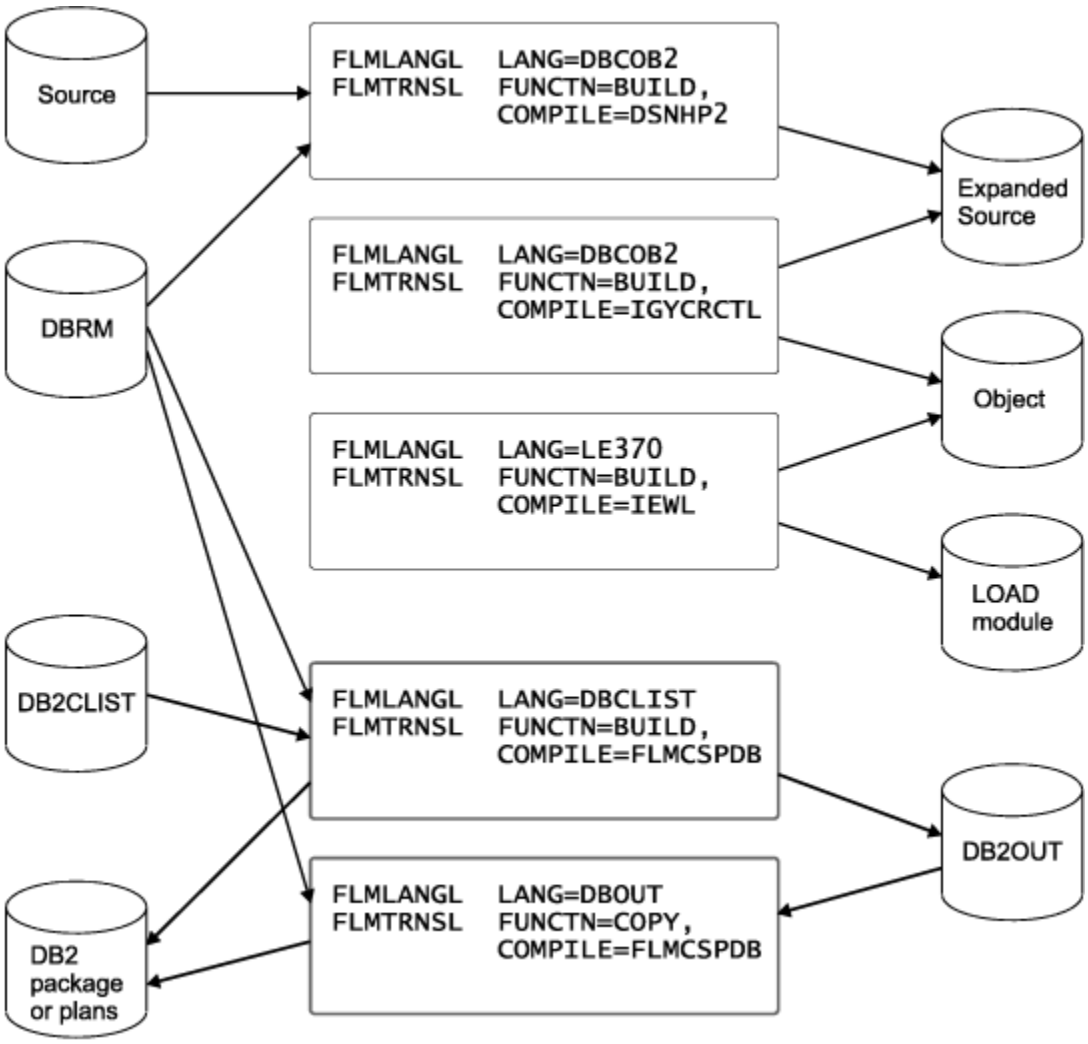


Figure 155. DB2CLIST: flow of processing through the translators

Getting started

See:

- “Create a program that has SQL statements” on page 300
- “Create a generic architecture definition to control the bind” on page 301
- “Create a high-level (HL) architecture definition to link link-edit to bind” on page 301
- “Alternative High Level (HL) architecture definition to link link-edit to bind” on page 301
- “Other architecture definition considerations” on page 301
- “Create DB2 CLIST” on page 302

Create a program that has SQL statements

The first step is to have a program that has SQL statements. To control the building of this program you would normally use a Linkedit Control (LEC) architecture definition. This defines all the source modules in your load module. It will also include the DB2 Language module. For example, a LEC architecture definition for an MVS batch program with DB2 might be created as follows:

```
INCLD SCLMDB2P SOURCE
CMD   INCLUDE SYSLIB(DSNELI)
*
PARM RMODE(24),AMODE(31)
*
LOAD  SCLMDB2P LOAD
LMAP  SCLMDB2P LMAP
```

Figure 156. Sample LEC architecture definition

Create a generic architecture definition to control the bind

The DB2CLIST contents will be copied to the DB2OUT member. This is so that the DB2OUT member can be used in the bind process for levels in the hierarchy higher than the development group. Before creating the actual DB2CLIST member we can create the generic architecture definition that will define this process. There are a couple of ways this can be achieved by using SINC or INCLD depending on how your translators have been defined. This example relates to how the sample translators shipped with SCLM have been defined.

```
SINC  SCLMDR2P DB2CLIST
OUT3  SCLMDR2P DB2OUT
```

Figure 157. Sample generic architecture definition for bind member

Create a high-level (HL) architecture definition to link link-edit to bind

So that SCLM knows the correct processing scope a High level (HL) architecture definition should be created to control both the compilation/link-edit and the bind. This HL architecture definition will just include the two previously defined architecture definitions.

```
INCL SCLMLECP ARCHDEF * Linkedit control archdef
INCL SCLMDR2P ARCHDEF * Bind control generic archdef
```

Figure 158. Sample HL architecture definition for overall compilation, link-edit and bind

Alternative High Level (HL) architecture definition to link link-edit to bind

It is possible to use the SINC or INCLD keywords to drive the bind process. The previous two steps discussed creating a generic architecture definition for the bind, using the SINC and OUTx keywords, then linking this generic architecture definition to the LEC architecture definition by means of a HL architecture definition.

It is possible, and might be preferable in your environment, not to use a generic architecture definition to drive the bind. Instead, you can just use the INCLD keyword in the HL architecture definition for the DB2CLIST member plus the INCL keyword for the LEC architecture definition. This way the default processing of the translator will run creating the DB2OUT member. This process will work with the shipped samples.

```
INCL  SCLMLECP ARCHDEF
INCLD SCLMDR2P DB2CLIST
```

Figure 159. Sample HL architecture definition for overall compilation, link-edit and bind (no generic architecture definition)

An advantage to this method is that it means you will have one less part, as the generic architecture definition is not required.

Other architecture definition considerations

Another consideration is using SCLM's ability to have the same member names across different types, thus linking your parts with a common name. Take the scenario where you have a COBOL program called SCLMDB2P that creates a load module of the same name SCLMDB2P.

You may want to name the LEC architecture definition and the HL architecture definition that controls the compilation, link and bind the same name. This can be achieved by creating a different SCLM type, for example LECDEF. This member will contain the Linkedit control architecture definition as described in Figure 156 on page 301. Then your HL architecture definition will reside in your ARCHDEF type and will reference the LECDEF and the DB2CLIST as shown in Figure 160 on page 302.

```
INCL  SCLMDB2P LECDEF
INCLD SCLMDB2P DB2CLIST
```

Figure 160. Sample HL architecture definition for overall compilation, link-edit and bind (no generic architecture definition)

This way your program source, OBJ, DBRM, LOAD, DB2CLIST member, LECDEF member and HL architecture definition member will all have the same member names. If you wish to utilize this implementation, your SCLM Administrator will need to define the SCLM type that will hold the Linkedit architecture definition members.

Create DB2 CLIST

You must create a DB2CLIST member for each DB2 application plan or package. The DB2CLIST is a TSO CLIST or REXX procedure that allows you to BIND or FREE the DB2 application. This exec should contain code to perform the following functions:

- Allow different DB2 Subsystem names to be assigned to each group
- Allow different DB2 Owner and Qualifier and other parameters to be assigned to each group
- BIND the application plan or package
- Optionally FREE the application plan or package.

The parameters and logic required are shown in Figure 161 on page 303.

The DB2CLIST member allows you to specify which DBRMs are bound into the application plan or package by use of the %INCLUDE statement. The INCLUDE statement consists of a %INCLUDE keyword followed by the name of the included DBRM. SCLM parses the DB2CLIST member and keeps a list of included DBRM names, as well as other accounting information in the account record for the DB2CLIST member. The INCLUDE directive and DBRM name must be on the same line. The format of the INCLUDE statement is:

```
/*  %INCLUDE dbrm-name   */
```

You can look at the names of included DBRMs for a DB2CLIST by browsing its accounting information:

1. Select the Utilities option from the SCLM Main Menu.
2. Select the Library option from the SCLM Utilities Menu.
3. From the SCLM Library Utility - Entry Panel, enter the DB2 type to be used during Build.
4. From the list of members, select the DB2CLIST that you want to examine and browse its accounting information.
5. From the Accounting Record for the DB2CLIST, select the Number of Includes.
6. Finally, you see the list of included DBRMs in the DB2CLIST.

The DB2CLIST is usually built and promoted by using an architecture definition. Use the SINC or INCLD keyword to reference the member from an architecture definition as shown previously. The DB2CLIST member can also be built or promoted directly without using an architecture member. When the DB2CLIST member is built or promoted directly or is processed through an INCLD architecture definition keyword, SCLM uses the defaults defined in the DB2CLIST language definition.

```
/* REXX */
   Arg INPARMS
   Parse var INPARMS '(' OPTION ' ' '(' GROUP ' '
   /*******************************************************************/
   /* SPECIFY AN INCLUDE FOR THE DBRM TO BE INCLUDED IN THE DB2       */
   /* PACKAGE. SCLM TRACKS DEPENDENCY ON DBRMS WITH THE COMMENTED     */
   /* OUT INCLUDE STATEMENT.                                          */
   /*******************************************************************/
   /*** THE INCLUDE STATEMENT MUST REMAIN COMMENTED OUT.            ***/
   /*******************************************************************/
   /*                                                                 */
   /* %INCLUDE SCLMDB2P                                               */
   /*                                                                 */
   /*******************************************************************/
   /* SET UP THE PARMS FOR A PACKAGE BIND.                            */
   /*******************************************************************/
   MEMBER  = "SCLMDB2P"
   EXPLAIN = "NO"
   /*---------------------------------------*/
   /* CALL SCLMBIND EXEC TO PERFORM BIND     */
   /*---------------------------------------*/
   PARMS = OPTION GROUP MEMBER EXPLAIN
   Address TSO "EX 'SCLMTEST.PROJDEFS.SOURCE(REXXBIND)' '"PARMS"'"
   EXITCC = RC

Exit EXITCC
```

Figure 161. DB2CLIST generic example

We can see in the above example the following key points

- The DB2CLIST translator will pass in an Option and a Group. The OPTION will be BIND if the translator is executed at Build time or at promote time during the COPY phase. The OPTION will be FREE if the translator is in the PURGE phase of the promote.
- The DBRM name is specified in the commented out %INCLUDE. This will maintain SCLM dependency between the DBRM member and the DB2CLIST that needs to be run to do the bind.
- Certain parameters are set up for binding such as EXPLAIN. Other parameters could also be set up, such as ISOLATION and VALIDATE for example. If there are bind options that are specific to a particular member then they can be set here.
- The bind processor is called to do the actual bind and set up other parameters based on the Group where the bind needs to occur. The actual bind could be coded in the DB2CLIST but it is better practice to keep your bind processor separate from the DB2CLIST members. This means that if any bind options change in future, the DB2CLIST might not need to be changed, only rebuilt, thus minimizing the changes required.
- It is possible to include a "Select" on the group and set group-specific parameters there rather than in the bind processor. It is also possible to include the bind invocation in the DB2CLIST but this is not advisable, as if a certain bind parameter changed then all the DB2CLIST members would need to change.

This is an example of the bind processor:

Bind exec example (Part 1 of 2)

```
/* REXX */

   Arg OPT GRP MEMBER EXP

   /*  Set bind options based on group  */

   rcode = 0

   Select
      When (GRP = 'PROD') Then
      Do
         SUBSYS    = 'DI21'
         OWNER     = 'PRODDBA'
         ACTION    = 'REP'
         VALIDATE  = 'RUN'
```

```
         ISOLATION = 'CS'
         EXPLAIN   = 'NO'
         QUALIFIER = 'PRODDBA'
      End
      When (GRP = 'TEST') Then
      Do
         SUBSYS    = 'DI21'
         OWNER     = 'TESTDBA'
         ACTION    = 'REP'
         VALIDATE  = 'BIND'
         ISOLATION = 'CS'
         EXPLAIN   = 'NO'
         QUALIFIER = 'TESTDBA'
      End
      When (GRP = 'DEV1') Then
      Do
         SUBSYS    = 'DI21'
         OWNER     = 'DEV1DBA'
         ACTION    = 'REP'
         VALIDATE  = 'BIND'
         ISOLATION = 'CS'
         EXPLAIN   = EXP
         QUALIFIER = 'DEV1DBA'

         Call Bind_it
      End
      When (GRP = 'DEV2') Then
         NOP          /* no bind needed */
      Otherwise
         NOP          /* no bind needed */
   End

Exit RCODE
```

Bind exec example (Part 2 of 2)

```
Bind_it:

   If OPT = 'BIND' Then
   Do
      /* Create a bind control statement as a single long line.  */

      DB2_Line = "BIND PACKAGE("GRP") MEMBER("MEMBER")" ||,
                     " OWNER("OWNER")"           ||,
                     " ACTION("ACTION")"         ||,
                     " VALIDATE("VALIDATE")"     ||,
                     " ISOLATION("ISOLATION")"   ||,
                     " EXPLAIN("EXPLAIN")"       ||,
                     " QUALIFIER("QUALIFIER")"


      /*  Write the bind control statement to the data queue and execute  */
      /*  DB2 to perform the bind.                                        */

      queue DB2_Line
      queue "End"
      Address TSO "DSN SYSTEM("SUBSYS")"
      rcode = RC
   End
   If OPT = 'FREE' Then
   Do
      /*  At this point SCLM passes the from-group information so that    */
      /*  the package at the from-group can be FREEd if required         */
   End

Return
```

More complex scenarios

The examples and samples in “Getting started” on page 300 cover some simple implementations of DB2 binding. Examples of some more complex requirements are discussed in this topic.

Storing bind options in a bind control file

Often the bind options do not follow a pattern. In this case the bind exec will need to read a data set where bind options are stored for each member. The bind options for the member would be parsed out and formed into the bind statement before the bind command is submitted.

This might also be a way for your DB2 database administrators to control the setting of the bind options so that they are stored in one place and the developers do not need to worry about them.

Binding on different LPARs

The SCLM bind processor in the above examples only works for binds on the machine where SCLM exists. For binds on other LPARs, the bind processor must use file tailoring to create a job to perform the bind. If the JES queue is shared between the LPARs then the job can be submitted from the bind exec with the appropriate routing parameter to submit the bind on the other LPAR. If the JES queue is not shared then a more complicated delivery of the bind JCL will be needed, possibly using FTP.

If you want to implement binds on other LPARs through a submitted job, it is best to create them in the Build user exit or Promote user exit or both. This way, only one bind job must be created for a package build or promote. See Chapter 2, “User exits,” on page 53 for more information.

For example, assume that your development and test environments reside on the same LPAR that SCLM is running on, but your PROD database and executables are on a different LPAR. In this case your bind processor would use the DB2CLIST method of binding for the DEV builds and promotes to TEST. However, when the promote to PROD is actioned, the DB2CLIST and DB2OUT members are still promoted to PROD but no bind is done. Instead, once the promote has finished the SCLM work, the promote copy user exit or purge user exit is invoked. The exit is coded such that a list of the DBRMs being promoted is extracted from the PROMEXIT file. The processor then builds a job using ISPF file tailoring techniques that will do the bind for all the required members. The generated job is transferred and submitted to the production LPAR.

Rebinding at lower levels after a promotion

The DB2 catalog tables relating to the bind contain the actual data set location of the DBRM that was used to perform that specific bind. However, when the code is promoted to the next level of the hierarchy, the information in the DB2 Catalog is now technically incorrect as the DBRM no longer resides in the location specified. This may cause problems if the DB2 REBIND option is used.

Once a DBRM is promoted to the next level of the hierarchy it is good practice not only to bind at the To Group level, but to issue a "back-bind" that also binds the package or plan at the From Group level. To ensure that this back-bind is executed correctly, the From Group DBRM library should be concatenated in front of the To Group DBRM library just in case the program has already been changed in the From Group. Of course this may not be necessary as the SQL that is executed by the package or plan is correct, it is just the DBRM member location that is now out of date.

Chapter 14. SCLM support for workstation builds

You can store the source for workstation applications in SCLM. You can then use the configuration functions to build and promote the application. The build function transfers the source to an ISPF connected workstation, runs the compiler or other workstation tool, and then stores the results back into SCLM.

Storing workstation applications in SCLM provides several benefits:

- You can use SCLM as a single point of access for the workstation code.
- You can protect and back up the application source, executables, and outputs using the host.
- Host applications and workstation applications can share source.
- You can use SCLM's configuration management to ensure that the application is current.
- You can use the library management and versioning capabilities to track the application parts through the hierarchy and to retain backup versions.

Note: If the SAF resource ISPF.WSA is defined in the FACILITY class and you are not permitted access to that resource, you will not be able to establish a connection to the workstation.

Requirements

Because of the differences in MVS and the workstation operating system, you must meet the following requirements for SCLM to store the application source:

- The file names must follow ISPF member naming conventions and cannot be more than 8 characters. Workstation file names can be in uppercase, lowercase, or have initial capital followed by lowercase letters. This mapping is specified using the `WSCASE` keyword in the ACTINFO file.
- Use consistent naming conventions for the extension names and subdirectory layout. The workstation build translator provided with SCLM (FLMLTWST) maps type names to extensions and subdirectories. Consistent use of the extension and subdirectory names across the workstations that you use will make sure that the mapping will work properly.
- Use consistent command names. The commands are defined by input data to the FLMLTWST translator.

Overview of workstation build

The only distinction that SCLM makes between a workstation application and a host application is where the compiler and other tools reside. The application source and the outputs from builds are stored in PDS data sets on the host. The result is that all of the SCLM functions work the same for a workstation application as they do for a host MVS application except for build.

The difference between building a workstation application and a host application is that special build translators are used for the workstation application. The user doing the workstation build must use a workstation.

SCLM provides three build translators to build workstation applications. One translator, FLMLTWST, is the driver and calls the other translators to perform various tasks. To allow customization of the events that take place during a workstation build, the FLMLTWST translator is written in REXX. This allows the translator to be customized to meet the project's needs. The FLMLTWST translator performs the following tasks:

- Initialization and set up

  SCLM checks the parameters, retrieves and checks the workstation information, sets up file name mapping information, and sets up command information.

© Copyright IBM Corp. 1990, 2019

- Build map parsing

  FLMLTWST calls the FLMTBMAP translator to get the contents of the build map for the member being built. FLMLTWST parses the information in the build map to get the list of inputs that must be transferred to the workstation and any additional parameters that have been specified for the workstation command, such as a compiler or other tool. FLMLTWST also gets the list of outputs after the command is complete.

  At the same time, the SCLM member names are mapped to workstation file names based on the file name mapping information.

- Construct command parameters

  FLMLTWST supports running multiple workstation commands during each invocation. The parameters for each of the commands are put together based on the parameters passed to FLMLTWST, the contents of the build map (input and output file names can be included in the parameters), and on the workstation command information.

- Response file construction

  Some workstation commands support passing parameters using a file called a *response* file. If the workstation command information specifies a response file, one is created in a temporary data set and will be sent to the workstation with the other workstation command inputs.

  If multiple workstation commands will be issued, the response file for the first workstation command is sent with the input files. Response files for later commands are sent just before each command is run.

  Response files are only generated and sent to the workstation if the workstation command information indicates that one is to be used. If no response file is used, the command parameters are specified with the workstation command.

- Transfer inputs to the workstation

  FLMLTWST constructs a list of the input files (includes, source members, and response file) to be sent to the workstation. The FLMTXFER translator is then called to send the files to the workstation. FLMTXFER uses the FILEXFER service to transfer files to the workstation.

  The FLMTXFER translator keeps track of the SCLM members that have been sent to the workstation. This record is used to ensure that include members and source members are only transferred to the workstation once to reduce the time required to build a workstation application. The record of what has been transferred to the workstation is preserved in memory allocated by SCLM build. The result is that, within a single SCLM build, FLMTXFER only downloads a member once no matter how many source members that include it are built.

  If the date and time of the host member's statistics are the same as the date and time of its workstation counterpart, SCLM assumes that they are the same, and does not download the member a second time.

- Perform the workstation command

  FLMLTWST constructs the workstation command based on the information obtained in the set-up step. The command is issued on the workstation and SCLM waits for the result.

  Repeat this step for each workstation command that will run for the member being built. Before each command is issued, a response file is constructed and transferred to the workstation if needed.

- Transfer the outputs to the host system

  FLMLTWST uses a list of outputs obtained from the build map to construct a list of files to transfer from the workstation to the host system. The FLMTXFER translator performs the transfer from the workstation to the host. The data sets where the files are transferred are the data sets allocated to the ddname specified in the translator definition for FLMLTWST. If FLMLTWST ends successfully, build transfers the members into the SCLM hierarchy.

  If you have set the FLMALLOC macro IOTYPE=P, the date and time on the host member statistics are synchronized with the date and time of the corresponding workstation file, so that if the member is used for another build step, it will not be downloaded again.

Information for the project manager

You must consider several things when setting up a project to support workstation applications. This section covers items that are specific to workstation applications. See Chapter 1, “Defining the project environment,” on page 3 for information about general project setup.

Naming conventions

Determine what SCLM type names to use and the mapping between SCLM type names and workstation file extensions.

The recommended approach is to have a one-to-one mapping between the SCLM type and the workstation extension. In addition to the type-to-extension mapping, SCLM needs to know the format of the data within each type (ASCII text or binary). To avoid having to define a mapping for each type, use something in the type name that indicates the format of the data. For example, add `BIN` to the workstation extension to create the SCLM type names for types that will contain binary data. This will minimize the number of mapping definitions for the ACTINFO file, because the wildcard character can be used to define a pattern in the type and extension names.

Another approach is to merge several workstation extensions into the same SCLM type. In this case, the workstation file names without the extension must be unique. The drawback of this approach is that after the files are combined into one SCLM type, they lose their individual extensions. The mapping is **from** the type **to** the workstation. SCLM does not know what a file was once called on the workstation. Only one extension can be defined for each type. This means that when the files are combined, SCLM will use the same extension for all of them when transferring them from or to the workstation. This may or may not be a problem, depending on the type of data combined. It would not be a good idea, for example, to combine C++ header files with H and HPP extensions into the same SCLM type, because the C++ source members might include header files with one or both of those extensions and would not find them if the extensions were changed. There might be other situations where the loss of the extension identity wouldn't matter.

Workstation file names, excluding the paths and extensions, must be valid ISPF PDS member names. Workstation file names can be in uppercase, lowercase, or have initial capital followed by lowercase letters. This mapping is specified using the `WSCASE` keyword in the ACTINFO file.

Languages

Next, you need to know which languages you will need.

One way to do this is to create a complex language definition that performs all of the steps required to go from source to executable code or to whatever you want the final result to be. The drawback to this approach is that when anything changes all of the steps are performed rather than the minimal set. For example, suppose there was a language that:

1. Compiled C source to an .obj
2. Compiled the resource source to an .res
3. Linked the .obj files into an .exe
4. Ran the resource compiler to add the resources from the .res to the .exe file

If the resource source changes, all of those steps are performed when some of them could be avoided.

Another approach is to create a language for each step. However, some tools produce outputs that are only needed until the next command is run. For example, the output from step “3” on page 309 should not be saved into the hierarchy until after the resource compiler has been run. Saving one .exe into the SCLM hierarchy from the compiler and another copy from the resource compiler increases the project data set size and the time required to build.

A better approach is to create languages for each step that produces outputs that are kept permanently in the hierarchy. So, for the previous example, you would need three languages:

1. One language to compile C source and store the .obj files

2. One language to compile the resource source and store the .res files
3. One language to link the .obj files and add the resources from the .res files.

What workstation tools will you use?

The ACTION parameter on the FLMLTWST translator determines the workstation command that is run. The FLMLTWST translator maps the actions to a workstation command, determines the basic parameters to pass to that command, maps the workstation extensions to input and output parameters, and then orders the parameters.

In addition to the ACTION specified by the language definition, you can perform other actions in a build step by use of the CMD ACTION statement. For more information about the CMD statement and its use with workstation applications, refer to "FLMLTWST Workstation Build translator" on page 608.

What parameters do you need for the workstation tools?

Specify parameters in three places:

- In the translator (FLMLTWST). The parameters specified in FLMLTWST are used for every member of every language that calls it. They should be only the parameters that FLMLTWST requires, such as the parameters that specify the input and output file names.

  You can specify parameters to FLMLTWST for the workstation command in three ways:

  – In the language definition and on architecture PARM statements
  – On the architecture CMD statement
  – Using parameters that are associated with inputs and outputs

  The order of the parameters is specified in the input data to the FLMLTWST translator and is the order required by the workstation command.
- On the FLMTRNSL macro in the language definition. These parameters are used for every member of the language. These should be parameters that the project requires. For example, the /Kg+ parameter can be specified to ensure that messages are produced for all GOTO statements.
- In an architecture member. These parameters are specific to a member. For example, the /DAPPL=A parameter can be used to define a preprocessor macro.

Workstation information

The FLMLTWST translator needs information about the workstation such as the response file name and the directory name to prefix all files transferred to or from the workstation. It gets this information by reading from a data set.

The naming convention for the data set must be identified so that you can specify it in all the language definitions. Typically, the same information is used for all languages, although it is not required. The naming convention can include variables to substitute the userid, project, group or other information into the data set name pattern. The variables used depend on where builds take place and on local data set naming standards. If the user determines the workstation, the userid should be part of the data set name. If the group determines the workstation, the group variable should be used without the userid variable. For more information, refer to "USERINFODD statements" on page 611 and "FLMCPYLB macro" on page 538.

More information on SCLM, ISPF, and workstation builds

- This chapter contains information on SCLM support for workstation builds on OS/2 and Windows.
- For information about the ACTINFO files, USERINFO files, and workstation language definitions, refer to "FLMLTWST Workstation Build translator" on page 608.
- For information about the FLMLRC2 sample parser, see page "FLMLRC2 C, C++, and Resource file parser for workstation source" on page 594.

- For information about the FLMLRIPF sample parser, see page “FLMLRIPF Script and OS/2 IPF Source Parser” on page 601.
- For information on setting up SCLM or PDF to view and edit on the workstation, see the topic "Installing the Client/Server component" in the *z/OS ISPF User's Guide Vol I*.
- The International Technical Support Centers (ITSC) *Version 4 of ISPF and SCLM Implementation Guide*, GG24-4407, provides a good overview of SCLM and the ISPF Client/Server.

ISPF Sample and Macro libraries

The ISPF Sample and Macro libraries contain a number of files to support SCLM workstation builds. The ISPF Sample Library contains the following members:

FLMWBMIG
Sample migration EXEC for IBM CSET++ for OS/2 "Hello World 6" sample.

FLMWBUSR
Sample USERINFO file.

FLMWBAIO
Sample ACTINFO file for IBM CSET++ for OS/2 "Hello World 6" sample.

FLMWBAIW
Sample ACTINFO file for Borland C++ "Hello World" sample.

FLMWBPRJ
Sample workstation project definition.

FLMWBJCL
Sample JCL to allocate the data sets for the FLMWBPRJ sample project.

FLMWBTMP
Sample workstation language definition template.

FLMWBOS2
High-level architecture definition to build IBM CSET++ for OS/2 "Hello World 6" sample.

FLMWBIPF
Architecture definition to build IBM CSET++ for OS/2 "Hello World 6" help file.

FLMWBDLL
Architecture definition to build IBM CSET++ for OS/2 "Hello World 6" DLL file.

FLMWBEXE
Architecture definition to build IBM CSET++ for OS/2 "Hello World 6" EXE file.

FLMWBWIN
High-level architecture definition to build Borland C++ "Hello World" sample.

The Macro Library contains sample language definitions for OS/2 and Windows. The IBM CSET++ for OS/2 language definitions are:

FLM@WICC
Compile

FLM@WDUM
Compile dummy object to hold DLLs

FLM@WEXE
Link EXE

FLM@WIPF
Build Help

FLM@WLNK
Link386 to Link the DLL

FLM@WRC
Resource compile

The Borland C++ for Windows language definitions are:

FLM@WBCC
Compile

FLM@WBRC
Resource Compile

FLM@WTLK
TLINK OBJ to EXE

The IBM CSET++ for AIX® sample language definition is:

FLM@WXLC
Compile

Information for the developer

See:

- "Migrating applications into SCLM" on page 312
- "Architecture definition members for workstation applications" on page 312
- "Specifying options" on page 313
- "Including outputs from other build steps" on page 313
- "Running multiple workstation commands" on page 313

Migrating applications into SCLM

To migrate a workstation application into SCLM:

1. Get the following project information from the project manager:
   - The name of the development group where the members will be stored
   - The type names and their mapping to workstation file extensions
   - The languages to use for source members
   - The default parameters specified in the language definition for each language
   - The actions and defaults specified in the ACTINFO file for workstation build
2. Transfer the application source to the MVS system into the data sets for the development group based on the *workstation file to SCLM type name* mapping established for the project.

   Files containing data that can be edited on MVS must be transferred with ASCII-to-EBCDIC translation. Other files can be transferred in binary format (no translation). The FILEXFER service is recommended to avoid possible translation problems.
3. Migrate the members into SCLM using the languages supplied by the project manager.
4. Create architecture definition members as needed.

Architecture definition members for workstation applications

Architecture definition members must be created in any of the following cases:

- The source member requires options that were not specified in the language definition or action information data set.
- You need to override the inputs or outputs used in the language definition.
- The output member names are not the same as the source member name. See "Statement uses" on page 278 for a description of the output keywords for architecture members.

  Some things can be done in the language definition to support adding a prefix or suffix to the output member name, but these capabilities do not support all possibilities. For more information, refer to the DFLTMEM parameter on the FLMALLOC macro.

- Outputs from the builds of other members are inputs to this build, for example, linking object modules together.
- Multiple workstation commands must be issued to complete the build step.
- To specify a relationship between components other than the source-to-include and input-to-output relationships generated by SCLM. An example would be to specify a relationship between the executable, DLL, and help components of a workstation application.

Specifying options

Options can be specified to the workstation compiler, linker, or other tool by using the architecture definition CMD statement. This statement must be followed by the keyword PARMS and the parameters that are passed to the workstation tool. In the following example, the option '/Ss' is added to the options passed to the workstation tool.

```
SINC  SAMPLE   C       * source member
OBJ   SAMPLE   OBJBIN  * generated object member
LIST  SAMPLE   LISTING * listing file
*
* The following CMD statement has compile options for this member
*
CMD PARMS /Ss
```

Figure 162. Specifying Options in a Workstation Architecture Definition

If multiple CMD PARMS statements appear in the architecture member, the options are passed to the workstation tool in the order they appear in the architecture member. They are added to the workstation command as specified in the ACTINFO input to the FLMLTWST translator.

If you want to add options to be passed to the FLMLTWST translator, you can use the PARM and PARMx architecture statements. However, these options are considered FLMLTWST options rather than options for the workstation command.

Including outputs from other build steps

Use the architecture definition statements INCLD, INCL, and SINC to include members that are outputs from building other members. Using the INCLD and INCL statements ensures that SCLM builds the correct member to generate the output.

When a CC or generic architecture definition is built, SCLM uses the language definition of the member on the first SINC statement. For LEC architecture definitions, the LE370 language is used. To override the language, specify the LKED architecture statement with the name of the language definition to use.

The following example shows an architecture member that can link several object members together to produce an .exe file. The language of EXE is used.

```
INCL  SAMPLE   ARCHDEF * archdef which produced sample object
INCLD COMMON   C       * source member which produced common object
*
LKED  EXE
*
LOAD  PROG1    EXEBIN  * .exe file
LMAP  PROG1    MAP     * listing file
```

Figure 163. Including Outputs as Inputs

Running multiple workstation commands

Building some members requires that multiple workstation commands be issued. The FLMLTWST translator issues a workstation command for each action it finds. The first action is the one specified by the ACTION parameter to FLMLTWST in the language definition, or the default action if none is specified. Additional actions can be performed by using the architecture CMD statement with the ACTION keyword. The ACTION keyword must be followed by an action defined in the FLMLTWST translator.

Figure 164 on page 314 shows an architecture member that links two object modules together and then runs another workstation command before transferring the outputs to the MVS system. In this example, the second command runs the OS/2 resource compiler to add the information from a binary resource file to the .exe generated by the link.

```
*
LKED EXE                * link language
*
KREF OBJ                * include generated object modules
*
INCL  MAHJONGC ARCHDEF  * archdef that produces MAHJONGG OBJBIN
INCL  TILE     ARCHDEF  * archdef that produces TILE OBJBIN
SINC  MAHJONGG DEF      * DEF source
*
LOAD  MAHJONGG EXEBIN   * Generated .exe file
LMAP  MAHJONGG MAP      * Generated .map file
*
* Run resource compiler after the link completes
*
CMD ACTION RCEXE
*
KREF OUT1               * include generated .res file
*
INCLD MAHJONGG RC       * Source that produces MAHJONGG RESBIN
*
```

Figure 164. Multiple Workstation Commands

The order of the INCL and INCLD statements in the previous example is not important. The FLMLTWST translator determines which files are inputs to each step based on information defined in the translator. The appropriate options are also added for each of the inputs and outputs by the FLMLTWST translator.

Sample language definition

Figure 165 on page 315 shows a language definition for compiling C source members on the workstation. A description of the items in the language definition follows.

```
*********************************************************************
*                                                                   *
*     SCLM LANGUAGE DEFINITION FOR IBM CSET/2 OR CSET++ FOR OS/2    *
*                      COMPILE SOURCE TO OBJECT                     *
*                                                                   *
*********************************************************************
*
*
CPPOS2   FLMLANGL    LANG=CPPOS2,                                    C
               VERSION=2,                                           C
               CHKSYSLB=IGNORE
*
          FLMINCLS TYPES=(H,HPP,@@FLMTYP,@@FLMETP)
H         FLMINCLS TYPES=(H)
HPP       FLMINCLS TYPES=(HPP)
*
* PARSER
*
          FLMTRNSL   CALLNAM='C/C++ PARSE',                        C
               FUNCTN=PARSE,                                       C
               CALLMETH=TSOLNK,                                    C
               COMPILE=FLMLRC2,                                    C
               PORDER=1,                                           C
               OPTIONS=(STATINFO=@@FLMSTP,                         C
               LISTINFO=@@FLMLIS,                                  C
               LISTSIZE=@@FLMSIZ)
*
*          (* SOURCE *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
* BUILD
*
          FLMTRNSL   CALLNAM='C/C++',                              C
               FUNCTN=BUILD,                                       C
               CALLMETH=ISPLNK,                                    C
               COMPILE=SELECT,                                     C
               VERSION=1,                                          C
               GOODRC=0,                                           C
               PORDER=1,                                           C
               OPTIONS='CMD(FLMLTWST ACTION=COMPILE,BMAPINFO=@@FLM$MP,SC
               CLMINFO=@@FLMINF,BLDINFO=@@FLMBIO,PARMS='
*
*          (* OBJ *)
           FLMALLOC  IOTYPE=P,RECFM=VB,LRECL=1024,                  C
               RECNUM=4000,DDNAME=OBJ,CATLG=Y,KEYREF=OBJ,           C
               DFLTTYP=OBJBIN,DFLTMEM=*,LANG=EXE
*          (* LIST *)
           FLMALLOC  IOTYPE=O,RECFM=VB,LRECL=256,                   C
               RECNUM=4000,DDNAME=LIST,CATLG=Y,PRINT=I,             C
               KEYREF=LIST,DFLTTYP=LST
```

Figure 165. Workstation C Language Definition (Part 1 of 2)

```
*          (* USERINFO *)
           FLMALLOC  IOTYPE=A,DDNAME=USERINFO
             FLMCPYLB @@FLMUID.SCLM.USERINFO
*          (* ACTINFO *)
           FLMALLOC  IOTYPE=A,DDNAME=ACTINFO
             FLMCPYLB @@FLMPRJ.PROJDEFS.ACTINFO
*          (* MESSAGE *)
           FLMALLOC  IOTYPE=W,RECFM=VB,LRECL=256,DISP=MOD,             C
               RECNUM=4000,DDNAME=MESSAGE,PRINT=I
*          (* MSGXFER *)
           FLMALLOC  IOTYPE=W,RECFM=VB,LRECL=256,CATLG=Y,              C
               RECNUM=4000,DDNAME=MSGXFER
*          (* BMAP *)
           FLMALLOC  IOTYPE=W,RECFM=VB,LRECL=256,                      C
               RECNUM=4000,DDNAME=BMAP,PRINT=I
*          (* FILES *)
           FLMALLOC  IOTYPE=W,RECFM=VB,LRECL=256,CATLG=Y,              C
               RECNUM=4000,DDNAME=FILES,PRINT=I
*          (* RESPONSE *)
           FLMALLOC  IOTYPE=W,RECFM=VB,LRECL=256,                      C
               RECNUM=4000,DDNAME=RESPONSE,PRINT=I,CATLG=Y
*
```

Figure 166. Workstation C Language Definition (Part 2 of 2)

FLMLANGL macro
This macro specifies the language name, CPPOS2, the language version, "1", and that SCLM is to ignore any includes that are not in the project hierarchy.

FLMINCLS macro
This macro indicates the types searched when looking for includes. Includes with the workstation file extension 'h' are found in the H type. Other includes are found in the type of the source member or its extended type.

FLMTRNSL macro (functn=parse)
This macro identifies the parser to use when the members of this language are updated. The parser scans the member for include dependencies and counts statistics. For a description of the FLMLRC2 sample parser, see "FLMLRC2 C, C++, and Resource file parser for workstation source" on page 594.

FLMTRNSL macro (functn=build)
This is the definition of the build translator. It calls FLMLTWST to perform the compile on the workstation. The ACTION parameter is set to compile to indicate that the compiler is to be called. The PARMS parameter at the end of the parameter string allows for PARM keywords in the language definition to specify additional parameters. The other parameters are used to pass information between SCLM build and the translators that FLMLTWST calls.

FLMALLOC macro (ddname=obj)
This macro allocates the ddname that will hold the .obj file generated on the workstation. The RECFM and LRECL values must match the allocation of the data set in the hierarchy where the .obj file will be stored.

IOTYPE=O
Indicates that a sequential data set will be allocated to hold the output.

IOTYPE=P
Indicates that a partitioned data set will be allocated to hold the output. Using IOTYPE=P can improve build performance for builds with more than one step by copying the date and time of the workstation file to the host member. If the file is needed for subsequent build steps, the copy on the workstation will be used rather than downloading the file that was just uploaded.

DFLTMEM=*
Indicates that the output member in the PDS will have the same name as the member being built.

RECNUM
Indicates the maximum number of records that can be stored in the data set

CATLG=Y
Allows the file to be transferred from the workstation to the data set allocated to this ddname.

KEYREF=OBJ
Indicates that this is an object module. This references the architecture OBJ statement. See “Architecture statements” on page 277 for more information on architecture statements.

DFLTTYP
Indicates the type in the hierarchy where the member is stored.

LANG
Gives the language to associate with the output member. This can be used later if the member is the input to another translator.

Because the KEYREF parameter is OBJ, the FLMLTWST translator requires the ddname to be OBJ also or the OBJ parameter must be specified giving the ddname. For example, to use the ddname OBJBIN for outputs with a KEYREF of OBJ, you must specify "OBJ=OBJBIN" in the options string of the FLMLTWST translator.

FLMALLOC macro (ddname=list)
This is the allocation for the ddname to hold the .lst (listing) file that was generated on the workstation. This FLMALLOC has IOTYPE=O to allocate a sequential data set to hold the listing that will be stored back in the hierarchy. The PRINT parameter is also specified to initialize the data set and then copy it to the user's BUILD.LISTnn data set if needed. The IOTYPE=O or IOTYPE=P is needed because of the PRINT parameter.

FLMALLOC macro (ddname=userinfo)
This macro allocates the USERINFO data set. The FLMCPYLB macro that follows it allocates an existing data set to the ddname. The data set has the userid as the high-level qualifier, followed by SCLM.USERINFO. See “USERINFODD statements” on page 611 for information about the contents of this data set.

FLMALLOC macro (ddname=actinfo)
This is the allocation for the ACTINFO data set. The FLMCPYLB macro that follows it allocates an existing data set to the ddname. The data set has the project as the high-level qualifier, followed by "PROJDEFS.ACTINFO".

FLMALLOC macro (ddname=message)
This ddname stores messages from the translators that FLMLTWST calls. If the FLMTXFER translator fails, this is the first place to look.

FLMALLOC macro (ddname=msgxfer)
This ddname is used to transfer message files from the workstation to the host. After the messages are transferred to the host, they are appended to the messages ddname.

FLMALLOC macro (ddname=bmap)
This is the ddname where the FLMTBMAP translator writes the build information.

FLMALLOC macro (ddname=files)
This is the ddname to which FLMLTWST writes the list of files for FLMTXFER to transfer.

FLMALLOC macro (ddname=response)
This is the ddname where FLMLTWST generates the response file that is sent to the workstation. ACTION=COMPILE uses a response file; but if no response file is needed for the action, this ddname can be omitted.

Workstation setup

Workstation build expects the workstations to transfer files and issue commands in a consistent way. However, some information can vary from workstation to workstation. This information is contained in the user info data set allocated to the ddname that is specified by the USERINFO parameter when calling the FLMLTWST translator. See “USERINFODD statements” on page 611 for information about the contents of this data set.

Directories and file names

FLMLTWST constructs workstation file names from four components:

- The data directory is obtained from the userinfo data set (as specified by the DATA_DIR keyword). It can contain drive letters and whatever is necessary to establish the base path for the files and subdirectories.
- The subdirectory is obtained from the ACTINFO data set. The subdirectory is based on the type of the member. Subdirectories can be used to place different types of members in different directories for the workstation command or tool.
- The file name is the SCLM member name.
- The extension is obtained from the ACTINFO data set that maps SCLM types to extensions.
- The case (upper or lower) of the workstation file name is set based on the **WSCASE** value specified in the ACTINFO data set.

When SCLM constructs the full file name from the above components, it does not add or remove any characters from each of the components. Each component must be set up so that when it is combined with the others it will make a valid file name.

The FLMLTWST translator expects the data directory name not to end with a '/' or '\', but the subdirectory should start and end with these characters. The extension contains the '.' character.

Here are some examples of how FLMLTWST would put these four components together:

Table 25. Examples of FLMLTWST combining components

Data Directory	Subdirectory	File Name (Member)	Extension	Generated File Name
e:\temp	\	example1	.c	e:\temp\example1.c
e:	\temp\	example2	.h	e:\temp\example2.h
\temp	\bin\	example3	.exe	\temp\bin\example3.exe

The FLMLTWST translator does not clean out the directories after the workstation command is complete and the outputs have been transferred to the MVS system. The workstation owner must clean out the directories periodically to ensure that the workstation disks do not fill up.

Multiple builds on one workstation

SCLM supports using a single workstation for doing multiple builds either for a single user or multiple users. However, if the builds are taking place at different groups, either the base directory or the subdirectory must differ based on the group. This will avoid the problem of different builds overlaying one another's files.

One setup would have all builds at a specific group in the SCLM hierarchy occur on a specific workstation. In this case, the hierarchy view for all builds taking place on the workstation will be consistent so a single set of directories can be used or the directory names can vary based on the user performing the build.

Another setup would have a separate workstation for each user. In this case, either each user would need to ensure that all builds running concurrently are for the same group or the directory names would need to vary based on the group where the build is taking place.

Two methods to vary the directory name by the build group are:

- Include the @@FLMGRP variable in the FLMCPYLB allocation of the USERINFO data set. Then ensure that the USERINFO data sets that now include the group name in the data set name also vary the base directory based on the group name.
- Update the logic of FLMLTWST to accept a parameter with the group name where the build is taking place. Then generate the subdirectory based on the group. The language definition must set the group parameter to @@FLMGRP to pick up the build group.

Chapter 15. Leaving a Member Behind on Promotion

APAR OA18614 has introduced new functionality to allow a member to be left behind during promotion. Prior to this APAR being applied, SCLM copies all the required components to the next level.

You can specify whether an EDITABLE member and its accounting record are promoted to the next level or not. During promotion, the outputs (load modules) built using the non-promoted EDITABLE member are either:

- Rebuilt at the next level (REBUILD), or
- Not rebuilt at the next level

Note: The message FLM53109 "WARNING, PREDECESSOR VERIFICATION FAILED" occurs if the NOPROM member with the authorization code that is not defined to the target group cannot be promoted using one of the authorization codes defined to the from group. If this NOPROM member has an accounting status of NOPROM-R, a manual build must be performed at the target level, as the FLM53109 warning message ends the promote processing and the NOPROM rebuild step following the promote will not run.

Sample scenarios for each of the above are described below.

Load modules rebuilt at the next level (REBUILD)

Scenario: There are DB2 table changes in the development group required for the updating of the DB2 DCLGEN copybooks, but these DB2 DCLGEN copybooks are not to be promoted until the DB2 changes are complete. At this stage, you create a fix for a program which uses one of the DB2 DCLGEN copybooks but does not require the other DB2 development changes. This fix needs to be promoted to production.

When building at the development level, the DB2 copybook is used so the program can be tested. But when promoting the program fix, you want the DB2 copybook not to be promoted and the program to be rebuilt at the next level using the version of the DB2 DCLGEN copybook at that level or above.

Load modules not rebuilt at the next level

Scenario: A copybook is being modified in development and a fix for a single program which uses the copybook needs to be promoted to production. However, promoting this copybook would cause problems when building other programs using the copybook after promotion.

You want to build using the development version of the copybook but, when promoting the program fix, wants the copybook not to be promoted and the program source, load and so on, promoted as is.

Note: By promoting everything except the copybook and its accounting record, the build map containing the non-promoted member is in a broken state. This is because the build map date and time for the non-promoted member does not match the member account record and member statistics date and time.

Setting a member as not being promotable

To specify that a member is non-promotable, use one of the following methods:

- The N line command in Library Utilities (option 3.1) or Unit of Work (option 3.11)
- FLMCMD NOPROM service
- FLMLNK NOPROM service

You can specify that an EDITABLE member is:

© Copyright IBM Corp. 1990, 2019

Non-promotable but the build maps containing the non-promoted member are rebuilt after promotion

For the N line command, you specify "No promote (Rebuild)" on the SCLM non-promoted Member Update panel or, if using the FLMCMD/FLMLNK NOPROM service, you specify the REBUILD parameter. This sets the Account Status field in the accounting record to NOPROM-R.

The accounting status of NOPROM-R is used during build and promote to tell SCLM not to promote the member and ensure the build maps containing the member are rebuilt.

Non-promotable and the build maps containing the non-promoted member are not rebuilt after promotion

For the N line command, you specify "No promote (No Rebuild)" on the SCLM Not Promoted Member Update panel or, if using the FLMCMD/FLMLNK NOPROM service, you specify the NOREBUILD parameter. This sets the Account Status field in the accounting record to NOPROM-N.

The accounting status of NOPROM-N is used during build and promote to tell SCLM not to promote the member and ensure the build maps containing the member are copied as is.

Promotable

For the N line command, you specify "Remove no promote status" on the SCLM Not Promoted Member Update panel or, if using the FLMCMD/FLMLNK NOPROM service, you specify the REMOVE parameter. This sets the Account Status field in the accounting record to EDITABLE.

You can only issue this option against a member with an Account Status of NOPROM-R or NOPROM-N.

Using the N line command in Library Utilities (option 3.1) or Unit of Work (option 3.11)

You can only issue the N Line command in the Library Utilities (option 3.1) or Unit of Work (option 3.11) against a member with an accounting record that has an accounting status of EDITABLE, NOPROM-N, or NOPROM-R.

The N line command displays the SCLM Not Promoted Member Update panel.

```
  Menu  SCLM  Utilities  Help
 ──────────────────────────────────────────────────────────────────────

                SCLM Not Promoted Member Update
Command ===> ____________________________________________________

   SCLM Library:
   PROJECT  . . . . : SLMTEST6
   GROUP  . . . . . : DEV1
   TYPE . . . . . . : SOURCE
   MEMBER . . . . . : FLM01EQU

   Options
   NOPROM:  1  1. No promote (Rebuild)
               2. No promote (No Rebuild)
               3. Remove no promote status

 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward
 F9=Swap     F10=Actions  F12=Cancel
```

Figure 167. SCLM Not Promoted Member Update panel (FLMUSN#P)

When you press Enter, SCLM changes the Account Status field to one of the following values:

- NOPROM-R, if you specify the "No promote (Rebuild)" option.
- NOPROM-N, if you specify the "No promote (No Rebuild)" option.
- EDITABLE, if you specify the "Remove no promote status" option *and* the account status is currently NOPROM-N or NOPROM-R.

FLMCMD NOPROM service

The following example shows invoking the FLMCMD NOPROM service to set the member FLM01EQU as non-promotable. Using the REBUILD parameter causes all the build maps containing FLM01EQU to be rebuilt when promoted.

For more information, see the NOPROM service in Chapter 15.

```
/*REXX*/
/*--------------------------------------------------------------------*/
/*                                                                    */
/* FLMCMD NOPROM command                                              */
/*                                                                    */
/*--------------------------------------------------------------------*/
trace n
 /* allocate FLMMSG DSN */
 PARSE ARG PROJECT ALTPROJ GROUP TYPE MEMBER SVCPARM

ADDRESS TSO
msgdsn = "'"||USERID()||".TEMP.FLMMSG'"
X=MSG('OFF')
IF SYSDSN(msgdsn) <> "OK" THEN DO
   "ALLOC FI(AAAMSGS) DATASET("msgdsn") ",
        "UNIT(SYSDA) TRACKS SPACE(5 5) ",
        "LRECL(80) BLKSIZE(3120) RECFM(F B) DSORG(PS) ",
        "NEW CATALOG "
   "FREE FI(AAATEMP)"
END
"FREE FI(FLMMSGS) "
X=MSG('ON')
"ALLOC FI(FLMMSGS) DATASET("msgdsn") SHR REUSE"

PROJECT="SCLMPROJ"
ALTPROJ="SCLMPROJ"
GROUP  ="DEV1"
TYPE   ="SOURCE"
MEMBER ="FLM01EQU"
SVCPARM="REBUILD"

"FLMCMD NOPROM,"PROJECT","ALTPROJ","GROUP","TYPE","MEMBER",",
         ","SVCPARM",FLMMSGS"

 "ISPEXEC BROWSE DATASET("msgdsn")"
return
```

FLMLNK NOPROM service

The following example shows invoking the FLMLNK NOPROM service to set the member FLM01EQU as non-promotable. Using the NOREBUILD parameter causes SCLM to promote everything except the FLM01EQU member and its accounting record.

For more information, see the NOPROM service in Chapter 15.

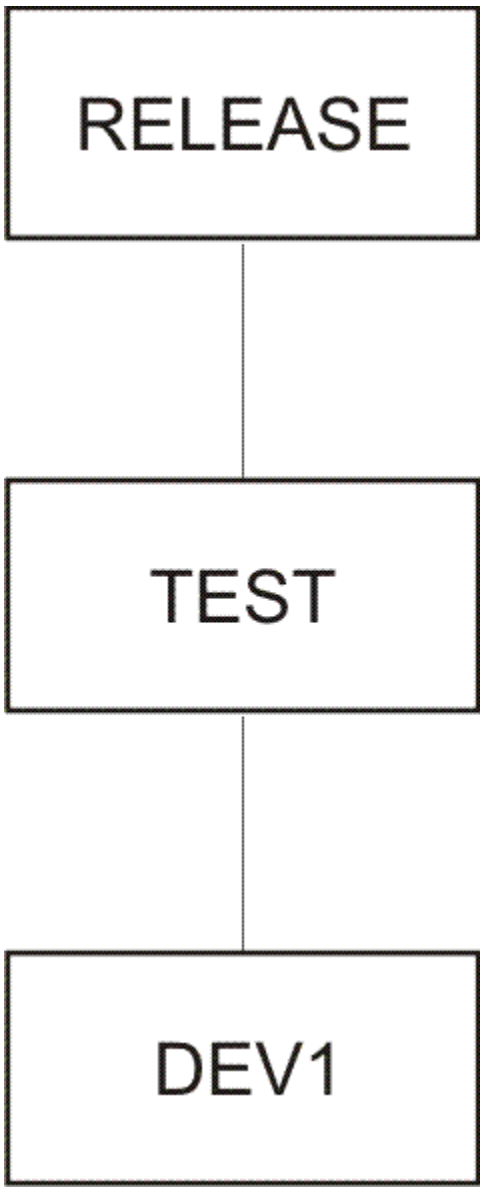
```
IDENTIFICATION DIVISION.
 PROGRAM-ID.   FLM02CBL.
*@@@
```

```
 77  SCLM-APPL-ID       PIC X(08) VALUE SPACE.
 77  SCLM-AUTHCODE     PIC X(08) VALUE SPACE.
 77  SCLM-NOPROM-FIELD  PIC X(16).
 77  SCLM-ACCESS-KEY    PIC X(16) VALUE SPACE.
 77  SCLM-ACCT-INFO     PIC S9(04) COMP VALUE ZERO.
 77  SCLM-LIST-INFO     PIC S9(04) COMP VALUE ZERO.
 77  SCLM-MSG-ARRAY     PIC S9(04) COMP VALUE ZERO.
 77  SCLM-DD-PRSLIST    PIC X(08) VALUE 'SYSOUT'.
 77  SCLM-DD-MSGS       PIC X(08) VALUE 'MDS3602M'.
 77  SCLM-DD-EXIT       PIC X(08) VALUE SPACE.
 77  SCLM-MSG-LINE      PIC X(80) VALUE SPACE.
*
 LINKAGE SECTION.
PROCEDURE DIVISION.
1-DRIVER.
*-------------------------------------------------------
* FLMLNK NOPROM,JPHILP,OS2G,DEV1,SOURCE,FLM01EQU,NOREBUILD
*-------------------------------------------------------
     MOVE 'SCLMPROJ'      TO SCLM-PROJECT
     MOVE 'SCLMPROJ'      TO SCLM-ALT-PROJ
     MOVE 'DEV1'          TO SCLM-GROUP
     MOVE 'SOURCE'        TO SCLM-TYPE
     MOVE 'FLM01EQU'      TO SCLM-MEMBER
     MOVE 'NOREBUILD'     TO SCLM-NOPROM-FIELD

     MOVE 'START'     TO SCLM-SERVICE
     CALL 'FLMLNK' USING SCLM-SERVICE
                         SCLM-APPL-ID.
     DISPLAY 'START RETURN CODE =  ' RETURN-CODE.
     MOVE 'INIT'      TO SCLM-SERVICE
     CALL 'FLMLNK' USING SCLM-SERVICE
                         SCLM-APPL-ID
                         SCLM-PROJECT
                         SCLM-ALT-PROJ
                         SCLM-SCLM-ID
                         SCLM-MSG-LINE.
     DISPLAY 'INIT  RETURN CODE =  ' RETURN-CODE.

     MOVE 'NOPROM'    TO SCLM-SERVICE
     CALL 'FLMLNK' USING SCLM-SERVICE
                         SCLM-SCLM-ID
                         SCLM-GROUP
                         SCLM-TYPE
                         SCLM-MEMBER
                         SCLM-NOPROM-FIELD
                         SCLM-ACCESS-KEY
                         SCLM-MSG-LINE.
     DISPLAY 'NOPROM RETURN CODE =  ' RETURN-CODE.
     IF RETURN-CODE > 0 THEN
         DISPLAY SCLM-MSG-LINE.
     MOVE 'FREE'      TO SCLM-SERVICE
     CALL 'FLMLNK' USING SCLM-SERVICE
                         SCLM-SCLM-ID
                         SCLM-MSG-LINE.
     DISPLAY 'FREE  RETURN CODE =  ' RETURN-CODE.

     MOVE 'END'       TO SCLM-SERVICE
     CALL 'FLMLNK' USING SCLM-SERVICE
                         SCLM-APPL-ID
                         SCLM-MSG-LINE.
     DISPLAY 'END   RETURN CODE =  ' RETURN-CODE.
     DISPLAY ' '.
     DISPLAY 'NOPROM COMPLETED'.
     DISPLAY 'LOAD FLM02CBL ENDED'.

     GOBACK.
 1-EXIT. EXIT.
```

Process of not promoting a member (REBUILD)

This section describes the process of not promoting a member and causing a REBUILD of the load modules which use it at the next level.

For the following description of this project the project will have the following structure.



Scenario: The DEV1 group contains a DB2 DCLGEN copybook SCLMTB01 for a DB2 table which is in the process of being modified. You want to build and test in DEV1 using this copybook but do not want SCLMTB01 promoted until it is ready.

Using the new functionality the member can be modified so it is not copied to the next level during promotion. Any load modules which use SCLMTB01 are rebuilt after promotion to the next level (TEST) to use the version of the copybook at TEST or above.

To do this, you can use either the N line command in the Library Utility (option 3.1) or Unit of Work (option 3.11) or the FLMCMD/FLMLNK NOPROM service. For this example, we will use the N line command in the Library Utility option. By issuing the N line command, SCLM invokes the following panel.

```
  Menu  SCLM  Utilities  Help
 ──────────────────────────────────────────────────────────────────────────

               SCLM Not Promoted Member Update
Command ===> ____________________________________________________

   SCLM Library:
   PROJECT  . . . . : SLMTEST6
   GROUP  . . . . . : DEV1
   TYPE . . . . . . : COPYLIB
   MEMBER . . . . . : SCLMTB01

   Options
   NOPROM:  1  1. No promote (Rebuild)
               2. No promote (No Rebuild)
               3. Remove no promote status

 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward
 F9=Swap     F10=Actions  F12=Cancel
```

Figure 168. SCLM Not Promoted Member Update panel (FLMUSN#P)

By specifying "No promote (Rebuild)", SCLM modifies the accounting record for SCLMTB01 to have an account status of NOPROM-R.

```
          SLMTEST6.DEV1.COPYLIB(SCLMTB01): Accounting Record
Command ===> ____________________________________________________


  Physical Data Set . : SLMTEST6.DEV1.COPYLIB
  Accounting Status . : NOPROM-R        Change Group . . . . : DEV1
  Change User ID  . . : JPHILP          Authorization Code . : P
  Member Version  . . : 2               Auth. Code Change  . :
  Language  . . . . . : TEXT            Translator Version . :
  Creation Date . . . : 2007/01/08      Change Date  . . . . : 2007/01/16
  Creation Time . . . : 16:55:08        Change Time  . . . . : 09:51:05
  Promote User ID . . :                 Access Key . . . . . :
  Promote Date  . . . : 0000/00/00      Build Map Name . . . :
  Promote Time  . . . : 00:00:00        Build Map Type . . . :
  Predecessor Date  . : 2007/01/08      Build Map Date . . . : 2007/01/16
  Predecessor Time  . : 16:56:22        Build Map Time . . . : 09:51:05

 Enter "/" to select option
   Display Statistics
   Number of Change Codes      : 0
   Number of Includes          : 0
   Number of Includes          : 0
   Number of Compilation Units : 0
   Number of User Entries      : 0
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F12=Cancel
```

While building the record, SCLM analyzes the components and determines that the member SCLMTB01 has an account status of NOPROM-R. Once the build has completed, SCLM updates the build maps containing the SCLMTB01 member to have a NOPROM build map record. The build report shows the build maps that were updated.

```
          ******* N O P R O M   M E M B E R S            *******  Page 5

BUILD     BMAP      NOPROM    NOPROM MBR
MAP       TYPE      MEMBER    TYPE
--------  --------  --------  ----------
SCLMDB2   SOURCE    SCLMTB01  COPYLIB

****************************** Bottom of Data ******************************
```

The build map for the SCLMDB2 COBOL member looks as follows.

```
          SLMTEST6.DEV1.SOURCE(SCLMDB2): Build Map Contents        Line
Command ===> ____________________________________________________ Scroll ===> CSR
                              Build Map Contents
                              ------------------

  Keyword  Member                                  Type     Last Time Modified  Ver
  -------- --------------------------------------- -------- ------------------- ---
  SINC     SCLMDB2                                 SOURCE   2007/01/25 12:06:52 6
  OBJ      SCLMDB2                                 OBJ      2007/01/25 12:06:55 6
  LIST     SCLMDB2                                 LIST     2007/01/25 12:06:55 6
  OUT1     SCLMDB2                                 DBRM     2007/01/25 12:06:55 6
  I1*      COBCOPY1                                COPYLIB  2007/01/08 16:57:59 1
  NOPROM   SCLMTB01                                COPYLIB
  I1*      SCLMTB01                                COPYLIB  2007/01/16 09:51:05 2
  I1*      SCLMTB02                                COPYLIB  2007/01/16 09:51:12 3

  * Internal Keywords
    I#     - Included member referenced by SINC member, # = Imbedded Group
    NOPROM - Member was/will be left behind on promotion.
  ****************************** Bottom of Data ******************************


 F1=Help      F2=Split     F3=Exit      F4=retrieve  F5=Rfind     F7=Up
 F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

The NOPROM build map record is used during promote to indicate that the promotion contains members that are not to be copied to the next level.

When promoting SCLMDB2 from DEV1 to TEST, SCLM does the following actions:

- Encounters the NOPROM map record on the build map SCLMDB2.
- Compares the group specified on the NOPROM map record (DEV1) and, as it is same as the group being promoted from (DEV1), the normal date and time validation of the non-promoted member in the build maps takes place.
- Verifies that the SCLM components are current by comparing the build maps, accounting records, and member statistics.
- While performing this verification, SCLM checks the accounting record for the accounting status for the SCLMTB01 member. As it is set to NOPROM-R, SCLM does not copy to the next level the member SCLMTB01, its accounting record, or any of the build maps containing the NOPROM member SCLMTB01.
- When copying components to the next level, SCLM does not copy the SCLMTB01 member, its accounting record, or any of the build maps (that is, SCLMDB2) containing the SCLMTB01 member.
- Once the copy phase is complete, SCLM invokes a build at the TEST level. As the build maps containing the SCLMTB01 member were not promoted, these are rebuilt at the TEST level using the version of SCLMTB01 at the TEST level or above.

The promote report shows the non-promoted members and the build maps that were affected by the non-promoted member.

```
********************************************************************************
**                                                                            **
**                N O T   P R O M O T E D   M E M B E R S                     **
**                                                                            **
********************************************************************************
BACKUP    NOT PROMOTED MEMBER    TYPE      MEMBER    OPTION ________  ________
________  ________

SCLMTB01  COPYLIB             REBUILD

                                                                    PAGE   10
********************************************************************************
**                                                                            **
**              BUILD MAPS AFFECTED BY NOT PROMOTED MEMBERS                   **
**                                                                            **
********************************************************************************
BUILD     BMAP      BMAP      NOPROM    NOPROM MBR MAP       TYPE      REBUILT
MEMBER    TYPE ________  ________  ________  ________  __________

SCLMDB2   SOURCE    YES       SCLMTB01  COPYLIB

****************************** Bottom of Data ******************************
```

Once the promote is complete, when you view the SCLMDB2 build map you can see the SCLMTB01 member used to build SCLMDB2 is dated 2007/01/08 16:56, not 2007/01/16 09:51. This shows that SCLMDB2 was built using an earlier version of the copybook SCLMTB01 at the TEST level, or higher.

```
          SLMTEST6.TEST.SOURCE(SCLMDB2): Build Map Contents        Line
 Command ===> ____________________________________________________ Scroll ===> CSR
                              Build Map Contents
                              ------------------

   Keyword  Member                                Type     Last Time Modified
Ver
   -------- ------------------------------------ -------- -------------------
---
   SINC     SCLMDB2                              SOURCE   2007/01/25 12:06:52 6
   OBJ      SCLMDB2                              OBJ      2007/01/25 12:10:22 7
   LIST     SCLMDB2                              LIST     2007/01/25 12:10:22 7
   OUT1     SCLMDB2                              DBRM     2007/01/25 12:10:22 7
   I1*      COBCOPY1                             COPYLIB  2007/01/08 16:57:59 1
   I1*      SCLMTB01                             COPYLIB  2007/01/08 16:56:22 1
   I1*      SCLMTB02                             COPYLIB  2007/01/16 09:51:12 3

   * Internal Keywords
     I#     - Included member referenced by SINC member, # = Imbedded Group
   ****************************** Bottom of Data
********************************



  F1=Help      F2=Split     F3=Exit      F4=retrieve  F5=Rfind     F7=Up
  F8=Down      F9=Swap     F10=Left     F11=Right    F12=Cancel
```

Process of not promoting a member (NOREBUILD)

This section describes the process of not promoting a member but promoting the load modules and related members (NOREBUILD).

Note: Prior to this new functionality, promotion of SCLM members left the SCLM environment in a complete state where the dates and times in the accounting records, member statistics, and the build maps matched. Performing a build at this level does not result in any members being built.

The new functionality of promoting everything except a member and its accounting record (No Promote - No Rebuild) results in SCLM being in a broken state. The build maps containing the non-promoted member have dates and times for the member which was not promoted. This means that SCLM needs to cater for several issues when promoting and building using the members with an accounting status of NOPROM-N. These issues are:

- **Need to backup the non-promoted member.** Since you are promoting the build maps that were built using the non-promoted member, you need to take a backup of this member. This ensures that, even if the member has been modified or deleted after you promoted, you can still recreate the outputs (load modules).
- **SCLM needs to be able to promote build maps at a level where a member was left behind.** After promotion, where a member was left behind but the build maps were not rebuilt, the level you promoted into is in a broken state. Normally, when attempting to promote these changes further, SCLM gave you date and time inconsistencies between the build map and accounting record for the non-promoted members.

  For the new functionality, SCLM has been changed so that when promoting it ignores the date and time inconsistencies between the build map, accounting record, and member statistics if:

  – The build map contains a NOPROM build map record, and
  – The build level (not viewable) found on the NOPROM build map record does not match the current build level
- **SCLM needs to be able to build at a level where a member was left behind.** After promotion, where a member was left behind but the build maps were not rebuilt, the level you promoted into is in a broken state. Normally, when attempting to build at this level, it causes the build maps with date and time inconsistencies (due to left-behind members) to be rebuilt.

  For the new functionality, SCLM has been changed so that when building it ignores the date and time inconsistencies between the build map, accounting record, and member statistics if:

  – The build map contains a NOPROM build map record, and
  – The build level (not viewable) found on the NOPROM build map record does not match the current promote from level

SCLM project setup when promoting with no rebuilding of build maps

To allow SCLM to backup the non-promoted members, you must perform the following actions:

- Allocate a NOPROM backup data set. This data set is a fixed block data set with a LRECL of 1024.
- Modify the SCLM project definition to add a NPROMBK parameter on the FLMCNTRL macro to specify the NOPROM backup data set name.
- Allocate a CONTROL VSAM data set (if not already defined). The sample JCL for defining this member can be located in ISP.SISPSAMP(FLM02CNT).
- Modify the SCLM project definition to add in a CONTROL parameter on the FLMCNTRL macro to specify the CONTROL VSAM data set name.

If the above steps have not been performed, SCLM still completes a promotion containing the non-promoted member; however, no backups are taken. To recreate the outputs (for example, the Load module) based on the left-behind member, you must use the left-behind member. If this member was modified or deleted after promotion, it may not be possible to recreate the SCLM outputs.

Build containing a non-promotable member (NOREBUILD)

The following example is based on the SCLM sample project.

Scenario: A copybook FLM01EQU is being modified in development and a fix for a load module FLM01LD4 which uses the copybook needs to be promoted to production. However promoting the copybook FLM01EQU would cause problems when building other load modules (for example, FLM01LD3) using the copybook after promotion.

You therefore want to build using the development version of the copybook FLM01EQU, but when promoting the load module FLM01LD4 you do not want the copybook copied to the next level (TEST).

To set up FLM01EQU so that it is not promoted and that the load module FLM01LD4 is not rebuilt, you can issue either the FLMCMD/FLMLNK NOPROM service or the N line command in the Library Utility (option 3.1) or Unit of Work (option 3.11). For this example, we will use the N line command in the Library Utility. By issuing the N line command, SCLM invokes the following panel.

```
  Menu  SCLM  Utilities  Help
 ──────────────────────────────────────────────────────────────────────

                SCLM Not Promoted Member Update
Command ===> ____________________________________________________

   SCLM Library:
   PROJECT  . . . . : SLMTEST6
   GROUP  . . . . . : DEV1
   TYPE . . . . . . : SOURCE
   MEMBER . . . . . : FLM01EQU

   Options
   NOPROM:  2  1. No promote (Rebuild)
               2. No promote (No Rebuild)
               3. Remove no promote status

 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward
 F9=Swap     F10=Actions  F12=Cancel
```

By specifying "No promote (No Rebuild)", SCLM modifies the accounting record for FLM01EQU to have an account status of NOPROM-N.

```
          SLMTEST6.DEV1.SOURCE(FLM01EQU): Accounting Record
 Command ===> ____________________________________________________

   Physical Data Set . : SLMTEST6.DEV1.SOURCE
   Accounting Status . : NOPROM-N        Change Group . . . . : DEV1
   Change User ID  . . : JPHILP          Authorization Code . : P
   Member Version  . . : 2               Auth. Code Change  . :
   Language  . . . . . : HLAS            Translator Version . :
   Creation Date . . . : 2001/10/30      Change Date  . . . . : 2007/01/25
   Creation Time . . . : 16:58:56        Change Time  . . . . : 12:14:38
   Promote User ID . . :                 Access Key . . . . . :
   Promote Date  . . . : 0000/00/00      Build Map Name . . . :
   Promote Time  . . . : 00:00:00        Build Map Type . . . :
   Predecessor Date  . : 2001/10/30      Build Map Date . . . : 2007/01/25
   Predecessor Time  . : 16:58:56        Build Map Time . . . : 12:14:38

  Enter "/" to select option
    Display Statistics
    Number of Change Codes      : 1
    Number of Includes          : 0
    Number of Compilation Units : 0
    Number of User Entries      : 1
  F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
 F12=Cancel
```

While building the record, SCLM analyzes the components to be built and determines that the member FLM01EQU has an account status of NOPROM-N. Once the build has completed for FLM01LD4, SCLM updates the build maps containing the FLM01EQU member to have a NOPROM build map record.

The build report shows the build maps that were updated.

```
          ******* N O P R O M   M E M B E R S             *******  Page 5

BUILD     BMAP      NOPROM    NOPROM MBR
MAP       TYPE      MEMBER    TYPE
--------  --------  --------  ----------
FLM01MD4  SOURCE    FLM01EQU  SOURCE
FLM01MD5  SOURCE    FLM01EQU  SOURCE
FLM01MD6  SOURCE    FLM01EQU  SOURCE

******************************** Bottom of Data ************************
```

The build map for the FLM01MD4 member looks as follows.

```
Work Element List for UOW  FLM01AP1 in SLMTEST6                Member 1 of 2
Command ===> ____________________________________________ Scroll ===> CSR

S=Sel/Edit  A=Account   M=Map       B=Browse    D=Delete    E=Edit      V=View
C=Build     P=Promote   U=Update    T=Transfer  N=NOPROM
Z=Versions
    Member   Type     Status      Account  Language Chg Date         User
    FLM01AP1 ARCHDEF              (Current UOW ARCHDEF)
    FLM01SB1 ARCHDEF              RELEASE  ARCHDEF  2001/10/30 16:59 JPHILP
    FLM01SB2 ARCHDEF              RELEASE  ARCHDEF  2001/10/30 16:59 JPHILP
****************************** Bottom of data ********************************
```

The NOPROM build map record in the build maps for FLM01MD4, FLM01MD5, and FLM01MD6 is used during promote to indicate that the promotion contains members that are not to be promoted.

Promote containing a non-promotable member (NOREBUILD) from the same level containing the NOPROM member

After the member has been specified as being non-promotable (NOREBUILD) and SCLM has performed a build, these changes can then be promoted to the next level (TEST).

When promoting FLM01LD4 from DEV1 to TEST, SCLM does the following actions:

- Encounters the NOPROM map record on the build maps FLM01MD4, FLM01MD5, and FML01MD6.
- Compares the group specified on the NOPROM map record (DEV1) and, as it is same as the group we are promoting from (DEV1), the normal date and time validation of the non-promoted member in the build maps takes place.
- Verifies that the SCLM components are current by comparing the build maps, accounting records, and member statistics.
- While performing this verification, SCLM checks the accounting record for the accounting status for the FLM01EQU member. As it is set to NOPROM-N, all the build maps containing the NOPROM member FLM01EQU are copied to the next level.
- Reads the CONTROL file to determine the name of the backup member. To do this, SCLM reads the CONTROL file to determine the NOPROM backup number; this is used to generate the backup member name. If the backup number is 00000001, then the backup member is A0000001.
- Backs up the non-promoted member FLM01EQU into the NOPROM PDS specified by the NPROMBK parameter on the FLMCNTRL macro in the project definition.
- Updates the build maps for FLM01MD4, FLM01MD5, and FML01MD6 at the DEV1 level to add in the backup member name generated above (A0000001) into the NOPROM build map record.
- When copying components to the next level, the FLM01EQU member and its accounting record are not promoted, but build maps containing the FLM01EQU member are promoted.

Viewing the non-promoted backup member

A build map containing NOPROM build map for which a backup has been taken looks as follows.

```
          SLMTEST6.DEV1.SOURCE(FLM01MD4): Build Map Contents     Line
Command ===> ____________________________________________ Scroll ===> CSR
                              Build Map Contents
                              ------------------

  Keyword  Member                                Type     Last Time Modified  Ver
  -------- ------------------------------------ -------- ------------------- ---
  SINC     FLM01MD4                             SOURCE   2001/10/30 16:58:57 1
  OBJ      FLM01MD4                             OBJ      2007/01/25 12:19:00 2
  LIST     FLM01MD4                             SOURCLST 2007/01/25 12:19:00 2
_ NOPROM   FLM01EQU                             SOURCE
  I1*      FLM01EQU                             SOURCE   2007/01/25 12:14:38 2

  * Internal Keywords
    I#     - Included member referenced by SINC member, # = Imbedded Group
    NOPROM - Member was/will be left behind on promotion.
             Use the S line command to view the not promoted member.

  ****************************** Bottom of Data ********************************
```

By typing an S alongside the NOPROM member, you can view the backup of the member that was taken as a part of the promotion of this build map.

Promote containing a non-promotable member (NOREBUILD) from a level not containing the NOPROM member

Scenario: You have tested the changes at TEST and want to promote the changes to the RELEASE level. However the SCLM components at TEST are in a broken state as build maps FLM01MD4, FLM01MD5, and FLM01MD6 at TEST contain the member FLM01EQU, but this member was not promoted from DEV1.

When promoting from TEST to RELEASE, SCLM does the following actions:

- Encounters the NOPROM map record in the build maps build maps FLM01MD4, FLM01MD5, and FLM01MD6. The FLM01EQU member is added to the list of members that are not being promoted.
- Compares the group specified on the NOPROM map record (DEV1) and, as it is not equal to the group being promoted from (TEST):
  - Skips the date and time validation of the non-promoted member, FLM01EQU
  - Does not backup the non-promoted member, FLM01EQU
- Verifies that the SCLM components are current by comparing the build maps, accounting records, and member statistics. If SCLM encounters the member FLM01EQU, it is flagged as being up to date.
- When copying components to the next level, the FLM01EQU member and its accounting record are not promoted, but build maps containing the FLM01EQU member are promoted.

Build containing a non-promotable member (NOREBUILD) at a level which does not contain the NOPROM member

After the changes have been promoted to RELEASE, the SCLM components are in a broken state as build maps FLM01MD4, FLM01MD5, and FLM01MD6 contain the member FLM01EQU, but this member was not promoted from DEV1.

Note: Prior to this new functionality, a build perform against FLM01LD4 rebuilt the load module due to the broken state of the build maps FLM01MD4, FLM01MD5, and FLM01MD6. This is not acceptable as it would override the changes you have just promoted.

When performing a build containing a NOPROM build map record where the level specified on the build maps record does match the build level, SCLM performs the following actions:

- When analyzing the SCLM components, encounters the NOPROM map record on the build maps FLM01MD4, FLM01MD5 and FML01MD6.

For each of these build maps, SCLM performs the following actions:

- Compares the group specified on the NOPROM map record (DEV1) to see if it is different to the group you are building at. If it is different, and the date and time of the accounting record does not match the date and time in the build maps, then SCLM knows that the member FLM01EQU has not been promoted.
- As the FLM01EQU member has not been promoted, it is set as "up to date" in the build map and SCLM issues message FLM44523.
- If other changes are encountered which would cause the build map to be rebuilt, SCLM issues message FLM44522 and completes with a return code of 8.

  The above situation occurs if a member used by the build map was changed and promoted but the build map was not rebuilt. To resolve this problem, you need to do one of the following actions:

  – Delete the build map causing the problem and rebuild. This causes the build map to be rebuilt using the version of the non-promoted member at the build level or above (that is, RELEASE).
  – Rebuild the build map at the group containing the non-promoted member (DEV1) and re-promote your changes. This rebuilds the build map using the members at the higher level, as well as the non-promoted member.

Build after promotion of the non-promotable member (NOREBUILD)

If the member FLM01EQU is ready to be promoted into RELEASE, to remove the non-promotable status of the member FLM01EQU, you can use the N line command in Library Utilities (option 3.1) or Unit of Work (Option 3.11), or run the FLMCMD/FLMLNK NOPROM service. Once this is complete, the accounting record for the member FLM01EQU looks as follows.

```
          SLMTEST6.DEV1.SOURCE(FLM01EQU): Accounting Record
Command ===> ____________________________________________________________

  Physical Data Set . : SLMTEST6.DEV1.SOURCE
  Accounting Status . : EDITABLE        Change Group . . . . : DEV1
  Change User ID  . . : JPHILP          Authorization Code . : P
  Member Version  . . : 2               Auth. Code Change  . :
  Language  . . . . . : HLAS            Translator Version . :
  Creation Date . . . : 2001/10/30      Change Date  . . . . : 2007/01/25
  Creation Time . . . : 16:58:56        Change Time  . . . . : 12:14:38
  Promote User ID . . :                 Access Key . . . . . :
  Promote Date  . . . : 0000/00/00      Build Map Name . . . :
  Promote Time  . . . : 00:00:00        Build Map Type . . . :
  Predecessor Date  . : 2001/10/30      Build Map Date . . . : 2007/01/25
  Predecessor Time  . : 16:58:56        Build Map Time . . . : 12:14:38

 Enter "/" to select option
   Display Statistics
   Number of Change Codes      : 1
   Number of Includes          : 0
   Number of Compilation Units : 0
   Number of Compilation Units : 0
   Number of User Entries      : 0
 F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
F12=Cancel
```

By promoting the member FLM01EQU, or a build map containing FLM01EQU, SCLM invokes a promotion process that copies the member FLM01EQU and its accounting record to the next level. Once the member FLM01EQU has been promoted to the RELEASE level, if a rebuild of the build maps FLM01MD4, FLM01MD5, and FLM01MD6 has not occurred, these build maps still contain the NOPROM build map records.

When building at the RELEASE level after promotion of FLM01EQU, SCLM does the following actions:

- Encounters the NOPROM map record on the build maps FLM01MD4, FLM01MD5, and FLM01MD6.
- Compares the group specified on the NOPROM map record (DEV1) to see if it is different to the level at which you are building. If it is different, and the date and time of the accounting record matches the date and time in the build maps, then SCLM knows that the member FLM01EQU has been promoted.
- Removes the NOPROM build map record entries from the build maps FLM01MD4, FLM01MD5, and FLM01MD6, even if they were not re-built.

Restricting the setting of non-promotable

You can restrict the use of the N line command in the Library Utility (option 3.1) or Unit of Work (option 3.11), or the FLMCMD/FLMLNK NOPROM service, by the use of the FLMNPROM macro within the SCLM project definition.

If FLMNPROM has not been specified in the SCLM project definition, there are no restrictions on the use of the NOPROM service or the N line command in Library Utility (option 3.1) or Unit of Work (Option 3.11) within the SCLM project.

By specifying the FLMNPROM macro, you can specify which groups, types, and languages can be set as non-promotable.

Examples

The following example shows how to specify that editable members with a language of COBCOPY in the SOURCE type, in any group, can be marked as non-promotable:

```
FLMNPROM GROUP=*,TYPE=SOURCE,LANG=COBCOPY,NPROM=YES
```

The following example shows how to specify that all editable members with a language of COBCOPY in the SOURCE type, in all groups except EMERFIX, can be marked as non-promotable:

```
FLMNPROM GROUP=*,TYPE=SOURCE,LANG=COBCOPY,NPROM=YES
FLMNPROM GROUP=EMERFIX,TYPE=SOURCE,LANG=COBCOPY,NPROM=NO
```

Chapter 16. Member encoding and decoding

SCLM allows you to encode:

Editable members
The language associated with the member is used to determine if the member is to be encoded.

When the editable member is saved in Edit, or using the MIGRATE, LOCK/PARSE/STORE or SAVE SCLM services if the member's associated language has ENCODE=Y specified on the FLMLANGL macro, then the member is encoded.

SCLM build outputs (Non-editable members)
The FLMALLOC macro in the Build translator determines if the build output is to be encoded.

If the ENCODE=Y parameter is specified in a build translator on a FLMALLOC macro with IOTYPE=O or P, then during the build process when SCLM copies the temporary data set and members to the SCLM controlled libraries, the members are encoded.

These areas in SCLM allow SCLM to handle encoding and decoding of members:

- Edit
- View/Browse
- Build
- Versioning
- SCLM Services

Edit
If a member has a language with ENCODE=Y specified on the FLMLANGL macro, then when entering edit, SCLM attempts to decode the member into a temporary data set. If the member is not encoded, normal editing of the member occurs.

Upon exiting the edit session, if the member has been modified, the member is saved and encoded if the members language specifies ENCODE=Y on the FLMLANGL macro. The following edit commands have been created and modified to allow SCLM to handle encoded members:

SCREATE
If the member currently being edited is encoded, then the new member that is created is encoded when saved.

SCOMPARE
Checks to see if the data set and member to be compared is encoded. If it is, SCLM decodes it into a temporary data set prior to it being compared to the member being edited.

Note: SCOMPARE determines where the member exists in the SCLM hierarchy, rather than use the allocated concatenated data set prior to it being compared to the member being edited.

SCOPY
Checks to see if a member is encoded. If it is, SCLM decodes the member prior to it being copied into the edit session.

SMOVE
Checks to see if a member is encoded. If it is, SCLM decodes the member prior to it being moved into the edit session.

SREPLACE
If the member currently being edited is encoded, then the new member to be replaced is encoded when it is saved.

Browse and View
If an editable member has a language with ENCODE=Y specified on the FLMLANGL macro, when entering browse and view, SCLM attempts to decode the member into a temporary data set. If the member is not encoded, normal browsing and viewing of the member will occur.

© Copyright IBM Corp. 1990, 2019

For non-editable members, the member will be decoded if the SCLM project contains a build translator with the ENCODE=Y parameter specified on the FLMALLOC macro.

Build

The build process handles encoded editable members and creates encoded outputs which have been specified using the ENCODE=Y parameter on the FLMALLOC macro.

When building, SCLM performs the following actions:

- Determines if the input member (IOTYPE=S) is encoded. If it is, the member will be decoded into a temporary data set which is used by the translator.
- Determines the input members (IOTYPE=I) that are required by the build translator. This list of members is used to determine if they are to be decoded. The processing of these input member will be different depending on whether CCODE processing is being used or not.

For CCODE build processing SCLM:

- Allocates a temporary to contain the CCODE members.
- Decodes any CCODE members, which are encoded, into the CCODE temporary data set. The remaining CCODE members will be copied into the CCODE temporary data set.
- Allocates an ENCODE temporary data set to contain any encoded input members not processed into the CCODE temporary data set.
- Decodes any non-processed input members, which are encoded, into the ENCODE temporary data set.
- Concatenates both of the temporary data sets with the required hierarchical data sets.

For normal build processing (not CCODE processing), SCLM allocates a temporary data set into which it decodes the encoded input members. This temporary data set is concatenated with the required hierarchical data sets.

Once the build process is complete SCLM copies the temporary data sets/members with IOTYPE=O and P into the SCLM controlled libraries, As part of this process, if the FLMALLOC macro with IOTYPE=O and P has an ENCODE=Y specified, the members being copied into the SCLM controlled libraries will be encoded.

Versioning

If a member has a language with ENCODE=Y specified on the FLMLANGL macro and either the group you are saving the member to, or the group you are promoting into, has versioning active, SCLM decodes and encodes the version PDS member when it is being updated.

To create and encoded version PDS member SCLM:

- Decodes the version PDS member (if required).
- Decodes the new source member (if required).
- Creates the new version PDS member.
- When saving the new version PDS member, SCLM encodes the member.

The Audit and Version Utility panels, as well as the VERDEL and VERRECOV SCLM services have been modified to decode and encode the version PDS member if the member being processed has a language with ENCODE=Y specified on the FLMLANGL macro.

SCLM services

The SCLM services handle encoding and decoding of the SCLM members. The services that have been created or modified are:

ENDEC
: A new SCLM service that ENCODES or DECODES SCLM members

MIGRATE
: Decodes a member if it has a language which SPECIFIES ENCODE=Y to allow it to be parsed by SCLM. If the member is to be saved and has a language with ENCODE=Y, SCLM encodes the member.

PARSE
Decodes a member if it has a language which specifies ENCODE=Y to allow it to be parsed by SCLM.

STORE
If the member has a language with ENCODE=Y, SCLM encodes the member when it is saved.

SAVE
Decodes a member if it has a language which specifies ENCODE=Y to allow it to be parsed by SCLM. If the member has a language with ENCODE=Y, SCLM will encode the member when it is saved.

VERDEL
Decodes a member if it has a language which specifies ENCODE=Y. After updating the Version PDS member, SCLM will encode it when it is saved.

VERRECOV
Decodes the version PDS member to allow SCLM to restore a member. If a member is to be restored into an SCLM group and the member specifies it is encoded (ENCODE=Y on the FLMLANGL macro), the stored member will be encoded.

Setting up encoding and decoding

Encoding and decoding can be set up on editable members using the ENCODE=Y parameter on the FLMLANGL macro, or on SCLM outputs (non-editable members) using the ENCODE=Y parameter on the FLMALLOC parameter in the language translators.

To set up SCLM to use encoding and decoding, perform these steps:

1. Determine the languages of the editable members that are to be encoded.
2. Modify the required language translators to have ENCODE=Y specified on the FLMLANGL macro.
3. Determine for each language which SCLM outputs are required to be encoded.

   The outputs that are encoded are things like compiler listings. However, if outputs are to be used either by another language translator or external processes, the member needs to be decoded prior to being used. The ENDEC SCLM service can be used to decode the member.
4. Modify the language translators to specify ENCODE=Y on the FLMALLOC macro for the outputs that require the SCLM members to be encoded.

   The ENCODE=Y parameter requires an IOTYPE=O or P. The translator may need to be modified to handle this.
5. Assemble the SCLM project definition.
6. To ensure the existing editable members or the SCLM outputs are encoded, the ENDEC service can be run. Otherwise, the editable members are encoded the next time they are saved. The SCLM outputs are encoded the next time the language translator is run.

Removing encoding and decoding

To remove the encoding for a language, perform these steps:

1. Modify the language translator to specify ENCODE=N.
2. For all members with the language in question, you must run the ENDEC service to decode the members at all levels in the hierarchy.
3. For each of the members with that language, you must run the ENDEC service on the associated version PDS members at all levels in the hierarchy.

For the SCLM outputs, if the ENCODE=Y parameter on the FLMALLOC macro is modified or removed from a language translator, the non-editable member is not required to be decoded unless:

- SCLM users require SCLM output to be in a decoded state.
- The SCLM project does not have any SCLM language translators with an ENCODE=Y parameter specified on the FLMALLOC macro. In this case, SCLM does not know that SCLM project contains any encoded SCLM outputs and so encoded output members are not viewable from within SCLM.

Chapter 17. SCLM security

This chapter discusses the methods that can be used to secure your SCLM environment.

Table 26 on page 337 shows a summary of these methods.

Table 26. Summary of methods to secure an SCLM environment

Type of SCLM security	Methods	How it works	Comments
Securing the SCLM data sets	Enhanced Access Control (EAC)[1].	EAC ensures data sets can only be accessed within SCLM.	External 3rd party product.
	RACF or alternate security product[2].	Limits access to data sets to certain users.	Will not stop users accessing SCLM data sets outside of SCLM.
	SCLM project/alternate based access to data sets using XFACILIT resources[4].	XFACILIT resources limit what data sets can be used by an SCLM project/ alternate.	SCLM internal security features.
Securing members within SCLM	SCLM subproject security using XFACILIT resources[4].	SCLM allows members to be associated with subproject. XFACILIT resources can be used to define user access to the subprojects.	SCLM internal security features.
Encoding SCLM members	Internal SCLM process used to encode/decode SCLM members[3].	SCLM allows members to be encoded such that they are not viewable outside of SCLM.	SCLM internal security features.
Securing SCLM dialogs and services	Enhanced Access Control (EAC)[1].	EAC can be used to secure users access to the SCLM online dialog functionality and SCLM services.	External 3rd party product.
	SCLM dialog/service security using XFACILIT resources[4].	XFACILIT resources can be used to secure users access to the SCLM online dialog functionality and SCLM services.	SCLM internal security features.

Note:

1. Enhanced Access Control is a 3rd Party Product available through IBM which is external to SCLM and is not discussed in this chapter. For more information, refer to the EAC documentation.
2. To secure data sets using RACF or an alternate security product, see your security administrator or refer to the relevant security documentation.
3. To determine how to encode SCLM members, see Chapter 16, “Member encoding and decoding,” on page 333.
4. The SCLM internal security features are discussed in further detail in the remainder of this chapter.

© Copyright IBM Corp. 1990, 2019

SCLM internal security

The SCLM internal security features allow the items shown here to be secured using XFACILIT resources:

- SCLM project/alternate access to data sets (**SCLM DSN security**).

  This allows the SCLM administrator the ability to secure that SCLM controlled data sets and the SCLM account/version VSAM data sets to a SCLM project/alternate. This prevents users bypassing security by setting up their own SCLM project to point at production data sets.
- User access to members within an SCLM project/alternate (**SCLM subproject security**).

  An SCLM administrator can set up subprojects within an SCLM project/alternate and can associate members to these subprojects. The XFACILIT resources determine what access a user has to a subproject, and if they are able to access a member.

  Additionally, if member encoding is active then the editable members with ENCODE=Y specified for the associated language are not viewable outside of SCLM. For more information, see Chapter 16, “Member encoding and decoding,” on page 333.
- User access to SCLM dialogs and services (**SCLM service security**).

  An SCLM administrator can set up role-based security to determine what functionality or services a user is able to access.

  For example, you could set up a Developer role which has access to edit and build, but not promote. Alternatively, you could set up a Team leader role with the ability to promote and transfer ownership of members.

Enabling security

SCLM security is enabled on a z/OS partition by updating the IEASYM*xx* member to set the new symbol, SCLMSEC, to a value of "ACTIVE" and IPL'ing the z/OS system. For example:

```
SYMDEF(&SCLMSEC='ACTIVE')
```

During SCLM initialisation, the SCLMSEC symbol is retrieved and if it has a value of "ACTIVE", then SCLM security is active. This means that if the remaining installation steps have not been completed, you:

- Are not able to access the online options, or run FLMLNK or FLMCMD SCLM services.
- Do not have access to the SCLM-controlled data sets. This includes the account and audit/version VSAM data sets, as well as the hierarchical data sets specified in the SCLM project.
- Are forced to specify an SCLM subproject when saving a member. Since no subprojects have been specified in the SCLM project, you will encounter definition errors when attempting to save members in SCLM.

Note: The examples shown of setting up the XFACILIT resources use RACF. If you are using alternate security products, refer to the relevant manuals to determine how to set up the XFACILIT resources.

Determining the type of security to implement

By activating security, it means that all three types of SCLM security are automatically active. These security features are

- SCLM DSN security
- SCLM subproject security
- SCLM service security

The SCLM administrator must determine which security features are required for each SCLM project/alternate.

To turn off any of the security features, you must ensure that:

- The SCLM load module FLMSEC01 is in the link list and it is in an APF-authorised library.

- The program FLMSEC01 itself must be added to the authorized TSO command list in PARMLIB member IKJTSOxx.

If FLMSEC01 is not authorised, then security features are active even if there is a XFACILIT resource defined to turn it off.

To turn off **SCLM DSN security** for an SCLM project/alternate, you must create a XFACILIT resource class with a UACC of READ. The profile name should be in the format:

```
SCLM.SECDSN.OFF.project.alternate
```

Figure 169 on page 339 shows an example of turning off SCLM DSN security for an SCLM project/alternate.

```
CLASS      NAME
-----      ----
XFACILIT   SCLM.SECDSN.OFF.PRJ0120.* (G)

GROUP CLASS NAME
----- ----- ----
GXFACILI

LEVEL  OWNER      UNIVERSAL ACCESS  YOUR ACCESS  WARNING
-----  --------   ----------------  -----------  -------
 00    SCLM            READ              READ    NO

INSTALLATION DATA
-----------------
NONE

APPLICATION DATA
----------------
NONE

SECLEVEL
--------
NO SECLEVEL

CATEGORIES
----------
NO CATEGORIES

SECLABEL
--------
NO SECLABEL

AUDITING
--------
FAILURES(READ)

NOTIFY
------
NO USER TO BE NOTIFIED
```

Figure 169. Example of turning off SCLM DSN security for an SCLM project/alternate

To turn off SCLM subproject security for an SCLM project/alternate, you must create a XFACILIT resource class with a UACC of READ. The profile name should be in the format:

```
SCLM.SECSUB.OFF.project.alternate
```

Figure 170 on page 340 shows an example of turning off SCLM subproject security for an SCLM project/alternate.

```
CLASS      NAME
-----      ----
XFACILIT   SCLM.SECSUB.OFF.PRJ0120.* (G)

GROUP CLASS NAME
----- ----- ----
GXFACILI

LEVEL  OWNER      UNIVERSAL ACCESS  YOUR ACCESS  WARNING
-----  --------   ----------------  -----------  -------
 00    SCLM            READ              READ    NO

INSTALLATION DATA
-----------------
NONE

APPLICATION DATA
----------------
NONE

SECLEVEL
--------
NO SECLEVEL

CATEGORIES
----------
NO CATEGORIES

SECLABEL
--------
NO SECLABEL

AUDITING
--------
FAILURES(READ)

NOTIFY
------
NO USER TO BE NOTIFIED
```

Figure 170. Example of turning off SCLM subproject security for an SCLM project/alternate

To turn off SCLM service security for an SCLM project/alternate, you must create a XFACILIT resource class with a UACC of READ. The profile name should be in the format:

```
SCLM.SECSVC.OFF.project.alternate
```

Figure 171 on page 341 shows an example of turning off SCLM service security for an SCLM project/alternate.

```
CLASS      NAME
-----      ----
XFACILIT   SCLM.SECSVC.OFF.PRJ0120.* (G)

GROUP CLASS NAME
----- ----- ----
GXFACILI

LEVEL  OWNER      UNIVERSAL ACCESS  YOUR ACCESS  WARNING
-----  --------   ----------------  -----------  -------
 00    SCLM            READ              READ    NO

INSTALLATION DATA
-----------------
NONE

APPLICATION DATA
----------------
NONE

SECLEVEL
--------
NO SECLEVEL

CATEGORIES
----------
NO CATEGORIES

SECLABEL
--------
NO SECLABEL

AUDITING
--------
FAILURES(READ)

NOTIFY
------
NO USER TO BE NOTIFIED
```

Figure 171. Example of turning off SCLM service security for an SCLM project/alternate

Setting up SCLM DSN security

If SCLM DSN security is active, you must specify which data sets an SCLM project/alternate has access to.

This DSN security does not provide security for data sets to individual users. It only stops users from setting up a similar project using the data sets of another SCLM project in an attempt to bypass processes set up the SCLM administrator.

To secure the SCLM data sets that users have access to, you must use one or both of these:

- Enhanced Access Control (EAC) to ensure the SCLM data sets are only accessed under SCLM.
- RACF, or an alternate security product, to secure the SCLM-controlled data sets.

The data sets secured by SCLM DSN security are the accounting and audit/version VSAM data sets defined to the SCLM project, as well as the SCLM-controlled hierarchy data sets.

If you are using the FLMALTC macro to specify alternate source data sets, you must specify the actual data set name allocated by SCLM when securing the SCLM-controlled hierarchy data sets.

To secure the data sets, you must create a XFACILIT resource class with a UACC of READ. The profile name should be in the format:

```
SCLM.DSN.project.alternate.dsn
```

where:

project
The SCLM project name.

alternate
The SCLM alternate project name.

dsn
The data set you want to secure.

Note: You can set up generic resources by specifying an asterisk (*) for either the *project*, *alternate*, or *process* in the profile name.

These are examples of specifying generic resources:

SCLM.DSN.SCLM01.ALT01.SCLM01.DEV.**
Selects all SCLM01.DEV data sets.

SCLM.DSN.SCLM01.ALT01.SCLM01.*.SOURCE.**
Selects all SCLM01 SOURCE data sets.

Figure 172 on page 342 shows an example of setting up SCLM DSN security.

```
CLASS      NAME
-----      ----
XFACILIT   SCLM.DSN.PRJ0120.*.PRJ0120.** (G)

GROUP CLASS NAME
----- ----- ----
GXFACILI

LEVEL  OWNER      UNIVERSAL ACCESS  YOUR ACCESS  WARNING
-----  --------   ----------------  -----------  -------
 00    SCLM            NONE              READ    NO

INSTALLATION DATA
-----------------
NONE

APPLICATION DATA
----------------
NONE

SECLEVEL
----------------
NONE

SECLEVEL
--------
NO SECLEVEL

CATEGORIES
----------
NO CATEGORIES

SECLABEL
--------
NO SECLABEL

AUDITING
--------
FAILURES(READ)

NOTIFY
------
NO USER TO BE NOTIFIED

      USER      ACCESS
----      ------
AUDITOR    READ
DEVELOP    READ
```

Figure 172. Example of setting up SCLM DSN security

Setting up SCLM subproject security

If SCLM subproject security is active, you must:

1. Define the subprojects available to each SCLM project/alternate using the FLMPROJ macro to specify each subproject. For example:

```
PAYROLL    FLMPROJ DESC='PAYROLL SYSTEM'
ACCOUNTS   FLMPROJ DESC='ACCOUNT PAYABLE'
```

2. Re-assemble the SCLM project definition.
3. Define who has access to the subprojects using the XFACILIT resources.

To define access to the subprojects, you must create a XFACILIT resource class with a UACC of NONE. The profile name should be in the format:

```
SCLM.SUB.project.alternate.subproject.type
```

where:

project
The SCLM project name.

alternate
The SCLM alternate project name.

subproject
The subproject defined in SCLM that you want to secure.

type
Determines what types in the SCLM project you can access.

The *type* on this resource class can be used to refine the security to allow you to define what types a user can access within the SCLM subproject.

Note: You can set up generic resources by specifying an asterisk (*) for either the *project*, *alternate*, *subproject*, or *type* in the profile name.

Figure 173 on page 344 shows an example of setting up SCLM subproject security.

```
CLASS      NAME
-----      ----
XFACILIT   SCLM.SUB.PRJ0120.*.VISA.* (G)

GROUP CLASS NAME
----- ----- ----
GXFACILI

LEVEL  OWNER      UNIVERSAL ACCESS  YOUR ACCESS  WARNING
-----  --------   ----------------  -----------  -------
 00    SCLM            NONE             UPDATE    NO

INSTALLATION DATA
-----------------
NONE

APPLICATION DATA
----------------
NONE

SECLEVEL
NO SECLEVEL

CATEGORIES
----------
NO CATEGORIES

SECLABEL
--------
NO SECLABEL

AUDITING
--------
FAILURES(READ)

NOTIFY
------
NO USER TO BE NOTIFIED

USER      ACCESS
----      ------
AUDITOR   READ
DEVELOP   UPDATE


   ID     ACCESS  CLASS                ENTITY NAME
-------- ------- -------- ---------------------------------------
NO ENTRIES IN CONDITIONAL ACCESS LIST
```

Figure 173. Example of setting up SCLM subproject security

Once the XFACILIT resource has been created, you must provide users access to the XFACILIT resource. The easiest way to do this is to set up a RACF group and give this group the required access to the subproject. Users can be given access to the RACF group.

The access given to a user or RACF group to the subproject XFACILIT resource is important in defining what the user is able to do with a member. For example, when a member is being edited, what a user is able to do depends on whether they have READ, UPDATE, or ALTER authority to the subproject XFACILIT resource.

So, for example, to provide auditors with read access and developers or project leaders with update access to the subproject VISA, you set up two groups:

AUDITOR
Read access to XFACILIT resource SCLM.SUB.PRJ0120.*.VISA.*

VISA
Update access to XFACILIT resource SCLM.SUB.PRJ0120.*.VISA.*

In this example, in RACF you would provide given the required users access to the AUDITOR and VISA RACF groups. Once the RACF resources have been refreshed, the user should have the ability to access members with a VISA subproject in the SCLM project PRJ0120.

Table 27 on page 345 lists the various SCLM functions and the subproject access provided to the user.

Table 27. Subproject access for each SCLM function

Service	Subproject access			Notes
	READ	UPDATE	ALTER	
ACCTINFO	Y			READ access allows the user to display the account information. Issue the A line command in Library Utility (option 3.1) or UOW (option 3.11) and retrieve account information using the ACCTINFO service.
AUTHCODE	Y	Y		READ access allows the user to retrieve authcode information using the AUTHCODE service. UPDATE access allows the authcode to be modified using the U line command in Library Utility (option 3.1) or UOW (option 3.11) and using the AUTHCODE service.
BUILD	Y	Y	Y	Building using build (option 4), C line command in Library Utility (option 3.1) and UOW (option 3.11), or the BUILD service validates the access prior to each translator step: • READ access allows input members to be used to generate outputs. • Where the output member does not exist, the subproject is copied from the initial input member. • Where the output member does exist and the subproject is not changing, the user must have UPDATE access to the output member. • Where the output member already exists and the subproject is to change, the user must have ALTER access to the output member.
CCEXITS				The CCEXITS service access will be the same as for Edit.
DBACCT	Y			READ access for the DBACCT service allows SCLM to return the accounting information.
DBUTIL				No access validation is performed.
DELETE		Y		Update access allows the D line command in Library Utility (option 3.1) or UOW (option 3.11) and the DELETE service to delete the member.
DELGROUP		Y		UPDATE access allows the Delete from group (option 3.9) and the DELGROUP service to delete the members with subproject with UPDATE access.
DSALLOC				No access validation is performed.

Table 27. Subproject access for each SCLM function (continued)

Service	Subproject access			Notes
	READ	UPDATE	ALTER	
EDIT	Y	Y	Y	Editing using edit (option 2), E line command in Library Utility (option 3.1) and UOW (option 3.11) or the EDIT service validates the access in this way: **When entering EDIT:** No access is required if a new member or an existing member with no subproject is to be edited. UPDATE access to the subproject is required if the member exists with a subproject. **When saving the member:** If saving the member with the same subproject or if the member did not have a subproject, then UPDATE access for the subproject is required. If saving the member with a different subproject, then ALTER access for the existing subproject and UPDATE access for the new subproject is required.
END				No access validation is performed.
EXPORT	Y			READ access for each of the members allows them to be exported by means of Export (option 3.6) or the EXPORT service.
FREE				No access validation is performed.
GETBLDMP	Y			READ authority allows the M line command in Library Utility (option 3.1) and UOW (option 3.11) or the GETBLMP service to display or retrieve the build map information.
IMPORT	Y	Y	Y	Importing using Import (option 3.7) or the IMPORT service validates the access depending on the status of the member: • If the member is new, then only READ access to the subproject is required. • If the member exists with the same subproject, then only UPDATE access to the subproject is required. • If the member exists with a different subproject, then only ALTER access for the existing member's subproject and UPDATE authority to the incoming member's subproject is required.
INIT				No access validation is performed.
LOCK		Y		UPDATE access allows the LOCK service to lock the member.
MIGRATE		Y	Y	Migrating using Migrate (option 3.3) or the MIGRATE service validates the access depending on the status of the part: • If the part is new or does not have a subproject, READ access to the new subproject allows the member to be migrated. • If the part is already defined to SCLM and subproject passed to migrate does not match the existing subproject, then UPDATE access is required to the existing subproject and ALTER to the new subproject. • If the part is already defined to SCLM and subproject passed to migrate matches the existing subproject, UPDATE access is required to the existing subproject.

Table 27. Subproject access for each SCLM function (continued)

Service	Subproject access			Notes
	READ	UPDATE	ALTER	
NEXTGRP				No access validation is performed.
PARSE				No access validation is performed.
PROMOTE		Y		Promoting using Promote (option 5), P line command in Library Utility (option 3.1) and UOW (option 3.11) or the PROMOTE service validates the access in this way: UPDATE access to each member's subprojects is required for the promote to complete successfully.
RPTARCH				No access validation is performed.
SAVE		Y		The SAVE service validates access depending on the status of the part: • If the part is new or does not have a subproject, READ access to the new subproject allows the member to be saved. • If the part is already defined to SCLM and subproject passed to save does not match the existing subproject, then UPDATE access is required to the existing subproject and ALTER to the new subproject. • If the part is already defined to SCLM and subproject passed to save matches the existing subproject, UPDATE access is required to the existing subproject.
SCLMINFO				No access validation is performed.
START				No access validation is performed.
STORE		Y		The STORE service validates access depending on the status of the part: • If the part is new or does not have a subproject, READ access to the new subproject allows the member to be saved. • If the part is already defined to SCLM and subproject passed to save does not match the existing subproject, then UPDATE access is required to the existing subproject and ALTER to the new subproject. • If the part is already defined to SCLM and subproject passed to save matches the existing subproject, UPDATE access is required to the existing subproject.
TRANSFER		Y		UPDATE access allows transferring of ownership using the T line command in Library Utility (option 3.1) and UOW (option 3.11) for members with a subproject.
UNLOCK		Y		UPDATE access allows the UNLOCK service to unlock the member.
VERDEL		Y		UPDATE access allows the D line command in the Audit and Version Utility (option 3.8) and the VERDEL service to delete the version/audit record.
VERHIST	Y			READ access allows the H line command in the Audit and Version Utility (option 3.8) and the VERHIST service to produce the version history report.

Table 27. Subproject access for each SCLM function (continued)				
Service	**Subproject access**			**Notes**
	READ	**UPDATE**	**ALTER**	
VERINFO	Y			READ access allows the A and V line commands in the Audit and Version Utility (option 3.8) and the VERINFO service to read the version information or view the version member.
VERRECOV		Y		READ access allows the C, X, and R line commands in the Audit and Version Utility (option 3.8) and the VERRECOV service to restore the version member.

Setting up SCLM service security

If SCLM process security is active, you must specify what processes within SCLM a user has access to.

The easiest way to do this is to:

1. Group the users by the access they require within SCLM. For example: developer, project manager, SCLM administrator, and so on.
2. Set up RACF security groups with those groups or roles.
3. Provide the users with access to the appropriate RACF group.
4. Provide access for each of the SCLM functions to the appropriate security group (roles) you set up previously. To see how this is done, refer to the documentation below.

By setting up security groups in this way, it means that when adding a new user you simply have to give the new user access to the appropriate security group; for example, developer.

Table 28 on page 348 shows the processes which you are able to secure.

Table 28. SCLM processes that can be secured	
Process	**SCLM functionality secured**
ACCTINFO	ACCTINFO service and the A line command in Library Utility (option 3.1) and Unit of Work (option 3.11).
AUTHCODE	AUTHCODE service and the U line command in Library Utility (option 3.1) and Unit of Work (option 3.11).
BUILD	BUILD service, Build (option 4) and the C line command in Library Utility (option 3.1) and Unit of Work (option 3.11).
DBACCT	DBACCT service.
DELETE	DELETE service and the D line command in Library Utility (option 3.1) and Unit of Work (option 3.11).
DELGROUP	DELGROUP service and DELGROUP utility (option 3.9).
DSALLOC	DSALLOC service.
EDIT	EDIT service, Edit (option 2) and the E line command in Library Utility (option 3.1) and Unit of Work (option 3.11).
ENDEC	ENDEC service (Encode/Decode).
EXPORT	EXPORT service and Export utility (option 3.6).
IMPORT	IMPORT service and Import utility (option 3.6).
LOCK	LOCK service.

Table 28. SCLM processes that can be secured (continued)	
Process	**SCLM functionality secured**
MIGRATE	MIGRATE service and Migrate utility (Option 3.3).
NOPROM	NOPROM service and the N line command in Library Utility (option 3.1) and Unit of Work (option 3.11).
PROMOTE	PROMOTE service, Promote Utility (option 4) and the P line command in Library Utility (option 3.1) and Unit of Work (option 3.11).
SAVE	SAVE service.
SEARCH	Search Utility (option 3.13).
STORE	STORE service.
TRANSFER	The T line command in Library Utility (option 3.1) and Unit of Work (option 3.11).
UNLOCK	UNLOCK service.
VERDEL	VERDEL service and D line command in Audit and Version Utility (option 3.8).
VERHIST	VERHIST service and H line command in Audit and Version Utility (option 3.8).
VERINFO	VERINFO service and V line command in Audit and Version Utility (option 3.8).
VERRECOV	VERRECOV service and R line command in Audit and Version Utility (option 3.8).
VIEW	View utility (option 1) and the V or B line command in Library Utility (option 3.1) and Unit of Work (option 3.11).

To secure the online dialogs and services, you must create an XFACILIT resource class with a UACC of NONE. The profile name must be in the format:

```
SCLM.SVC.project.alternate.process
```

where:

project
The SCLM project name.

alternate
The SCLM alternate project name.

process
The SCLM process you want to secure.

Once the XFACILIT resource class has been created, you must provide the access to the XFACILIT resource to the appropriate users or group.

Note: You can set up generic resources by specifying an asterisk (*) for either the *project*, *alternate*, or *process* in the profile name.

Working with subproject security

This section describes subproject security.

Migrating members into SCLM subproject security

If you have existing members that you want to set up to use subproject security, once the SCLM project definition has been updated to define the subprojects and the XFACILIT resource class and user access has been created, you can associate members to subprojects by performing one of these tasks:

- Editing the member and typing the SPROF command to specify a subproject.

Issuing the SPROF command displays the panel shown in Figure 174 on page 350:

```
                        SCLM Edit Profile
Command ===> ____________________________________________________________

    SCLM Library: PRJ0120.ISPFRUN.ASM
    Member: AAC001

Press the Enter key with the language field blank to view a list of
valid languages or enter the desired values and press Enter.

Enter the Cancel command to exit with no change.

Language . . . . HASM____
Sub-project  . . ASSEMBLE  ASSEMBLER PROJECT
Change code  . . ________  (Use "=" to retrieve last entry)
Description  . . ________________________________________
```

Figure 174. SCLM Edit Profile panel

Blanking out the **Sub-project** field and pressing Enter displays the Valid Sub-projects panel shown in Figure 175 on page 350 where you can select a project.

```
                        Valid Sub-projects             Sub-project required
Command ===> ____________________________________________ SCROLL ===> PAGE


  Select one of the following valid sub-projects for member ZZZZ
  in Project PRJ0120

    S   Sub-project  Description
    -   -----------  ------------------------------------
    _   PAYROLL      PAYROLL SUB PROJECT
    _   VISA         VISA SUB PROJECT
****************************** Bottom of data ********************************
```

Figure 175. Valid Sub-projects panel

- Migrating the members using Migrate (option 3.3) and specifying the subproject. Migrate (option 3.3) displays the SCLM Migration Utility entry panel shown in Figure 176 on page 350

```
                    SCLM Migration Utility - Entry Panel
Command ===> ____________________________________________________________

Selection criteria:
  Project  . : PRJ0120
  Group  . . . ISPFRUN
  Type . . . . ASM_____
  Member . . . AAC002__    (Pattern may be used)

Member information:
  Authorization code  . . ________    Mode  . . . 3  1. Conditional
  Change code . . . . . . ________                   2. Unconditional
  Language  . . . . . . . HLASM___                   3. Forced
  Subproject  . . . . . . VISA____
Output control:
               Ex Sub                  Process . . 2  1. Execute
  Messages . . 1  2    1. Terminal                    2. Submit
  Report . . . 1  2    2. Printer
  Listings . . 1  2    3. Data set     Printer . . H
                       4. None         Volume  . . ______
```

Figure 176. SCLM Migration Utility entry panel

- Migrating the members using the MIGRATE service and specifying the subproject.

  For example:

```
FLMCMD MIGRATE,PRJ0120,PRJ0120,ISPFRUN,ASM,AAC005,
,HASM,,F,FLMMSG,,FLMRPT,,,VISA
```

  or

```
MOVE 'MIGRATE'   TO SCLM-SERVICE
CALL 'FLMLNK' USING SCLM-SERVICE
                    SCLM-SCLM-ID
                    SCLM-GROUP
                    SCLM-TYPE
                    SCLM-MEMBER
                    SCLM-AUTHCODE
                    SCLM-LANGUAGE
                    SCLM-CHANGE-CODE
                    SCLM-MODE
                    SCLM-DD-MSGS
                    SCLM-DD-PARSE
                    SCLM-DD-REPT
                    SCLM-DATE
                    SCLM-TIME
                    SCLM-SUBPROJ
```

- Saving the member using the SAVE SCLM service and specifying the subproject.

  For example:

```
FLMCMD SAVE,PRJ0120,PRJ0120,ISPFRUN,ASM,AAC003,,,,HASM,N,
       ,C,C,,VISA
```

  or

```
MOVE 'SAVE'      TO SCLM-SERVICE
CALL 'FLMLNK' USING SCLM-SERVICE
                    SCLM-SCLM-ID
                    SCLM-GROUP
                    SCLM-TYPE
                    SCLM-MEMBER
                    SCLM-AUTHCODE
                    SCLM-ACCESS-KEY
                    SCLM-USERID
                    SCLM-LANGUAGE
                    SCLM-SUP-PARS-LIST
                    SCLM-DD-PARSE
                    SCLM-CALL-STORE
                    SCLM-COMP-UNIT
                    SCLM-VERIFY-CC
                    SCLM-LIST-INFO
                    SCLM-MAX-PROM
                    SCLM-MSG-ARRAY
                    SCLM-SUBPROJ
```

- Saving the member using the LOCK/PARSE/STORE services.

```
MOVE 'STORE'     TO SCLM-SERVICE
CALL 'FLMLNK' USING SCLM-SERVICE
                    SCLM-SCLM-ID
                    SCLM-GROUP
                    SCLM-TYPE
                    SCLM-MEMBER
                    SCLM-ACCESS-KEY
                    SCLM-LANGUAGE
                    SCLM-USERID
                    SCLM-CALL-STORE
                    SCLM-VERIFY-CC
                    SCLM-STATS-INFO
                    SCLM-LIST-INFO
                    SCLM-MSG-ARRAY
                    SCLM-SUBPROJ
```

Viewing the subproject members

Select the SCLM Library Utility (option 3.1) to display the SCLM Library Utility entry panel shown in Figure 177 on page 352.

```
                    SCLM Library Utility - Entry Panel
Option ===> ____________________________________________________________________

blank Display member list    E Edit member            T Transfer owner
    A Browse account info    V View member            N NOPROM processing
    M Browse build map       C Build member           W WhereUsed
    B Browse member          P Promote member
    D Delete member info     U Update auth code

SCLM Library:
  Project  . : PRJ0120
  Group  . . . ISPFRUN
  Type . . . . ASM
  Member . . . ________    (Blank or pattern for member selection list)

Select and rank member list data  . . ATS  (T=TEXT, A=ACCT, M=BMAP, S=SUBP)

Enter "/" to select option
/  Hierarchy view                     Process . . 3  1. Execute
/  Confirm delete                                    2. Submit
/  View processing options for Edit                  3. View options
   Show Member Description
```

Figure 177. SCLM Library Utility entry panel

Type S in the **Select and rank member list data entry** field and press Enter to display a member list panel (see Figure 178 on page 352) on which the subproject associated with each member is displayed.

```
Member List : PRJ0120.ISPFRUN.ASM - HIERARCHY VIEW -            Member 1 of 5
Command ===> ____________________________________________ Scroll ===> CSR

A=Account  M=Map      B=Browse  D=Delete    E=Edit     V=View
C=Build    P=Promote  U=Update  T=Transfer  N=Noprom   W=WhereUsed

_   Member   Status      Account  Text       Chg Date   Chg Time   Subproj
_   AAC001               ISPFRUN  ISPFRUN    2008/04/30 16:00:39   ASSEMBLE
_   AAC002               ISPFRUN  ISPFRUN    2008/04/30 16:01:09   ASSEMBLE
_   AAC003               ISPFRUN  ISPFRUN    2008/04/30 16:01:18   ASSEMBLE
_   AAC004               ISPFRUN  ISPFRUN    2008/04/24 09:46:42   ASSEMBLE
_   AAC005               ISPFRUN  ISPFRUN    2008/04/30 16:01:25
****************************** Bottom of data ********************************
```

Figure 178. Member list panel showing associated subprojects

Resolving authority problems

If as a SCLM administrator you encounter authority problems, the method shown here should help you to resolve your problems.

1. For the SCLM project/alternate with which you are having problems, set up an XFACILIT resource in this format with a UACC of READ:

   ```
   SCLM.SECDBG.ON.project.alternate
   ```

   where:

   project
   The SCLM project name to be debugged.

   alternate
   The SCLM alternate project name to be debugged.

   Figure 179 on page 353 shows the XFACILIT SECDBG profile.

```
CLASS      NAME
-----      ----
XFACILIT   SCLM.SECDBG.ON.PRJ0120.* (G)

GROUP CLASS NAME
----- ----- ----
GXFACILI

LEVEL  OWNER      UNIVERSAL ACCESS  YOUR ACCESS  WARNING
-----  --------   ----------------  -----------  -------
 00    SCLM           READ               READ    NO

INSTALLATION DATA
-----------------
NONE

APPLICATION DATA
----------------
NONE
```

Figure 179. XFACILIT SECDBG profile

Note: Once the RACF resources have been refreshed, debugging information is displayed for every person or job which accesses the project/alternate.

2. After the RACF resource rules have been refreshed, SCLM displays information like that shown in Figure 180 on page 353.

```
FLMC0SVI: SCLM.SECSUB.OFF.PRJ0120.PRJ0120
    ACCESS = READ
    SEC00RC=  8
    SAFRC1 =  8
    SAFRC2 =  8
    SAFRE  =  0
  FLM085   Security error. RACROUTE REQ=AUTH, SAF RC=08, RACF RC=08, RACF RS=00
 FLMC0SVI: SCLM.SECSVC.OFF.PRJ0120.PRJ0120
    ACCESS = READ
    SEC00RC=  0
    SAFRC1 =  0
    SAFRC2 =  0
    SAFRE  =  0

 FLMC0SVI: SCLM.SECDSN.OFF.PRJ0120.PRJ0120
    ACCESS = READ
    SEC00RC=  0
    SAFRC1 =  0
    SAFRC2 =  0
    SAFRE  =  0

 FLMS1II: SCLM Security Status:
          Global     = ON
          Dataset    = OFF
          Service    = OFF
          Subproject = ON
```

Figure 180. SCLM security debug information

The information as displayed in Figure 180 on page 353 shows the set up of the SCLM security. You can see that SCLM attempted to read the XFACILIT resource SCLM.SECSUB.OFF.PRJ012.PRJ0120 to determine if subproject security was switched off. The RC=8 indicates that the SCLM did not find a subproject XFACILIT resource or, if it exists, it has a UACC of NONE. Hence, subproject security is active.

The next two calls were successful, showing that SCLM DSN security and SCLM service security are disabled.

3. Get the user to continue and reproduce the problem. You should see more debugging information relating to the error, like that shown in Figure 181 on page 354.

```
FLMC0SVI: SCLM.SUB.PRJ0120.PRJ0120.VISA.COBOL
    ACCESS = READ
    SEC00RC=  0
    SAFRC1 =  8
    SAFRC2 =  8
    SAFRE  =  0
  FLM085   Security error. RACROUTE REQ=AUTH, SAF RC=08, RACF RC=08, RACF RS=00
 FLMC0SVJ: rc  24 subproj=COBOL
```

Figure 181. Additional SCLM security debug information

From the above, you now know it is the XFACILIT resource that is causing the problem.

4. Go into RACF General Resource profiles (option 2) and display the resource by specifying option 'D' Display profile contents:

```
                      RACF - GENERAL RESOURCE SERVICES -  DISPLAY
OPTION ===>

ENTER THE FOLLOWING PROFILE INFORMATION:

   CLASS     ===> XFACILIT

   PROFILE   ===> SCLM.SUB.PRJ0120.PRJ0120.VISA.COBOL


                         <==end of data

      NOTE: Embedded Blanks are NOT ALLOWED in class or profile names.
            The profile name may be case sensitive.  View the help and
            select PROFILE NAME for more detail.
```

5. Type in YES in the **ACCESS LIST** selection field:

```
                  RACF - DISPLAY GENERAL RESOURCE PROFILE
COMMAND ===>

  CLASS:         XFACILIT
  PROFILE      _ SCLM.SUB.PRJ0120.PRJ0120.VISA.COBOL

 Enter YES     to select a profile type:
   ___  DISCRETE    ___  GENERIC    ___  NOGENERIC

 Enter YES     to select one or more of the following:
   ___  RESOURCE GROUP     ___  STDATA             ___  ICTX DATA
   YES  ACCESS LIST        ___  SECURED SIGNON
   ___  HISTORY            ___  SYSTEMVIEW
   ___  STATISTICS         ___  KERBEROS
   ___  TVTOC              ___  LDAP PROXY
   ___  SESSION            ___  EIM
   ___  DLF DATA           ___  CDTINFO

   ___  NO RACF
   ___  NO YOUR-ACCESS
```

6. Press Enter. SCLM displays the XFACILIT resource:

```
CLASS      NAME
-----      ----
XFACILIT   SCLM.SUB.PRJ0120.*.VISA.* (G)

GROUP CLASS NAME
----- ----- ----
GXFACILI

LEVEL  OWNER      UNIVERSAL ACCESS  YOUR ACCESS  WARNING
-----  --------   ----------------  -----------  -------
 00    SCLM            NONE               NONE    NO

INSTALLATION DATA
-----------------
NONE

APPLICATION DATA
----------------
NONE

SECLEVEL
--------
NO SECLEVEL

CATEGORIES
----------
NO CATEGORIES

SECLABEL
--------
NO SECLABEL

AUDITING
--------
FAILURES(READ)

GLOBALAUDIT
-----------
NONE

NOTIFY
------
NO USER TO BE NOTIFIED


USER      ACCESS
----      ------
USERS2   UPDATE

   ID     ACCESS  CLASS                ENTITY NAME
-------- ------- -------- -------------------------------------
NO ENTRIES IN CONDITIONAL ACCESS LIST
```

Notice that, even though you entered a fully-qualified resource, RACF returned the actual resource which secures that resource, SCLM.SUB.PRJ0120.*.VISA.*.

In this case, the user USERS1 did not have access to the subproject VISA.

Part 4. SCLM Reference

© Copyright IBM Corp. 1990, 2019

Chapter 18. Invoking the SCLM services

This chapter introduces the services you can use to retrieve and process information that is stored in SCLM project hierarchies. It describes and provides brief examples showing the different interfaces you can use to invoke the services:

- The FLMCMD command processor interface
- The FLMLNK subroutine call interface
- The SCLM Commands interactive panels

It lists the general categories of parameters, variables, and return codes relevant to invoking SCLM services. It also explains the notation conventions used to document the services.

Invoking the SCLM services

You can invoke the SCLM services in any of the following ways:

- By a command function dialog (CLIST or REXX) through the ISPF interface.
- By a program function dialog through a call to FLMCMD or FLMLNK.
- By selecting the service from the SCLM FLMCMD Services Menu, then entering the service parameters in an ISPF interface panel.
- By entering a command to invoke a specific service panel, then entering the service parameters in the panel.

Command invocation of the SCLM services

The SCLM services can be invoked by using the FLMCMD command in a CLIST or REXX command procedure or by issuing the FLMCMD command as a TSO command.

You cannot invoke the following services using the FLMCMD command:

```
DBACCT          PARSE
END             START
FREE            STORE
INIT
```

The FLMCMD interface

The general format for a command invocation is:

```
FLMCMD service_name,project_name,prj_def_name,parameter1,parameter2,...
```

The maximum length of the command invocation statement is 512 characters.

FLMCMD parameter conventions

service_name
Alphanumeric; up to 8 characters long.

project_name
Alphanumeric; up to 8 characters long.

prj_def_name
Alphanumeric; up to 8 characters long.

The remaining parameters are positional and depend on the service being requested.

© Copyright IBM Corp. 1990, 2019

Lowercase parameters are optional. If a value is not specified for an optional parameter, SCLM will use default values if they exist. All default values are described within the parameter descriptions for each service.

If you omit a parameter, account for it by inserting a comma in its place. The following example shows how you would omit `parm2`:

```
FLMCMD service_name,project_name,prj_def_name,parm1,,parm3
```

Do not insert blanks in the command format. Blanks entered before a parameter will cause the value passed to the service to be incorrectly padded with leading blanks.

Using command invocation variables

If you invoke FLMCMD from a CLIST, you can use a CLIST variable anywhere within a statement as the service name or as a parameter. A CLIST variable consists of a name preceded by an ampersand (&). The CLIST processor replaces each variable with its current value before processing the FLMCMD command.

Note: SCLM follows all rules pertaining to TSO CLISTs. For more information, refer to *z/OS TSO/E Command Reference* and *z/OS TSO/E CLISTs*.

Using the FLMCMD file format

Use the FILE format of FLMCMD to process multiple commands as a single command invocation. You can enter the multiple commands either in a data set or from your screen. The FILE format of the command invocation is:

The ddname is the data definition name allocated to the FLMCMD command data set. The record length of the command data set cannot exceed 255 bytes. If you do not specify the ddname, SCLM enters interactive mode and prompts you for command lines. For more information, see "Interactive command processing" on page 361.

Performance considerations

The START service loads the SCLM modules that can be processed into memory and initializes the SCLM service environment. The INIT service loads the load module of a project definition into memory. The FREE service closes all of the open project databases. Each of these functions takes time. Therefore, to optimize the SCLM services execution time, minimize the number of START, INIT, and FREE service calls.

You can reduce the number of START, INIT, and FREE service calls by using the FILE format of FLMCMD. As an SCLM service program, the FLMCMD command processor must call the START service to begin a service session. It must also call the INIT and FREE services for every unique `project/prj_lib_def` combination it encounters. Therefore, ten separate invocations of the FLMCMD command processor result in nine more calls to the START service and nine more calls to the INIT and FREE services than one invocation of the FLMCMD command processor that has all ten commands in a data set.

In addition, opening a command file takes time. In processing a single command, the general format of FLMCMD processes faster than the FILE format of FLMCMD.

SCLM opens the VSAM data sets for a project as they are needed; however each open takes time. Projects can reduce the number of opens required by reducing the number of data sets defined on the FLMCNTRL and FLMALTC macros in the project.

Command data set conventions

Command data sets use the following conventions:

- The sequence numbers of the command data set should be turned off.
- SCLM processes all commands in the command data set regardless of the success or failure of previous commands.

- Each command must start on a new line.
- If a command takes more than one line, the continuation character should be the first character of the continuation line.

  If you enter spaces between the continuation character and the character that follows, those spaces will be treated as part of the parameter.
- If a command line exceeds the maximum record length of the command data set, continue the command by adding a plus sign (the continuation character) in the first position of the continuation line. You can add any number of continuation lines for any command.
- The maximum command length is 512 bytes. Note that if a command consists of several command lines, SCLM deletes trailing blanks.
- An asterisk (*) indicates comment lines. Place it in the first nonblank character of a command line. You can enter any number of comments within the command data set, but you cannot add a comment line within a series of command continuation lines.

The following example shows a command data set. The first command calls the SCLM LOCK service; the second command calls the SCLM UNLOCK service.

```
*
* This is an example of a command data set.
     * Note that comments do not have to start in column 1.
*
* The following command calls the SCLM LOCK service.
LOCK,PROJ1,,USER1,SOURCE,FLM01MD2,TESTAC,XXX#04,USERID
*
* The following command consists of four lines,
* and calls the SCLM UNLOCK service.
UNLOCK,PROJ1,,
+USER1,
+SOURCE,
+FLM01MD2,XXX#04
```

The following example shows a CLIST command procedure that calls the FILE format of FLMCMD.

```
PROC 0
  ALLOC DDNAME(SCLMIN) DA('USERID.FLMCMD.INPUT') SHR
  FLMCMD FILE,SCLMIN
  SET &FLMCMDCC =
  FREE DDNAME(SCLMIN)
  EXIT CODE(&FLMCMDCC)
END
```

Interactive command processing

To use interactive command processing, omit the ddname input parameter when using the FILE format of FLMCMD. You then get a prompt for the Command lines. SCLM processes your input exactly as if the commands were in a command data set.

Figure 182 on page 362 shows a sample interactive command session.

```
  Menu  List  Mode  Functions  Utilities  Help
 ------------------------------------------------------------------------------
                               ISPF Command Shell
 Enter TSO or Workstation commands below:

 ===> FLMCMD FILE ____________________________________________________________
_______________________________________________________________________________
_______________________________________________________________________________

 Place cursor on choice and press enter to Retrieve command

 =>
 =>
 =>
 =>
 =>
 =>
 =>
 =>
 =>
 Enter a command line; press enter to process a command line; or 'QUIT'
LOCK,PROJ1,,USER1,SOURCE,FLM01MD2,TESTAC,XXX#04,USERID
```

Figure 182. Sample Interactive Command Session (ISRTSO)

To end interactive command processing, enter the QUIT command.

Note:

1. You must perform interactive command processing, like all SCLM processing, from an ISPF environment. Otherwise, the following error message appears:

   ```
   ISPS118 SERVICE NOT INVOKED. A VALID ISPF ENVIRONMENT DOES NOT EXIST.
   ```

2. During interactive command processing, you can enter comment lines but you cannot enter continuation lines.
3. If you allocate the ddname to your screen and also specify it on the FILE format of FLMCMD, you can get unpredictable results.

For a description of the FLMCMD return codes, see “SCLM service return codes” on page 377.

Call invocation of the SCLM services

Programs can use the FLMLNK subroutine interface to call the SCLM services. The examples in Chapter 19, “SCLM services,” on page 379 show call statements in Pascal syntax and service names and keywords as literals enclosed in single quotes (' ').

Note: None of the languages require you to use literals. You can specify parameters as variables, as in the examples on the following pages.

You cannot call the following services using the FLMLNK subroutine interface:

DBUTIL
RPTARCH

Note: SCLM services can be issued from function modules that reside either below or above the 16-megabyte line. The interface module FLM$LNK, alias FLMLNK has the attributes RMODE(24) and AMODE(ANY). These attributes allow both 24-bit and 31-bit addressing mode callers. Modules that reside above the 16-megabyte line (RMODE(ANY)) and include FLM$LNK in their load module can override the RMODE(24) attribute during link-edit. Data areas above the 16-megabyte line are also supported.

Standard register conventions are used. Registers 2-14 are preserved across the call.

The FLMLNK subroutine interface

See:

- “FLMLNK parameter conventions” on page 363

- "FORTRAN, Pascal, and C" on page 363
- "PL/I" on page 364
- "COBOL" on page 364

FLMLNK parameter conventions

Note: If you are using FLMLNK, you must pad each parameter to the maximum length. To do so, you must insert blank spaces so that each parameter takes up exactly the maximum amount of space allotted for it.

Programs in the FLMLNK subroutine interface use the following conventions:

- The service_name parameter is positional and required. All other parameters must appear in the order described for each service. Parameter positions on the CALL statement must specify a value up to the last parameter coded. Some services allow for CALLs where the parameter list ends before the last one in the service description, thus taking the default specification for those parameters (see individual service descriptions for details).
- SCLM uses the maximum parameter length when referencing and updating parameter values. Parameter values with fewer characters than the maximum must be padded with blanks for the remainder of the field. Parameters that are not padded with blanks cause unpredictable results. Be sure that all padding is done by inserting trailing blanks. Padding a parameter with leading blanks causes an incorrect value to be passed to the service.
- To omit a parameter, insert a blank enclosed in single quotes (' ') in its place.

  Note: Single quotes show service names and keywords in call invocation examples.
- You must indicate the last parameter in the calling sequence with a '1' as the high-order bit in the last entry of the address list. PL/I, COBOL, Pascal, and FORTRAN call statements automatically generate this high-order bit. In assembler call statements, you must use the VL keyword.

FORTRAN, Pascal, and C

For FORTRAN, Pascal, and C, the general call format for invoking SCLM services from functions by using FLMLNK is:

```
lastrc := FLMLNK(service_name,parameter1,parameter2,...);
```

The parameters for the FORTRAN, Pascal, or C invocation are the same as those shown for the call invocation.

SCLM returns the return code from the specified SCLM service in the FORTRAN, Pascal, or C integer variable specified on the invocation. In these examples, the variable LASTRC is used.

FORTRAN example

For functions written in FORTRAN, pass arguments as FORTRAN variables or literals.

```
INTEGER      LASTRC*4
CHARACTER    SERVIS*8,SCLM_ID*8,GROUP*8
DATA         SERVIS/'DELETE  '/
DATA         SCLM_ID/'SCLM0001'/
DATA         GROUP/'USER1   '/
             ⋮

LASTRC=FLMLNK(SERVIS,SCLM_ID,GROUP,...)
```

For FORTRAN service requests, initialize parameter variables by using literals in assignment statements. You must use previously defined constants in assignment statements.

```
CHARACTER    DELET*8,SERVIS*8
DATA         DELET/'DELETE  '/
   ⋮

SERVIS=DELET
```

Pascal example

```
CONST
  SERVICE = 'DELETE  ';
  SCLM_ID = 'SCLM0001';
  GROUP   = 'USER1   ';
  ⋮

LASTRC := FLMLNK(SERVICE,SCLM_ID,GROUP,...);
```

For service calls in Pascal, initialize parameter variables by using literals in assignment statements:

```
SERVICE:='DELETE  ';
```

C example

In C programs, include the following declare statements and compiler directives:

```
  #pragma linkage(flmlnk,OS);
  extern int flmlnk();
```

Example

```
  int retcode;
  char *SERVICE, *SCLMID,*GROUP, ...;
  SERVICE = "DELETE  ";
  SCLMID = "SCLM0001";
  GROUP = "USER1   ";
  ⋮

  lastrc = flmlnk(SERVICE,SCLMID,GROUP,...);
```

PL/I

In PL/I programs, include the following declare statements:

```
DECLARE FLMLNK     /* NAME OF ENTRY POINT          */
        ENTRY
        EXTERNAL   /* EXTERNAL ROUTINE             */
        OPTIONS(   /* NEEDED OPTIONS               */
        ASM,       /* DO NOT USE PL/I DOPE VECTORS */
        INTER,     /* INTERRUPTS                   */
        RETCODE);  /* EXPECT A RETURN CODE         */
```

PL/I example

```
DECLARE SERVICE   CHAR(8)  INIT('DELETE  '),
        SCLM_ID   CHAR(8)  INIT('SCLM0001'),
        GROUP     CHAR(8)  INIT('USER1   '),
        ⋮

CALL FLMLNK(SERVICE,SCLM_ID,GROUP,...);
```

For service calls in PL/I, initialize parameter variables by using literals in assignment statements:

```
SERVICE='DELETE  ';
```

COBOL

For functions written in COBOL, arguments can be passed as literals, or variables, as in the following example:

COBOL example

```
WORKING-STORAGE TYPE.
   77  SERVIS      PICTURE X(8) VALUE 'DELETE  '.
   77  SCLMID      PICTURE X(8) VALUE 'SCLM0001'.
```

```
   77  GROUP       PICTURE X(8) VALUE 'USER1   '.
   ⋮

PROCEDURE DIVISION
   CALL 'FLMLNK' USING SERVIS SCLMID GROUP  ...  .
```

For service calls in COBOL, initialize parameter variables by using literals in assignment statements:

```
MOVE 'DELETE  ' TO SERVIS.
```

Selecting a service from the FLMCMD Services Menu

To display the SCLM FLMCMD Services Menu panel, select Easy Cmds (option 6A) from the SCLM main menu. This panel lists the available FLMCMD services. When you select an option from this list, ISPF displays a panel that provides data entry fields for the parameters associated with the selected service.

```
  Menu  Utilities  Help
──────────────────────────────────────────────────────────────────────────
                          SCLM FLMCMD Services Menu
                                                                More:     +
1  ACCTINFO  Retrieve accounting information
2  AUTHCODE  Retrieve or set authorization code for selected members
3  BUILD     Build a member
4  DBUTIL    Create reports and tailored data sets against an
               SCLM database
5  DELETE    Delete database components
6  DELGROUP  Delete database components from group
7  DSALLOC   Allocate data sets for group or type
8  EDIT      Edit a member of a controlled library
9  EXPORT    Extract SCLM accounting information
10 GETBLDMP  Retrieve build map information
11 IMPORT    Incorporate exported data into the hierarchy
12 LOCK      Lock a member or assign an access key
13 MIGRATE   Register the contents of a library with SCLM
14 NEXTGRP   Find the name of the next group in a hierarchy
15 PROMOTE   Promote a member from one library to another
16 RPTARCH   Create an architecture report
17 SAVE      Lock, parse, and store a member
Option ===> ____________________________________________________________
 F1=HELP      F2=           F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 183. SCLM FLMCMD Services Menu panel

For details about the specific service panels, see the relevant service description in Chapter 19, “SCLM services,” on page 379.

Automatic allocation of output data sets

Some services support the specification of ddnames for output such as reports and messages. If specified, these are automatically allocated before the service is invoked.

You can specify a data set prefix to be used when assigning names for the data sets allocated for a service. You do this through the Options —► Set Services Data Sets Prefix action bar choice. This option is available on the panels for each service that allocates data sets and on the SCLM FLMCMD Services Menu.

The format of the name for an automatically allocated data set is:

where:

prefix
Data set prefix specified via Options —► Set Services Data Sets Prefix, or *userid*.*projdef* if no data set prefix is specified.

member
Optionally, the name of the member specified on the service panel. Only applicable for the following services:

```
ACCTINFO    GETBLDMP    VERDEL
BUILD       PROMOTE     VERINFO
EDIT        RPTARCH     VERRECOV
```

service
The name of the service.

dstype
One of the following values, indicating the type of data set:

MSGS
(messages data set)

REPT
(report data set)

LIST
(listing data set)

EXIT
(exit data set)

TAIL
(tailoring data set)

screenid
S followed by the logical screen identifier stored in the ISPF variable ZSCREEN. For example, on the first logical screen *screenid* is S1.

Example 1: if user FRED uses the BUILD service to build member MEM1 in project PROJ1 with no data set prefix specified, this name will be used for the report data set if a report data set DD is specified: FRED.PROJ1.MEM1.BUILD.REPT

Example 2: if a data set prefix of WORK.JOB1 is specified, this data set name will be used for the BUILD report data set: WORK.JOB1.MEM1.BUILD.REPT

Entering a command to invoke a specific service panel

From any ISPF panel, you can invoke one of the FLMCMD services by entering TSO FLMCMD *srvname* on the command line. For example, entering the following command invokes the AUTHCODE service:

```
TSO FLMCMD AUTHCODE
```

When you enter this command, ISPF displays a panel that provides data entry fields for the parameters associated with the named service.

For details about the specific service panels, see the relevant service description in Chapter 19, “SCLM services,” on page 379.

Types of parameters

The various types of parameters discussed in this section include DDNAME, Character, and Pointer parameters.

DDNAME parameters

SCLM services send output to data sets associated with the ddnames you provide in the parameters passed to the service. You should allocate ddnames with the attributes specified in the parameter

descriptions. If you use different attributes to allocate the ddnames, SCLM accesses the data set using the attributes specified, but the format of the resulting file might not be usable.

As part of the processing for several of its services, SCLM updates partitioned data sets. For instance, the BUILD service copies compiler-produced object modules into an SCLM-controlled object partitioned data set. To eliminate the risk of corrupting a partitioned data set, allocate the data set with DISP=OLD.

Character parameters

Left-justify all character input parameters (character strings) to the SCLM services. Left-justify all character output parameters (character strings) from the SCLM services. Make the calling program buffer the length specified in the service descriptions. Failure to provide a buffer of the proper size causes unpredictable results.

Selection parameters

You can use patterns to specify a variety of acceptable values for the accounting information fields. A pattern consists of alphanumeric characters and three special characters: an asterisk (*), a logical NOT symbol (¬), and an equal sign (=).

Use an asterisk to match any string of characters including the null string. You can use it more than once.

Use the logical NOT symbol (¬) to negate the result of a match with the pattern. You can specify it only once. The logical NOT symbol is removed from the pattern before a match is attempted. Therefore, the position of the logical NOT symbol within the pattern is not significant.

Use an equal sign (=) to indicate all groups that are at the same layer in the hierarchy as the group you specify. An equal sign can only be specified once in the pattern.

Use the equal sign only in the group field. Do not use the equal sign in conjunction with other wildcard characters. If you use the equal sign, you must specify a valid group name. The name specified is taken literally.

Note: Do not use an equal sign (=) as the first character in a pattern because it is a special character in ISPF.

Use the patterns shown in Table 29 on page 367 to select accounting information.

Table 29. Pattern Examples

Pattern	Match
AB*Z	ABZ,ABCZ,ABCZYZ,ABCABZ
¬AB*Z	ABC,XABZ,ABZX
*AB*Z	ABZ,XABZ,ABCABZ,ABCZ,ABCZYZ
DEV1=	DEV1,DEV2
STAGE1=	STAGE1,STAGE2

Note: See Figure 66 on page 141 for an illustration of the hierarchy represented in the last two rows.

Pointer parameters

All pointer parameters to the SCLM services provide a fullword address to a predefined array or record structure.

The SCLM services use four pointer parameters:

$msg_array
(message array)

$acct_info
(accounting information)

$stats_info
(statistical information)

$list_info
(list information array)

For Pascal declarations of the services program invocations, see Chapter 20, “Sample programs using SCLM services,” on page 477.

Note: When creating programs that use SCLM services, the developer must be careful to manipulate the memory for pointer parameters correctly.

Input Parameters
The program calling the SCLM service must allocate the memory for the pointer parameter (one word) and the memory for the structure.

Output Parameters
The program calling the SCLM service must allocate the memory for only the pointer parameter (one word). If the information in the output structure will be referenced later in the program then the information in the structure must be copied to the program's local storage **before** the next call to an SCLM service. SCLM allocates and deallocates the memory where the output structure is stored.

For example, if you want to pass the $list_info array from the PARSE service to the STORE service, you must first copy the $list_info array to a local memory buffer. Then you must pass the local buffer pointer to the STORE service.

For examples of copying the $list_info array and the $stats_info record, see Chapter 20, “Sample programs using SCLM services,” on page 477.

Pointer parameter descriptions

This section describes each of the four pointer parameters:

$msg_array

A pointer to an array of messages SCLM services produce. Each record in the message array is 80 bytes. An END record denotes the end of the message array. Figure 184 on page 368 shows the contents of a message array with one message consisting of two message lines.

```
Record 1: FLM80500 - ACCESS KEY INCORRECT, ACCESS KEY: WRONG_KEY
Record 2:            GROUP: USER1, TYPE: SOURCE, MEMBER: FLM01MD1
Record 3: END
```

Figure 184. $msg_array Contents

$acct_info

A pointer to a record containing the static portion of an accounting record. The following describes the format of the record fields in the order in which they appear. For additional information on record field contents, refer to “Accounting record” on page 164.

The following fields contain data common to all members:

Field
Contents

acct_group
8 characters

acct_type
8 characters

acct_member
8 characters

SCLM_version
2 characters ('60 or 70')

accounting_status
1 character:

E
Editable

N
Noneditable

L
Lockout

I
Initial

X
NOPROM-R (REBUILD)

Y
NOPROM-N (REBUILD)

change_date
8 characters (YYYYMMDD format)

change_time
6 characters (HHMMSS format)

change_group
8 characters

change_userid
8 characters

------
3 characters (space for alignment)

member_version
Fullword integer

language
8 characters

authorization_code
8 characters

authorization_code_change
8 characters

access_key
16 characters

creation_date
8 characters (YYYYMMDD format)

creation_time
6 characters (HHMMSS format)

map_date
8 characters (YYYYMMDD format)

map_time
6 characters (HHMMSS format)

predecessor_date
8 characters (YYYYMMDD format)

predecessor_time
6 characters (HHMMSS format)

promote_date
8 characters (YYYYMMDD format)

promote_time
6 characters (HHMMSS format)

promote_userid
8 characters

db_qual
8 characters

The following fields are blank unless the accounting_status is N. Each field is 8 characters.

- translator_version
- map_name
- map_type
- language_version

The following fields contain statistical data for a member. Each field is a fullword integer.

- total_lines
- comment_lines
- non_comment_lines
- blank_lines
- total_stmts
- comment_stmts
- non_comment_stmts
- number_of_user_entries
- number_of_includes
- reserved_field
- number_of_changecodes
- number_of_cus

The fields preceded by an asterisk refer to statistics that the SCLM-supplied parsers do not collect.

$stats_info

A pointer to a record containing a member's statistical information. Each of the fields is a fullword integer. For a description of the record field contents, see “Statistics” on page 166. The following describes the format of the record fields.

- total_lines
- comment_lines
- non_comment_lines
- blank_lines
- * prolog_lines
- total_stmts
- comment_stmts
- * control_stmts
- * assignment_stmts
- non_comment_stmts

The fields preceded by an asterisk refer to statistics that the SCLM-supplied parsers do not collect.

$list_info

A pointer to an array of records containing the dynamic portion of an SCLM accounting record. The array contains records detailing a member's include, change code and user entry information. Each record in the array is 228 bytes.

Some of the SCLM services place restrictions on the data that you can specify with this parameter. See the description of each service to determine if it restricts the $list_info parameter data.

The records in the array contain two fields. The first field is 4 characters and indicates the record type. Valid record type values are:

Record Type
Description

END
Indicates the end of the array

INCL
Indicates an include

INCS
Indicates an include with an include-set name

COMP
Indicates the name of an include from the COMPOOL include set

CODE
Indicates a change code

USER
Indicates user data

EXTD
Indicates external dependencies.

The second field varies depending on the record type. For the following discussion, "member" refers to the member whose array contains dynamic accounting record information.

The following table describes the data in the second field for each record type:

Record Type
Description

END
No data

INCL
Member name (8 characters) upon which the "member" has an include dependency.

INCS
A record containing two parts. The first 8 bytes contain the include name; the next 8 bytes contain the include-set name.

Include sets are used when different types are to be searched for the includes. For example, an include set of INCLUDE could be used for includes of source code and an include set of SQL could be used for SQL declarations. The include-set name returned by a parser must match the name of an include set in the language definition that included that parser. Include sets are defined using the FLMINCLS macro.

Because the include-set name is then associated with a ddname allocation for the translator, there are usually no more include-set names returned by the parser than there are input ddnames supported by the translators in the language definition.

Use the INCS record to record dependencies when an include-set name is to be associated with the dependency. Use the INCL record to record a dependency when the dependency is to be associated with the default include set. Do not use both INCL and INCS records for the same dependency name unless two different include dependencies are to be recorded for the same member name.

An INCS record with blanks for the include-set name is the same as an INCL record for that dependency.

COMP
Indicates an include in the COMPOOL include-set $list_info entries returned by SCLM will always use the INCS record type to return information for includes in the COMPOOL include set. The preferred method of recording dependencies in the COMPOOL include set is to use INCS records. This record type is available for compatibility purposes only.

CODE
A record detailing a change code associated with the "member". The total record length is 22 bytes. The record contains a change code (8 characters), a change code date stamp (8 characters, YYYYMMDD format), and a change code time stamp (6 characters, HHMMSS format). The change code value will be translated to uppercase before it is passed to the SCLM service.

USER
User data (128 characters) associated with the "member".

EXTD
A record that describes an external dependency for an SCLM-controlled member. This record contains the following information:

group
Name of the SCLM group that is equivalent to the group where the external dependency resides (8 characters)

type
Name of type (8 characters)

name
Name of the external dependency (43 characters)

date/time
Date and time in SCLM format when the external dependency was last changed (14 characters: date in format YYYYMMDD, time in format HHMMSS).

Note: The SAVE service restricts the $list_info record type to CODE and END. SCLM deletes all existing user data records if you use the SAVE service.

Figure 185 on page 372 shows the contents of a list information array. Two change codes (PR1234 on 12/16/93 at 12:01:33 and CR000032 on 1/4/94 at 00:53:16) and a user entry indicating a customized member are associated with the "member".

```
Record 1: CODEPR1234  19931216120133
Record 2: CODECR000032199401040053 16
Record 3: USERTEST MEMBER - CUSTOMIZED
Record 4: END
```

Figure 185. $list_info Contents

ISPF variables

Some SCLM services use ISPF variables to communicate information with the caller. All variables contain character data. Integer data is converted to character format. The following table lists the ISPF variables which are used:

Table 30. ISPF variables used in SCLM services

Variable	Max Size	Services	Description
ZLOCKDSN	44	LOCK	Data set name for the member at the lock group
ZSAACKEY	16	VERINFO, ACCTINFO	Access key (see LOCK and UNLOCK services)

Table 30. ISPF variables used in SCLM services (continued)

Variable	Max Size	Services	Description
ZSAASTMT	8	VERINFO, ACCTINFO	Parser statistic - number of assignment statements
ZSAAUTH	8	VERINFO, ACCTINFO	Authorization code
ZSAAUTHC	8	VERINFO, ACCTINFO	Authorization code change
ZSABDATE	8	VERINFO, ACCTINFO	Baseline (predecessor) date
ZSABDAT4	10	VERINFO, ACCTINFO	Baseline (predecessor) date with a 4-character year
ZSABLINE	8	VERINFO, ACCTINFO	Parser statistic - number of blank lines
ZSABTIME	8	VERINFO, ACCTINFO	Baseline (predecessor) time
ZSACCCNT	8	VERINFO, ACCTINFO	Number of change codes for the member
ZSACDATE	8	VERINFO, ACCTINFO	SCLM creation date
ZSACDAT4	10	VERINFO, ACCTINFO	SCLM creation date with 4-character year
ZSACLINE	8	VERINFO, ACCTINFO	Parser statistic - number of comment lines
ZSACSTMT	8	VERINFO, ACCTINFO	Parser statistic - number of comment statements
ZSACTIME	8	VERINFO, ACCTINFO	SCLM creation time
ZSACUCNT	8	VERINFO, ACCTINFO	Number of ADA Compilation units
ZSADSN	44	VERINFO, ACCTINFO	Physical data set name for the group and type
ZSAGRP	8	ACCTINFO	SCLM group name
ZSAINCNT	8	VERINFO, ACCTINFO	Number of includes for the member
ZSALANG	8	VERINFO, ACCTINFO	SCLM language name for the member
ZSALDATE	8	VERINFO, ACCTINFO	Date the member was last changed
ZSALDAT4	10	VERINFO, ACCTINFO	Date, with a 4-character year, that the member was last changed
ZSALGRP	8	VERINFO, ACCTINFO	Group where the member was last changed

Table 30. ISPF variables used in SCLM services (continued)

Variable	Max Size	Services	Description
ZSALSTMT	8	VERINFO, ACCTINFO	Parser statistic - number of control statements
ZSALTIME	8	VERINFO, ACCTINFO	Time the member was last changed
ZSALUSER	8	VERINFO, ACCTINFO	Userid that last changed the member
ZSAMBR	8	ACCTINFO	SCLM member name
ZSAMDATE	8	VERINFO, ACCTINFO	Date of the build map that generated the member or if the member is not generated this is the last change date
ZSAMDAT4	10	VERINFO, ACCTINFO	Date, with a 4-character year, of the build map that generated the member or if the member is not generated this is the last change date
ZSAMMBR	8	VERINFO, ACCTINFO	Name of the build map that generated the member or blank if not a generated member
ZSAMTIME	8	VERINFO, ACCTINFO	Time of the build map that generated the member or if the member is not generated this is the last change time
ZSAMTVER	8	VERINFO, ACCTINFO	Version of the translator that generated the member
ZSAMTYPE	8	VERINFO, ACCTINFO	Type of the build map that generated the member or blank if not a generated member
ZSANLINE	8	VERINFO, ACCTINFO	Parser statistic - number of non-comment lines
ZSANSTMT	8	VERINFO, ACCTINFO	Parser statistic - number of non-comment statements
ZSAPDATE	8	VERINFO, ACCTINFO	Date the member was promoted from a lower group or zeros if no promote was done to get the member into the current group
ZSAPDAT4	10	VERINFO, ACCTINFO	Date, with a 4-character year, the member was promoted from a lower group or zeros if no promote was done to get the member into the current group
ZSAPLINE	8	VERINFO, ACCTINFO	Parser statistic - number of prolog lines
ZSAPTIME	8	VERINFO, ACCTINFO	Time the member was promoted from a lower group or zeros if no promote was done to get the member into the current group
ZSAPUSER	8	VERINFO, ACCTINFO	Userid last promoting the member from a lower group or blank if no promote was done to get the member into the current group
ZSASTAT	8	VERINFO, ACCTINFO	Status of the accounting record. Possible values are: EDITABLE, NON-EDIT, LOCKOUT, INITIAL,ERROR, NOPROM-N, and NOPROM-R
ZSATLINE	8	VERINFO, ACCTINFO	Parser statistic - total number of lines

Table 30. ISPF variables used in SCLM services (continued)

Variable	Max Size	Services	Description
ZSATSTMT	8	VERINFO, ACCTINFO	Parser statistic - total number of statements
ZSATYPE	8	ACCTINFO	SCLM Type
ZSAUECNT	8	VERINFO, ACCTINFO	Number of user entries for the member
ZSAVER	8	VERINFO, ACCTINFO	Version number of the member
ZSBMGRP	8	GETBLDMP	Build map group name
ZSBMTYP	8	GETBLDMP	Build map type name
ZSBMMEM	8	GETBLDMP	Build map member name
ZSBKWRD	8	GETBLDMP	Build map entry type
ZSBGRP	8	GETBLDMP	Build map entry group name
ZSBMEM	8	GETBLDMP	Build map member name
ZSBTYPE	8	GETBLDMP	Build map member type
ZSBDATE	8	GETBLDMP	Build date (in YYYYMMDD format)
ZSBTIME	8	GETBLDMP	Build time (as HHMMSS padded with blanks)
ZSBVER	8	GETBLDMP	Build version (integer)
ZSBLINE	72	GETBLDMP	Build map unformatted data line
ZSCIACTF	32 KB	SCLMINFO	Accounting file names for the project
ZSCIAUT	32 KB	SCLMINFO	Authorization code information for the project
ZSCIGRP	32 KB	SCLMINFO	Group information for the project
ZSCILANG	32 KB	SCLMINFO	Language information for the project
ZSCINPAT	32 KB	SCLMINFO	Data set name patterns information for the project
ZSCIPDEF	8	SCLMINFO	Alternate specified by the user
ZSCIPROJ	8	SCLMINFO	Project specified by the user
ZSCISVER	8	SCLMINFO	SCLM version ID for the project
ZSCTIME	8	VERINFO, ACCTINFO	Last time a change code was assigned to a member
ZSCITMST	14	SCLMINFO	Timestamp (date and time when the project was generated)
ZSCITYPE	32 KB	SCLMINFO	Type information for the project
ZSDNAME	110	VERINFO, ACCTINFO	Name of an ADA compilation unit
ZSDTYPE	8	VERINFO, ACCTINFO	Type of an ADA compilation unit. Possible values are: SPEC, BODY, and XREF
ZSIISET	8	VERINFO, ACCTINFO	Include-set name for an include

Table 30. ISPF variables used in SCLM services (continued)

Variable	Max Size	Services	Description
ZSIMBR	8	VERINFO, ACCTINFO	An include for a member
ZSUENTRY	128	VERINFO, ACCTINFO	Data from a user entry
ZSUNUM	8	VERINFO, ACCTINFO	Number of the user entry
ZSVACTN	8	VERINFO	Action generating the audit record. Possible values are: PUT and PURGE
ZSVAMBR	8	VERINFO	SCLM member name for action taken
ZSVCFMT	8	VERINFO	Current format of the version member. Possible values are: DELTA, FULL, and AUDIT
ZSVDATE	8	VERINFO	Date the audit record was generated.
ZSVDAT4	10	VERINFO	Date, with a 4-character year, that the audit record was generated.
ZSVFMSG	8	VERINFO	SCLM message id if versioning of the member failed.
ZSVGRP	8	VERINFO	SCLM group name for action taken
ZSVLDATE	8	VERINFO	Last change date of the member
ZSVLDAT4	10	VERINFO	Last change date, with 4-character year, of the member
ZSVLTIME	8	VERINFO	Last change time of the member
ZSVMBR	8	VERINFO	Member in the versioning PDS containing the version of the member or blank if only auditing was performed
ZSVPDS	44	VERINFO	The versioning PDS containing the version of the member
ZSVRESLT	8	VERINFO	Result of the versioning action. Possible values are: ATTEMPT, COMPLETE, and FAILED
ZSVRFMT	8	VERINFO	Requested format of the version. Possible values are: DELTA, FULL, and AUDIT
ZSVSDATE	8	VERINFO	SCLM change date for the member for which a version was requested.
ZSVSDAT4	10	VERINFO	SCLM change date, with 4-character year, for the member for which a version was requested.
ZSVSERV	8	VERINFO	SCLM service generating the audit record. Possible values are: BLDDEL, BUILD, DELETE, FREE, IMPORT, LOCK, EXT LIB, PROMOTE, STORE, UPTATHCD, UPTCHGCD, UNLOCK, and UPTUENTY
ZSVSTIME	8	VERINFO	SCLM change time of the member.
ZSVTIME	11	VERINFO	Time the audit record was generated.
ZSVTYPE	8	VERINFO	SCLM type for action taken
ZSVUSER	8	VERINFO	Userid performing the service which generated the audit record.

SCLM service return codes

Each service returns a numeric code, called a *return code*, indicating the results of the operation. Here are possible return codes:

0
Indicates successful completion. SCLM may generate messages.

2
Indicates successful completion. No action taken.

4
Indicates a warning condition. SCLM may generate messages.

8
Indicates an error condition. SCLM generates messages detailing the error.

12
Indicates a severe error condition. SCLM generates messages detailing the error.

Return codes and their meanings vary for each service and are listed with each service description. In addition to these return codes, the FLMCMD and FLMLNK interfaces each generate return codes.

For command invocation, SCLM returns the code in the CLIST variable. For call invocation, SCLM returns the code in registers 15 and 0. When using the FILE format of FLMCMD command invocation, SCLM sets the return code to the maximum return code encountered while processing the command data set.

Programs coded in Pascal or FORTRAN can examine the return code by using an integer variable, such as lastrc, in the following example:

```
lastrc := FLMLNK(service_name,parameter1,parameter2,...);
```

Programs coded in PL/I can examine the return code by using *PLIRETV*, a built-in function. You need the following declare statements:

```
DECLARE FLMLNK  EXTERNAL ENTRY OPTIONS(ASM INTER RETCODE);
DECLARE PLIRETV BUILTIN;
```

Programs coded in COBOL can examine the return code by using *RETURN-CODE*, a built-in variable.

FLMCMD command processor return codes

Possible return codes are:

12
Maximum application ID limit exceeded. FLMCMD has attempted to initialize an SCLM session, but the maximum number of SCLM sessions have already been started. End one or more of the active sessions and reissue the command.

16
The SCLM table verification failed. The version of the SCLM project definition macros used to compile the specified project definition does not match the version of SCLM being used. Verify that the project definition specified in the line command is correct. If the project definition was specified correctly, contact the project administrator.

20
The multicultural support table verification failed. The version of the NLS table did not match the version of SCLM being used. Contact the project administrator.

24
Unable to load the SCLM table (FLMTABLE). Contact the project administrator.

28
Unable to load the multicultural support table or the SCLM I/O load module (FLMIO24). Contact the project administrator.

FLMLNK call processor return codes

Possible return codes are:

20
Severe error condition. SCLM does not produce messages because the SCLM ID is not valid.

24
Severe error condition. SCLM does not produce messages because the SCLM services have not been initialized. See "START—Generate an Application ID for a Services Session" on page 458 for information about initializing an SCLM services session.

32
Severe error condition. An invalid parameter list was passed to the requested service.

34
Severe error condition. An invalid service was requested.

36
Severe error condition. The version of the FLMLNK subroutine does not match the version of the SCLM services module.

40
Severe error condition. Contact IBM service.

Return codes and their meanings vary for each service and are listed with each service description.

SCLM service messages

SCLM services issue two types of messages:

FLMMSGS
SCLM uses the ddname FLMMSGS for special services messages such as a completion status or return code message, and for error messages associated with the specified service parameters. These messages are usually routed to the default output device, such as your terminal. In order to suppress or re-route these messages, allocate the FLMMSGS ddname before invoking the SCLM service.

Service Specific Messages
Many of the services have parameters for handling messages. There are three types of message parameters:

msg_line
Services that only write one message have a **msg_line** parameter. Define a program variable that is 80 characters to hold the contents of this message line. This parameter only applies to services called through the FLMLNK interface.

$msg_array
Some services that can produce more than one message have a **$msg_array** parameter. Define program storage as described in "DDNAME parameters" on page 366 to store the service messages. The $msg_array is available only from services invoked through the FLMLNK interface.

ddname
Many of the services offer a **ddname** parameter which you can allocate to a file that stores the messages. Information for allocating the ddname is included in the description for each applicable service. If you leave the ddname parameter blank, the messages go to the default output device, for example, your terminal.

Chapter 19. SCLM services

Each service description includes an example of its use in the command procedure format and the Pascal call format. Default settings for each service call are shown in the command procedure format section for each service; the default values are underscored. Call invocations do not have defaults because some value must be specified for each parameter; a blank is identified for each parameter that will translate a blank into a default value.

See Chapter 20, “Sample programs using SCLM services,” on page 477 for an example of service invocations and declarations coded in Pascal.

SCLM service descriptions

This section contains information about the services available for SCLM.

- “ACCTINFO—Retrieve Accounting Information” on page 380
- “AUTHCODE—Retrieve or Set Authorization Code for Selected Members” on page 384
- “BUILD—Build a Member” on page 388
- “DBACCT—Retrieve Accounting Records for a Member” on page 394
- “DBUTIL—Generate a Tailored Output Data Set and Report” on page 395
- “DELETE—Delete Database Components” on page 400
- “DELGROUP—Delete from Group Database Components” on page 402
- “DSALLOC—Allocate Data Sets for Group/Type” on page 407
- “EDIT— Edit a Member of a Controlled Library” on page 410
- “END— End an SCLM Services Session” on page 413
-
- “EXPORT—Extract SCLM Accounting Information for a Group” on page 416
- “FREE—Free an SCLM ID” on page 419
- “GETXDEP—return cross-dependency information” on page 423
- “GETBLDMP—Retrieve Build Map Information” on page 420
- “IMPORT—Import SCLM Accounting Information to Current Project” on page 426
- “IMPORT—Import SCLM Accounting Information to Current Project” on page 426
- “LOCK—Lock a Member or Assign an Access Key” on page 430
- “MIGRATE—Create Accounting for Selected Members” on page 434
- “NEXTGRP— Retrieve the Next Group in an SCLM Hierarchy” on page 438
- “NOPROM—Change Promote Processing” on page 441
- “PARSE—Parse a Member for Statistical and Dependency Information” on page 442
- “PROMOTE—Promote a Member from One Library to Another” on page 444
- “RPTARCH—Generate an SCLM Architecture Report” on page 448
- “SAVE—Lock, Parse, and Store a Member” on page 451
- “SCLMINFO—Return Project Information” on page 455
- “START—Generate an Application ID for a Services Session” on page 458
- “STORE—Store Member Information in an Accounting Record” on page 459
- “UNLOCK—Unlock a Member in a Development Library” on page 462
- “VERDEL—Delete Version and Audit Information” on page 464

© Copyright IBM Corp. 1990, 2019

- “VERHIST—Retrieve Versioned Member Information” on page 466
- “VERINFO—Retrieve Version and Audit Information” on page 468
- “VERRECOV—Recover a Version” on page 471
- “XDEPUPDT—Update Cross-dependency Information” on page 474

Each service description consists of the following information:

Description
A description of the function and operation of the service. This description also refers to other services that you can use with this service.

Each service description shows the formats for:

- Command invocation, for use in a CLIST or REXX command procedure or as a TSO command
- Call invocation from a program module.

Format
The syntax that you use to code the service, showing both command invocation and call invocation.

Because this chapter shows command and call invocation formats in Pascal, statements are ended with a semicolon (;), which is the Pascal convention. You should use the syntax appropriate for your programming language.

Parameters
A description of any required or optional keywords or parameters.

Return Codes
A description of the codes the service returns. For all services, a return code of 12 or higher implies a severe error. This error is usually a syntax error, but it can be any severe error detected when using the services.

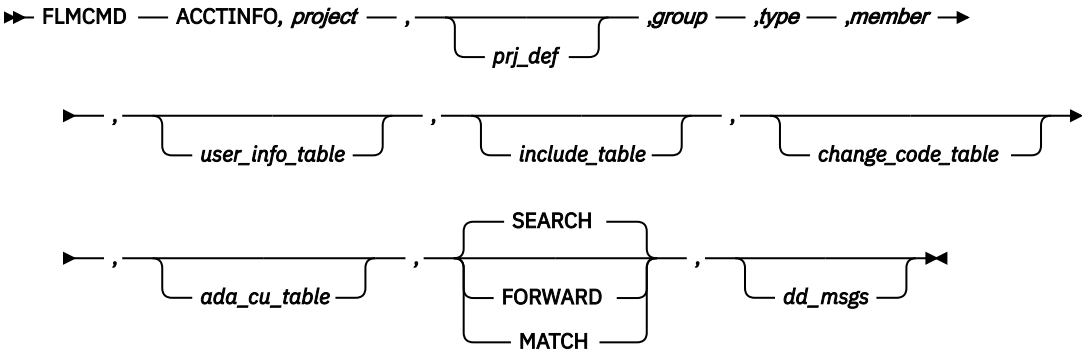
Examples
Sample usage of the service.

FLMLNK requires that the parameters be padded with blanks if the value specified is not as long as the maximum length allowed. Therefore, the examples of call invocations are padded with blanks to the maximum length allowed for each parameter.

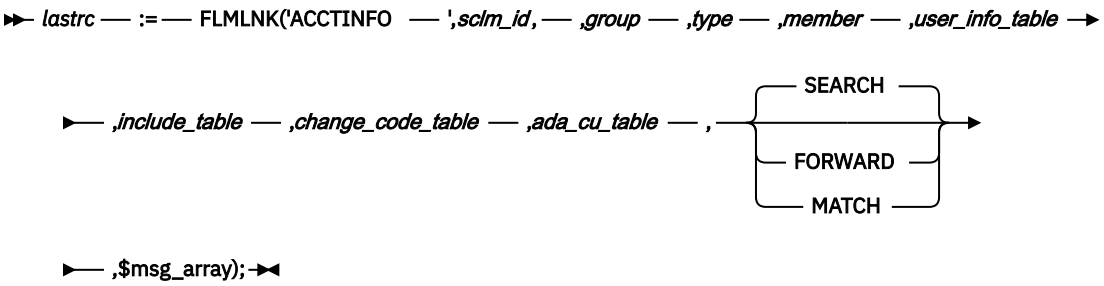
ACCTINFO—Retrieve Accounting Information

The ACCTINFO service retrieves the information about an SCLM-controlled member into ISPF variables and tables. The information is retrieved from the accounting file defined in the project definition for the group specified to the service. The service can search up the hierarchy for the member, search a group for the next matching member, or retrieve the information for a specific member. See “ISPF variables” on page 372 for a list of the variables updated by this service.

Command invocation format



Call invocation format



ISPF interface panel

```
  Menu  SCLM  Utilities  Help
──────────────────────────────────────────────────────────────────────────────
                  SCLM FLMCMD ACCTINFO Service - Entry Panel
                                                                More:     +
SCLM Library:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1____
   Type  . . . . SOURCE__
   Member  . . . ________

Names of open tables for service output:
   User Info . . . . . . . ________
   Includes  . . . . . . . ________
   Change Codes  . . . . . ________
   Compilation Units . . . ________

Search type . . _  1. Search
                   2. Forward
                   3. Match

Command ===> ____________________________________________________________
 F1=HELP      F2=           F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 186. ACCTINFO Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name. It defaults to the project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

group
The group associated with the accounting record. The maximum parameter length is 8 characters.

type
The type associated with the accounting record. The maximum parameter length is 8 characters.

member
The member under SCLM control. The maximum parameter length is 8 characters.

user_info_table
The name of the ISPF table to contain the user entries from the account record. The table must be open before calling the ACCTINFO service. A TBADD will be performed for each user entry in the account record. The maximum parameter length is 8 characters. The following ISPF variables must be used in the table definition in order to have their value stored in the table:

- ZSUNUM - the user entry number
- ZSUENTRY - the user entry data

include_table
The name of the ISPF table to contain the includes from the account record. The table must be open before calling the ACCTINFO service. A TBADD will be performed for each include in the account record. The maximum parameter length is 8 characters. The following ISPF variables must be used in the table definition in order to have their value stored in the table:

- ZSIMBR - the include member name
- ZSIISET - the include set name

change_code_table
The name of the ISPF table to contain the change codes from the account record. The table must be open before calling the ACCTINFO service. A TBADD will be performed for each change code in the account record. The maximum parameter length is 8 characters. The following ISPF variables must be used in the table definition in order to have their value stored in the table:

- ZSCCODE - the change code
- ZSCDATE - the change code date
- ZSCDAT4 - the change code date in 4-character date format
- ZSCTIME - the change code time

ada_cu_table
The name of the ISPF table to contain the ADA compilation units from the account record. The table must be open before calling the ACCTINFO service. A TBADD will be performed for each ADA compilation unit in the account record. The maximum parameter length is 8 characters. The following ISPF variables must be used in the table definition in order to have their value stored in the table:

- ZSDNAME- the ADA compilation unit name
- ZSDTYPE - the ADA compilation unit type

SEARCH|FORWARD|MATCH
SEARCH indicates that SCLM is to look up the hierarchy to find the accounting record if it does not exist at the specified group. This is the default.

MATCH indicates to check only the specified group for the accounting record.

FORWARD indicates that if the member and type names do not exactly match an accounting record the information from the next accounting record in the group is to be returned.

To retrieve all of the accounting records within a group use FORWARD and start with the member and type names set to all blanks. If an accounting record is found increment the last character of the member name before calling the ACCTINFO service again. Repeat this process until the service indicates that no record was found.

The maximum parameter length is 8 characters.

dd_msgs
The ddname indicating the destination of the messages generated by the ACCTINFO service. If you specify a blank ddname, SCLM routes the ACCTINFO messages to the default output device, such as your terminal. Otherwise, before you call the ACCTINFO service, you must allocate the ddname. The following attributes should be used: `RECFM=F, LRECL=80, BLKSIZE=80.` The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

$msg_array
An output parameter pointing to the message array. See “Pointer parameter descriptions” on page 368 for more information about $msg_array. This parameter is used for FLMLNK only.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See “SCLM service messages” on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See “SCLM service return codes” on page 377 for more information.

Possible return codes are:

0
Normal completion. An account record exactly matching the specified criteria was found and the information was stored successfully.

8
Error completion. No account record was found for the specified member.

- If FORWARD was specified, then there are no accounting records for the group which match or follow the specified type and member name.
- If MATCH was specified, then there is not an account record with the specified group, type and member name.
- If SEARCH was specified, then there are no matching account records found when searching up the hierarchy starting from the specified group.

12
Error completion. Refer to the messages for more information.

Example of command invocation format

This service command gets the accounting information from PROJ1.DEV1.SOURCE member COB. The MATCH keyword indicates that the accounting record must be for the COB member of the requested library.

```
parms="PROJ1,PROJ1,DEV1,SOURCE,"
parms=parms||"COB,,,,,MATCH"
address ispexec 'select cmd(flmcmd acctinfo,'parms')'
```

This REXX exec is an ISPF edit macro that you can execute from within an SCLM edit session. The edit macro determines the member and data set name that is being edited, which is then passed to the ACCTINFO service. The exec then writes the contents of several ACCTINFO variables to the screen.

```
/* REXX */
Address isredit 'MACRO'
Address isredit '(MEMBER) = MEMBER'
Address isredit '(DS) = DATASET'
Parse Var ds project'.'group'.'type
'FLMCMD ACCTINFO,'project',,'group','type','member
Say  'acctinfo rc = ' rc
Say  'member name = ' zsambr
Say  'language    = ' zsalang
Say  'change date = ' zsaldat4
```

Example of call invocation format

This service call retrieves the accounting information from the project represented by the sclm_id, the group, type, and member. inc_table is the name of a previously opened table, the ACCTINFO service provides the names of included members and their include set names. The FORWARD keyword means that if an exact match is not found in the given library, the next member in the member list is returned.

```
lastrc := FLMLNK('ACCTINFO', sclm_id, group, pds_type, srch_mem,
              ' ', inc_tabl, ' ', ' ', 'FORWARD ', $msg_array);
```

AUTHCODE—Retrieve or Set Authorization Code for Selected Members

The AUTHCODE service changes or retrieves the authorization code in the SCLM accounting information for members in a library that match a given pattern. The AUTHCODE service does not change the member's statistics or any other value in the accounting record, including the change date and time.

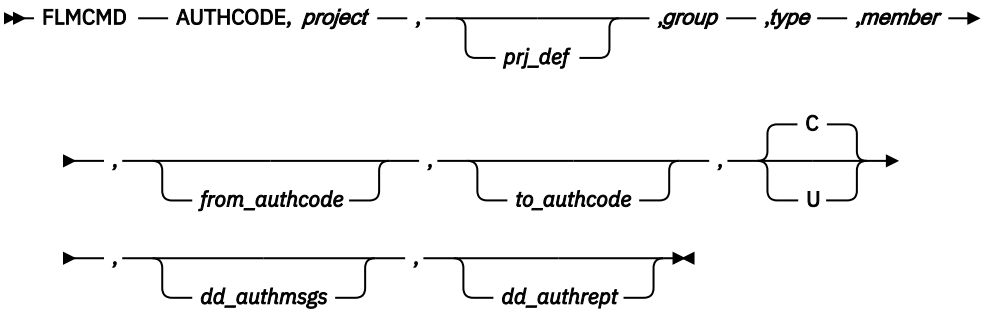
The AUTHCODE service can either set all authorization codes that match a given member and type pattern, or set only those authorization codes that also already have a particular authorization code.

To set the authorization code for all members that match a pattern, leave the `from_authcode` parameter blank.

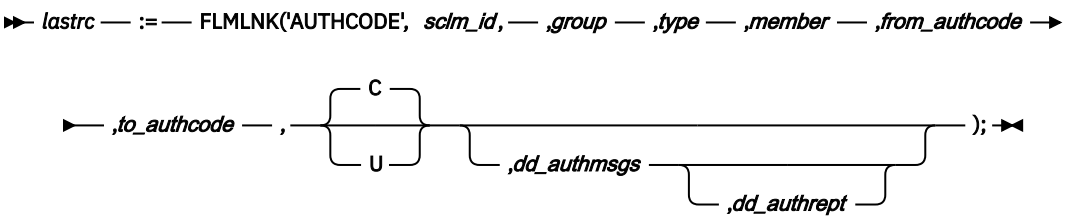
If only members with a certain authorization code are to be set, use the `from_authcode` parameter to tell SCLM to change only those members with the given authorization code.

To retrieve the authorization code, leave both the `from_authcode` and the `to_authcode` parameters blank. The existing authorization code is returned in variable ZSAAUTH if a single member is requested. If a pattern is requested, the existing authorization codes can be retrieved from the AUTHCODE report.

Command invocation format



Call invocation format



ISPF interface panel

```
  Menu  SCLM  Utilities  Help
_____________________________________________________________________
                  SCLM FLMCMD AUTHCODE Service - Entry Panel

SCLM Library:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1____
   Type  . . . . SOURCE__
   Member  . . . ________

From Authorization code _________ (If blank, all authcodes are changed for
                                  change request)
To Authorization code   _________ (Required for change request)

Mode  . . 1  1. Conditional
             2. Unconditional

DD Names for output data sets:
   Error message data set  ________  (Blank to write messages to the terminal)
   Report data set . . . . ________  (Blank to write report to the terminal)
Command ===> ___________________________________________________________
 F1=HELP      F2=          F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 187. AUTHCODE Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters.

prj_def
The project definition name. It defaults to the project name. The maximum parameter length is 8 characters.

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters.

group
The group at which the member's authcode is to be changed. The maximum parameter length is 8 characters.

type
A pattern used to select the types of members whose authcode is to be changed. The maximum parameter length is 10 characters.

member
A pattern used to select the members whose authcode is to be changed. You must specify a valid member name or a valid pattern. The maximum parameter length is 10 characters.

C|U
Indicates whether the AUTHCODE service is to execute conditionally (C) or unconditionally (U). This parameter only applies if the member name is a pattern. If C is selected, processing stops after the first error (default). If U is selected, the service continues to the next member even if an error occurs.

from_authcode
The authorization code to be changed from. If the from_authcode is blank and the to_authcode is given, then all members matching the pattern have the authcode updated. If the from_authcode is not blank, only those members matching the pattern, and whose authcode matches the from_authcode, are updated. The maximum parameter length is 8 characters.

to_authcode
The authorization code to be changed to. If the to_authcode and the from_authcode are both blank, no changes are made. If the from_authcode is given, then the to_authcode is required. The maximum parameter length is 8 characters.

dd_authmsgs
DDNAME of the destination of the AUTHCODE messages. If you specify a blank ddname, SCLM routes the authcode messages to the default output device, such as your terminal. Otherwise, before you call the AUTHCODE service, you must allocate the ddname. The following attributes should be used:

- DISP=MOD
- RECFM=F
- LRECL=80
- BLKSIZE=80.

The maximum parameter length is 8 characters.

dd_authrept
DDNAME of the destination of the AUTHCODE report. If you specify a blank ddname, SCLM routes the authcode report to the default output device, such as your terminal. Otherwise, before you call the AUTHCODE service, you must allocate the ddname. The following attributes should be used:

- RECFM=FBA
- LRECL=80
- BLKSIZE=3120.

The maximum parameter length is 8 characters.

Return codes

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See “SCLM service return codes” on page 377 for more information.

Possible return codes are:

0
Normal completion. Authcode changed or reported successfully.

2
Normal completion. Authcode not changed. One of the following occurred:

- To_authcode = existing authcode (no change needed)
- From_authcode requested does not equal existing authcode (no change wanted)
- Member is not editable.

4
Warning condition. Segment exists at a lower level with an authcode not equal to the `to_authcode`, which could overlay the current segment.

8
Error condition. Invalid type, member, or mode parameter. See the dd_authmsgs for details.

12
Severe error condition. Accounting record not found or severe error.

16
Severe error condition. One of the following occurred:

- Not authorized to update `to_authcode`, access_key mismatch, or not authorized to update data set.
- Verification failed.
- Error updating accounting record.
- Invalid group.

SCLM might not produce messages because there was an error invoking the AUTHCODE module.

Example of command invocation format

This example shows a command interface to the AUTHCODE service, to update the authorization code of SCLM70.USER.SOURCE(A) from base to private.

```
FLMCMD AUTHCODE,SCLM70,SCLM7010,USER,SOURCE,A,BASE,PRIVATE
```

This example shows a command interface to the AUTHCODE service to unconditionally update the authorization code from base to private for all members beginning with FLM in all types of group USER in project SCLM70.

```
FLMCMD AUTHCODE,SCLM70,SCLM7010,USER,*,FLM*,BASE,PRIVATE,U
```

This example selects the FLMCMD AUTHCODE service with no from_authcode or to_authcode. It then gets the authcode value from variable ZSAAUTH in the ISPF SHARED pool.

```
/* rexx exec to retrieve an authcode  */
PARMS = 'SCLM70,SCLM7010,USER,SOURCE'
MEM = 'BES3    '
address ispexec 'select cmd(FLMCMD AUTHCODE,'PARMS','MEM')'
address ispexec 'vget zsaauth shared'
say 'authcode is' zsaauth
```

Call invocation format

This example shows a program call to the AUTHCODE service. The example assumes that the START and INIT services have already completed successfully.

```
CALL FLMLNK('AUTHCODE ',SLMID,'USER ', 'SOURCE ','A ',
    'BASE ','PRIVATE ','C          ', MSGDD,
    REPTDD) RETCODE(R15);
```

This example shows a program call to the AUTHCODE service to unconditionally update the authorization code from 'base' to 'private' for all members beginning with FLM in all types of group USER in project SCLM70.

```
CALL FLMLNK('AUTHCODE ',SLMID,'USER ', '*       ', 'FLM* ',
   ' ','PRIVATE ','U          ', MSGDD,
   REPTDD) RETCODE(R15);
```

Example of an AUTHCODE report

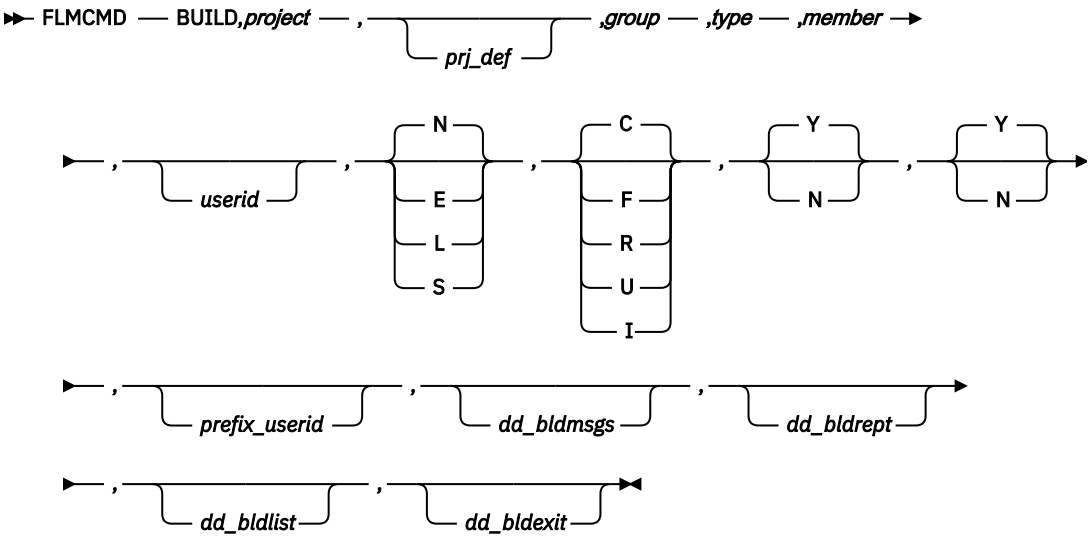
```
********************************************************************************
********************************************************************************
**
**
**                 SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)
```

```
**
**                              AUTHCODE UPDATE REPORT
**
**                                1999/03/08   11:02:15
**
**                            PROJECT:          PDFTDEV
**                            ALTERNATE:        PDFTNEWL
**                            GROUP:            BETH
**                            TYPE:             ARCHDEF
**                            MEMBER:           B*
**                            FROM_AUTHCODE:    PRIVATE
**                            TO_AUTHCODE:      BASE
**                            MODE:             UNCONDITIONAL
*******************************************************************************
*******************************************************************************
                             STARTING    ENDING       COMPLETION
MEMBER     TYPE              AUTHCODE    AUTHCODE      STATUS
--------   --------          --------    --------      -----------
BETHTEST   ARCHDEF           BASE        BASE          NOT_ATTEMPTED
BETH7      ARCHDEF           PRIVATE     BASE          SUCCEEDED
BETH8      ARCHDEF           PRIVATE     BASE          SUCCEEDED
BETH9      ARCHDEF           PRIVATE                   FAILED
BROKE      ARCHDEF           BASE        BASE          NOT_ATTEMPTED
BXC        ARCHDEF                                     NOT_EDITABLE
```

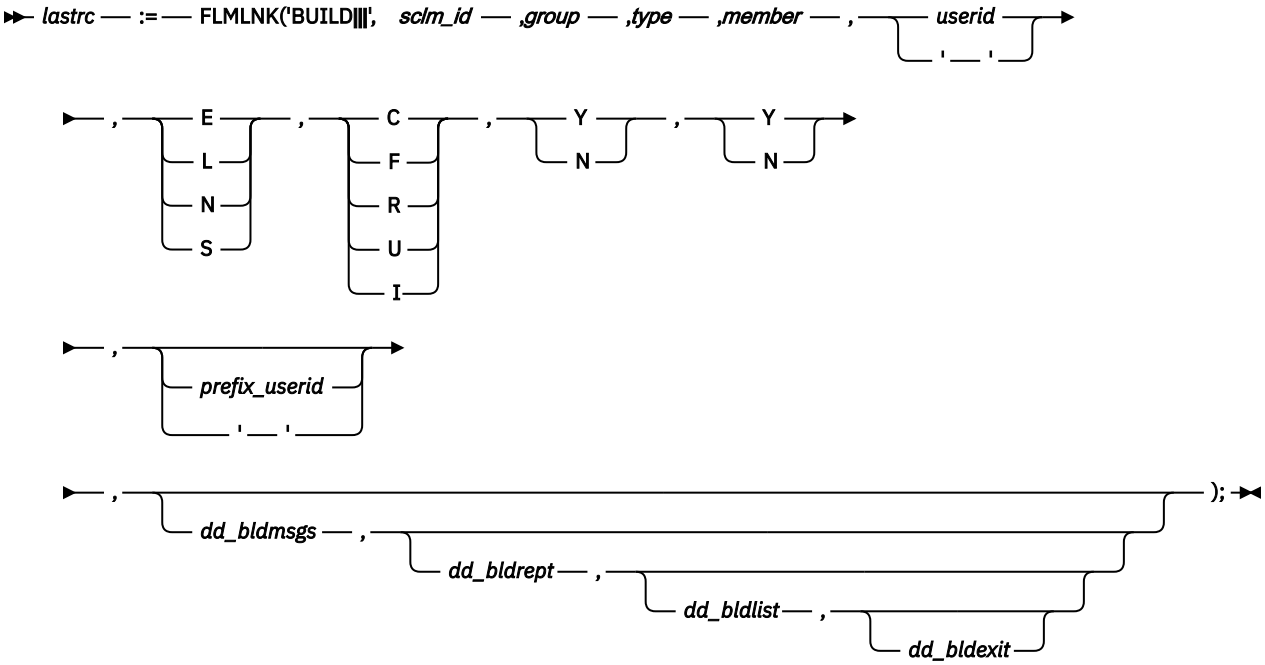
BUILD—Build a Member

The BUILD service compiles, links, and integrates software components according to a project's architecture definition. Before a member is built, the member's dependency information must exist in the project database. For this reason, either the STORE or SAVE service must complete successfully for the member before you call the BUILD service.

Command invocation format



Call invocation format



ISPF interface panel

```
   Menu  SCLM  Utilities  Options  Help
 ──────────────────────────────────────────────────────────────────────
                  SCLM FLMCMD BUILD Service - Entry Panel
Command ===>

SCLM Library:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1___
   Type  . . . . SOURCE__
   Member  . . . ________

Enter "/" to select option
/_ Error listings only
/_ Create build report

Mode  . . 1  1. Conditional       Scope . . 2  1. Limited
             2. Unconditional                  2. Normal
             3. Forced                         3. Subunit
             4. Report                         4. Extended
             5. Information

User id . . . . . . . . . . . . . DOHERTL   (If blank, your user id is used)
Prefix for temporary data sets    ________   (Blank to default to user id)

DD Names for output data sets:
   Messages  . . __________(Blank to write messages to the terminal)
   Report  . . . __________(Blank to write report to the terminal)
   Listings  . . __________(Blank to write list to the terminal)
   Exit  . . . . __________(Blank if no exit or no output required)
```

Figure 188. BUILD Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name used for the build. It defaults to the project parameter. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

sclm_id
An SCLM ID associated with a given project and project definition. The SCLM ID is generated by the INIT service. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

group
The group in which the build occurs. The maximum parameter length is 8 characters.

type
The type containing the member to be built. The maximum parameter length is 8 characters.

member
The member to be built. The maximum parameter length is 8 characters.

userid
The user ID of the person requesting the build. If no value is specified for FLMCMD or a blank (' ') is specified for FLMLNK, it defaults to your TSO prefix or user ID if no TSO prefix has been created. The maximum parameter length is 8 characters.

E|L|N|S
Indicates the build scope (E=extended, L=limited, N=normal, S=subunit). For the FLMCMD interface, the default is N. There is no default for FLMLNK. The maximum parameter length is 24 characters.

C|F|R|U|I
Indicates the build mode (C=conditional, F=forced, R=report, U=unconditional, I=Information). For FLMCMD, the default is C. There is no default for FLMLNK. The maximum parameter length is 24 characters.

Y|N
Y indicates that translator listings are to be copied to the dd_bldlist ddname only if errors occur. N indicates that all translator listings are to be copied to the dd_bldlist ddname. For FLMCMD, the default is Y. There is no default for FLMLNK. The maximum parameter length is 24 characters.

Y|N
Y indicates that a build report is to be produced and routed to the dd_bldrept ddname. N indicates that a build report is not to be produced. For FLMCMD, the default is Y. There is no default for FLMLNK. The maximum parameter length is 24 characters.

prefix_userid
The data set name prefix to be used when locating and cataloging temporary data sets. If no value is specified for FLMCMD or a blank (' ') is specified for FLMLNK, it defaults to the user ID parameter. The maximum parameter length is 17 characters.

dd_bldmsgs
The ddname indicating the destination of the build messages. If you specify a blank ddname, SCLM routes the build messages to the default output device, such as your terminal. Otherwise, before you call the BUILD service, you must allocate the ddname; the following attributes should be used: `RECFM=F, LRECL=80, BLKSIZE=80.` You cannot specify a blank ddname for FLMLNK. The maximum parameter length is 8 characters.

dd_bldrept
The ddname indicating the destination of the build report. If you specify a blank ddname, SCLM routes the build report to the default output device, such as your terminal. Otherwise, before you call the BUILD service, you must allocate the ddname; the following attributes should be used: `RECFM=FBA, LRECL=80, BLKSIZE=3120`. The maximum parameter length is 8 characters.

dd_bldlist
The ddname indicating the destination of the build listings. If you specify a blank ddname, SCLM does not generate the build listings. Otherwise, before you call the BUILD service, you must allocate the ddname; the following attributes should be used: `DISP=MOD, RECFM=VBA, LRECL=137, BLKSIZE=3120`. The maximum parameter length is 8 characters.

dd_bldexit
The ddname indicating the destination of the build user exit data. Specify this parameter only if your project definition defines a build user exit routine. Ask your project manager if your project is using a build user exit routine. If you specify a blank ddname, SCLM routes the build user exit data to NULLFILE. Otherwise, before you call the BUILD service, you must allocate the ddname; the following attributes should be used: `RECFM=FB, LRECL=160, BLKSIZE=3200`. The maximum parameter length is 8 characters.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See "SCLM service messages" on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM service return codes" on page 377 for more information.

Possible return codes are:

0
Normal completion. See the SCLM messages for more information.

4
Warning condition. See the SCLM messages for more information.

8
Error condition. See the SCLM messages for more information.

12
Severe error condition. SCLM does not produce messages because there was an error invoking the build module.

16
Severe error condition. SCLM does not produce messages because it was unable to retrieve SCLM ID information.

Example of command invocation

```
FLMCMD BUILD,PROJ1,,USER1,ARCHDEF,FLM01CMD,,,U,,N
```

This service command builds the FLM01CMD member of the ARCHDEF type in the USER1 group. The project name is PROJ1. The build mode is unconditional and SCLM does not generate a build report. SCLM sends messages and listings to the terminal. All other parameters are defaults.

Example of call invocation

This example shows general syntax. Call invocations are language-specific. See Chapter 20, "Sample programs using SCLM services," on page 477 for specific examples.

```
lastrc := FLMLNK('BUILD   ',                      (* service              *)
                 sclm_id,                         (* SCLM ID              *)
                 'USER1   ',                      (* group                *)
                 'ARCHDEF ',                      (* type                 *)
                 'FLM01CMD',                      (* member               *)
                 '        ',                      (* user ID              *)
                 'N                       ',      (* scope                *)
                 'F                       ',      (* mode                 *)
                 'N                       ',      (* listings             *)
                 'Y                       ',      (* report               *)
                 'PROJECT.WORKFILE ',         (* temp high-level qualifier *)
                 'BLDMSGS ',                      (* dest. of messages    *)
                 'BLDREPT ',                      (* dest. of report      *)
                 'BLDLIST ',                      (* dest. of listings    *)
                 'BLDEXIT ');                     (* exit routine         *)
```

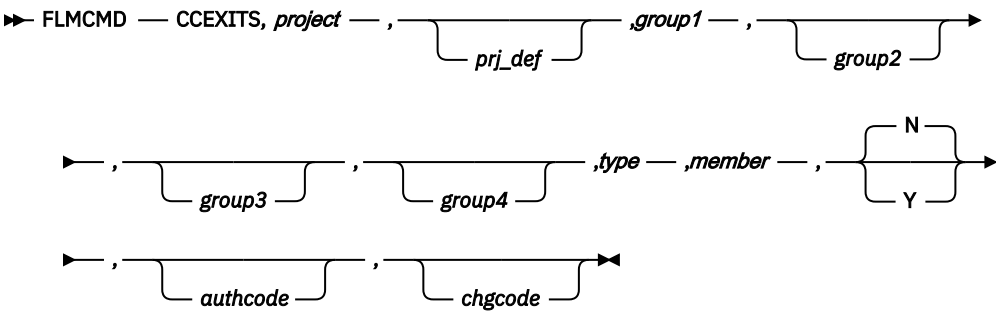
The service call builds the FLM01CMD member of the ARCHDEF type in the USER1 group. The sclm_id parameter contains a valid SCLM ID returned from the INIT service. The build scope is normal and the build mode is forced. SCLM copies all build listings to the build listings data set and generates a build

report. All temporary data sets are allocated with the high-level qualifier of PROJECT.WORKFILE. The ddnames for the messages, report, listings, and user exit data set (BLDMSGS, BLDREPT, BLDLIST, and BLDEXIT, respectively) must be allocated before calling FLMLNK.

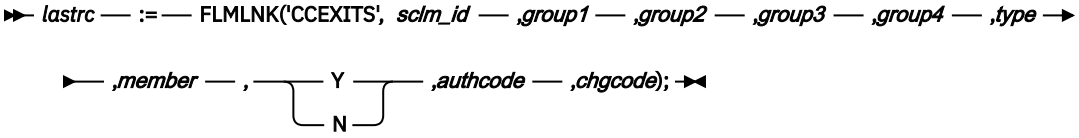
CCEXITS—Run User Exits without Edit

CCEXITS service allows users to invoke the CCVFY and VERCC exits without invoking Edit.

Command invocation format



Call invocation format




```
  Menu  SCLM  Utilities  Options  Help
──────────────────────────────────────────────────────────────────────
                        SCLM FLMCMD CCEXITS Service - Entry Panel

       SCLM Library:
              Project . . . SCLM70__
              Alternate . . ________
              Group . . . . USER___ . . . _TEST___   . . .RELEASE . . ._______
              Type  . . . . SOURCE__
              Member  . . . SOURCEOA_


          Enter "/" to select option
          _  Allocate Hierarchy


          Authorization code  . . PRIVATE  (If blank, the default auth code is used)
          Change code . . . . . . R8EDS__









           Command ===> ____________________________________________________________
            F1=Help      F2=Split     F3=Exit      F7=Backward  F8=Forward   F9=Swap
           F10=Actions  F12=Cancel
```

Figure 189. ISPF Interface Panel

Parameters

project
The project name. The maximum parameter length is 8 characters.

prj_def
The project definition name to be used for edit. It defaults to project. The maximum parameter length is 8 characters.

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters.

group1
The development group at which the member is to be edited. The maximum parameter length is 8 characters.

group2
Name of the second group in the concatenation. The maximum parameter length is 8 characters.

group3
Name of the third group in the concatenation. The maximum parameter length is 8 characters.

group4
Name of the fourth group in the concatenation. The maximum parameter length is 8 characters.

type
The type containing the member to be edited. The maximum parameter length is 8 characters.

member
The member to be edited. The maximum length for this parameter is 8 characters.

Y|N
Y indicates that SCLM will allocate the entire hierarchy, beginning with group1. If Y is selected, group2, group3, and group4 must be blank. N indicates that SCLM will allocate only group1, group2, group3, and group4 (default).

authcode
The authorization code to be used for the edit session. SCLM uses the authorization code for the verification just like the SCLM edit dialog. If you do not supply an authcode SCLM uses the authorization code from the existing member,if the member exists in the hierarchy. The maximum parameter length is 8 characters.

chgcode
The default change code for the edit session. The maximum parameter length is 8 characters.

Return codes

Possible return codes are:

0
Normal completion; user exits ran successfully.

2
Normal completion; no user exits defined.

8
Member not found.

16
Verification error from a user exit routine.

Example of command invocation format

```
FLMCMD CCEXITS,SCLM70,,USER,,,,SOURCE,SOURCE0A,Y
```

This service command invokes CCVFY and VERCC exits if member SCLM70.USER.SOURCE(SOURCE0A) is found in the hierarchy.

Call invocation format

```
CALL FLMLNK('CCEXITS ',SLMID,'USER ',BLNK8,BLNK8,BLNK8,
            'SOURCE ','SOURCE0A ','N
            'PRIVATE ','R8EDS ')
            RETCODE(R15);
```

This service invokes the CCVFY and VERCC exits if member SOURCE0A with type SOURCE is found in group USER.

The example assumes that the START and INIT services have already completed successfully, so that the SLMID value is valid. The authcode is set to "PRIVATE" and the change code is set to "R8EDS".

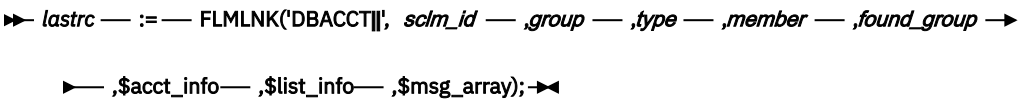
DBACCT—Retrieve Accounting Records for a Member

The DBACCT service retrieves accounting records from the project database and returns the information to you. SCLM retrieves the first occurrence of the accounting record in the hierarchy, starting at the specified group. Accounting records exist for any member for which the LOCK, SAVE, or STORE service completes successfully. For more information about SCLM accounting records, see “$acct_info” on page 368.

Command invocation format

You cannot use command procedures to call this service.

Call invocation format



Parameters

sclm_id
An SCLM ID associated with a given project and project definition. The SCLM ID is generated by the INIT service. The maximum parameter length is 8 characters.

group
The group in which the accounting record search begins. The maximum parameter length is 8 characters.

type
The type containing the accounting record to retrieve. The maximum parameter length is 8 characters.

member
The member whose accounting record is to be retrieved. The maximum parameter length is 8 characters.

found_group
An output parameter that indicates the group in which SCLM finds the first occurrence of the member's accounting record within the hierarchy. The maximum parameter length is 8 characters.

$acct_info
An output parameter pointing to a record containing the static portion of the member's accounting record. See “Pointer parameter descriptions” on page 368 for more details on $acct_info.

$list_info
An output parameter pointing to an array of records containing the dynamic portion of the member's accounting record. See “Pointer parameter descriptions” on page 368 for more details on $list_info.

$msg_array
An output parameter pointing to the message array. See “Pointer parameter descriptions” on page 368 for more information about $msg_array.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See “SCLM service messages” on page 378 for more information.

Other return codes might be produced by the FLMLNK processor. See “SCLM service return codes” on page 377 for more information about these.

Possible return codes are:

0
Normal completion.

4
Warning condition. SCLM could not find the accounting record.

8
Error condition. The $msg_array parameter contains the error message associated with this condition.

Example of call invocation

This example shows general syntax. Call invocations are language-specific. See Chapter 20, “Sample programs using SCLM services,” on page 477 for specific examples.

```
lastrc := FLMLNK('DBACCT  ',       (* service                        *)
                 sclm_id,          (* SCLM ID                        *)
                 'USER1   ',       (* group                          *)
                 'SOURCE  ',       (* type                           *)
                 'FLM01MD1',       (* member                         *)
                 found_group,      (* group found                    *)
                 $acct_info,       (* accounting information pointer *)
                 $list_info,       (* list information pointer       *)
                 $msg_array);      (* message array pointer          *)
```

This service call returns the first occurrence of the accounting record for the FLM01MD1 member of the SOURCE type beginning in the USER1 group. The sclm_id parameter contains a valid SCLM ID returned from the INIT service. SCLM returns all messages produced via the $msg_array.

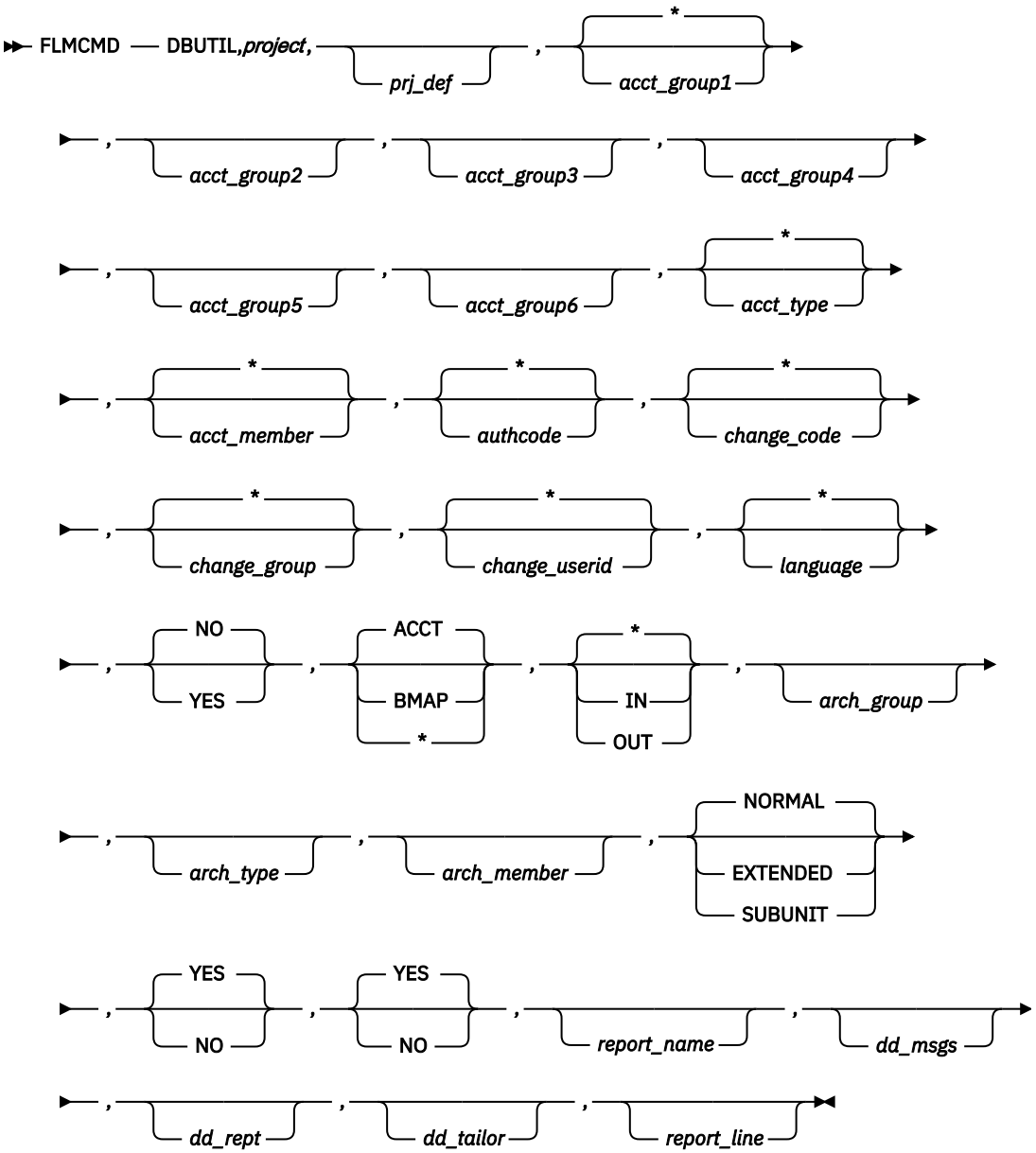
DBUTIL—Generate a Tailored Output Data Set and Report

The DBUTIL service retrieves information from the project database and creates tailored output and a report. SCLM generates the tailored output in the format you specify. It also describes the contents of the project database based on the selection criteria you supply. You can use the tailored output as input to future FLMCMD command invocations (using the FILE format of FLMCMD) or as input to other project-defined processors.

If you use the FILE format of FLMCMD to call the DBUTIL service, you can save the input parameters in a data set, then use the data set for future invocations of the DBUTIL service. See “Using the FLMCMD file format” on page 360 for details on using the FILE format of FLMCMD.

The report indicates the contents of the project database based on the selection criteria you supply to the DBUTIL service.

Command invocation format



Call invocation format

You cannot use call procedures to start this service.

ISPF interface panel

```
  Menu  SCLM  Utilities  Help
_______________________________________________________________________________
                  SCLM FLMCMD DBUTIL Service - Entry Panel
                                                                        More:     +
SCLM Library:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1____ . . . TEST____ . . . RELEASE_
         . . . . ________ . . . ________ . . . ________
   Type  . . . . SOURCE__
   Member  . . . ________

Selection criteria: (Patterns may be used)
  Authorization code  . . . *_______     Data type . . _  1. Account
  Change code . . . . . . . *_______                      2. Build map
  Change group  . . . . . . *_______                      3. Both
  Change user id  . . . . . *_______
  Language  . . . . . . . . *_______     Enter "/" to select option
                                         /  First occurrence only
Hierarchy search information:
  Architecture Control   3  1. In          Scope    1  1. Normal
Command ===> ____________________________________________________________________
 F1=HELP      F2=          F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 190. DBUTIL Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters.

prj_def
The project definition name to be used for the data extraction. It defaults to the project. The maximum parameter length is 8 characters.

acct_group1 - acct_group6|*
The group containing the members, accounting records, and build maps to be reported on. The maximum parameter length is 18 characters. You can specify up to six individual acct_groups, an asterisk for all, or up to six valid patterns. Only groups from the project definition are reported. The default is all account groups (*).

acct_type|*
The type containing the members, accounting records, and build maps to be reported on. Only types from the project definition are reported. The maximum parameter length is 18 characters. You can specify an individual acct_type, an asterisk for all of them, or a valid pattern. The default is all account types.

acct_member|*
The name of the members' accounting records and build maps on which the report will occur. The maximum parameter length is 18 characters. You can specify an individual acct_member, an asterisk for all of them, or a valid pattern. The default is all account members.

authcode|*
The current authorization code for the member. The maximum parameter length is 18 characters. You can specify an individual authcode, an asterisk for all of them, or a valid pattern. The default is all authorization codes.

change_code|*
A value previously assigned by a user for reference purposes. The maximum parameter length is 18 characters. You can specify an individual change_code, an asterisk for all of them, or a valid pattern. The default is all change codes.

change_group|*
The name of the group in which the member was last updated. The maximum parameter length is 18 characters. You can specify an individual change_group, an asterisk for all of them, or a valid pattern. The default is all change groups.

change_userid|*
The user ID of the person who made the last update to the member. The maximum parameter length is 18 characters. You can specify an individual change_userid, an asterisk for all of them, or a valid pattern. The default is all change_user IDs.

language|*
The language of the member. The maximum parameter length is 18 characters. You can specify an individual language, an asterisk for all of them, or a valid pattern. The default is all languages.

YES|NO
If you specify YES and use more than one group pattern, a precedence system determines which members are selected. If you specify NO, SCLM selects all versions of all members. The maximum parameter length is 24 characters. The default value is YES.

ACCT|BMAP|*
Specify the following type of data to report on:

ACCT
Accounting information

BMAP
Build map information

***
Build map and accounting information.

The maximum parameter length is 24 characters. The default value is ACCT.

IN|OUT|*
Specify the following scopes to select members:

IN
Controlled by the architecture definition

OUT
Not controlled by the architecture definition

***
Without using an architecture definition to identify them.

The maximum parameter length is 24 characters. The default is an asterisk, which indicates that members will be selected without an architecture definition. If you specify either IN or OUT, you must specify arch_group, arch_type, and arch_member.

arch_group
The group used to identify the lowest group in the hierarchy where the architecture begins. The maximum parameter length is 8 characters.

arch_type
The type containing the architecture definition that controls the selected members. The maximum parameter length is 8 characters.

arch_member
The member containing the architecture definition that controls the selected members. The maximum parameter length is 8 characters.

EXTENDED|NORMAL|SUBUNIT
Specify the following architecture scope to select:

NORMAL
Members that do or do not have compilation unit dependencies.

EXTENDED|SUBUNIT
Members that do have compilation unit dependencies.

The maximum parameter length is 24 characters. The default value is NORMAL.

YES|NO
Specify YES to include page header information in the tailored output. In addition to suppressing the page header information, NO positions the data in column 1 of the tailored output. No carriage returns appear in the output. The maximum parameter length is 24 characters. The default value is YES.

YES|NO
Specify YES to sum numeric data fields and to show the sum totals in the tailored output. The maximum parameter length is 24 characters. The default value is YES.

report_name
The title of the report to be written in the tailored output. The maximum parameter length is 35 characters. Commas are not allowed in the report name.

dd_msgs
The ddname indicating the destination of the DBUTIL service messages. If you specify a blank ddname, SCLM routes the DBUTIL service messages to the default output device, such as your terminal. Otherwise, before you call the DBUTIL service, you must allocate the ddname. The following attributes should be used: `RECFM=F, LRECL=80, BLKSIZE=80`. The maximum parameter length is 8 characters.

dd_rept
The ddname indicating the destination of the report. If you specify a blank ddname, SCLM routes the report to the default output device, such as your terminal. Otherwise, before you call the DBUTIL service, you must allocate the ddname. The following attributes should be used: `RECFM=FBA, LRECL=80, BLKSIZE=3120`. The maximum parameter length is 8 characters.

dd_tailor
The ddname indicating the destination of the tailored data set. If you specify a blank ddname, SCLM does not generate the tailored output. Otherwise, before you call the DBUTIL service, you must allocate the ddname. The following attributes should be used: `RECFM=F, V, FB,` or `VB; LRECL=80` (minimum); and `LRECL=2048` (maximum). If the LRECL value is less than 80, you receive an error message. The report continues to be generated, but it is wrapped using the LRECL value you specify. The maximum parameter length is 8 characters.

report_line
A line of data input that determines the content of the tailored output. Note that you can include commas in the report_line. If you specify all other parameters or if they default correctly, SCLM does not parse the report_line for commas. The maximum parameter length is 160 characters, but the report line will be wrapped if it is more than 80 characters long.

If you use SCLM variables with data lengths greater than 8, keep in mind that their values can exceed 8 characters. Place these variables at the end of the report line to ensure that the columns in the report line up evenly. See Chapter 23, “SCLM Variables and Metavariables,” on page 631 for more information.

Here is the default value for the report_line:

```
@@FLMMBR  @@FLMLAN @@FLMCML @@FLMNCL @@FLMBLL @@FLMTLS @@FLMCMS @@FLMNCS
```

Return codes

Additional special services messages are written to the FLMMSGS ddname. See “SCLM service messages” on page 378 for more information.

Other return codes might be produced by the FLMCMD processor. See “SCLM service return codes” on page 377 for more information.

Possible return codes are:

0
Normal completion. See the SCLM messages for more information.

4
Warning condition. See the SCLM messages for more information.

8
Error condition. See the SCLM messages for more information.

> 8
Severe error condition and SCLM does not produce messages. See “Return codes” on page 458 for a description of the return code.

Example of command invocation

```
FLMCMD DBUTIL,PROJ1,,USER1,,,,,,
*,*,,,,,,N,ACCT,*,,,,,N,N,NAME,,,
UTILTAIL,DELETE,@@FLMPRJ,PROJ1,@@FLMGRP,@@FLMTYP,@@FLMMBR
```

This service command retrieves accounting information in the USER1 architecture group. SCLM selects all versions of the member without using an architecture definition to identify them. SCLM also selects all accounting types and accounting members that match the pattern.

The dd_tailor parameter, UTILTAIL, indicates the destination of the tailored output called NAME. The report_line parameter passes SCLM variables to produce a cleanup report, which you can use to delete all of the members in a group. The cleanup report does not have header information and does not total numeric data fields.

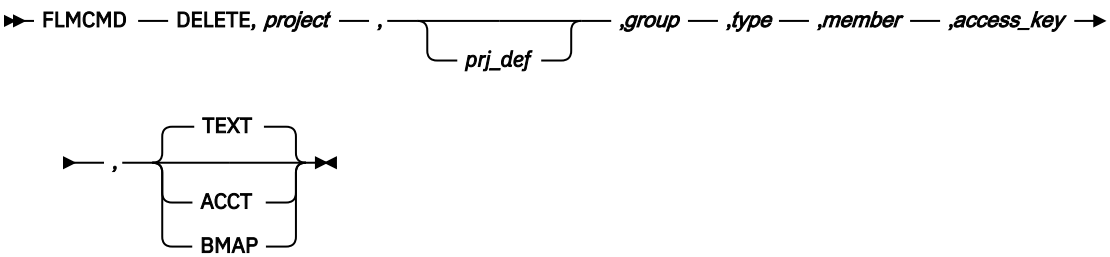
DELETE—Delete Database Components

The DELETE service deletes database components. You can delete an entire member plus its associated accounting record and build map, a member's accounting record and build map, or a member's build map.

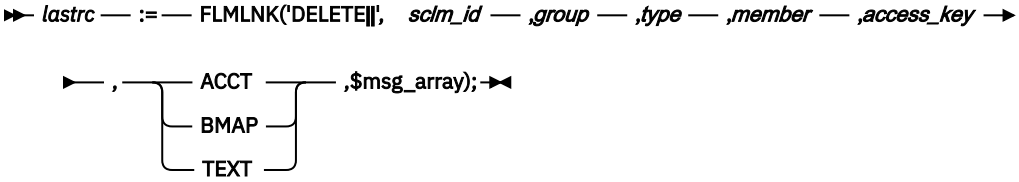
If you delete a member from a development group, and the next higher group is non-key, you should also delete the same member from the non-key group if it exists there.

Note: The DELETE function requires update authority for the member in order to delete the build map and accounting information.

Command invocation format



Call invocation format



ISPF interface panel

```
  Menu  SCLM  Utilities  Help
──────────────────────────────────────────────────────────────────────────────
                   SCLM FLMCMD DELETE Service - Entry Panel

SCLM Library:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1____
   Type  . . . . SOURCE__
   Member  . . . ________

Access key  . . . . . . . _________________
Data type . . _  1. Account
                2. Build map
                3. Text




Command ===> ____________________________________________________________________
 F1=HELP      F2=           F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 191. DELETE Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name to be used for the delete. It defaults to the project. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

group
The group in which the delete is to occur. The maximum parameter length is 8 characters.

type
The type containing the member, accounting record, and build map to be deleted. The maximum parameter length is 8 characters.

member
The name of the member, accounting record, and build map to be deleted. The maximum parameter length is 8 characters.

access_key
The access key assigned to the member with the LOCK service. If you supply the incorrect access key, the delete fails. The maximum parameter length is 16 characters.

ACCT|BMAP|TEXT
Indicates which types of data SCLM is to delete for the member. If you specify BMAP, SCLM deletes only the member's build map. If you specify ACCT, SCLM deletes the member's build map and accounting record. If you specify TEXT, SCLM deletes the member's build map, the member's

accounting record, and the member. The maximum parameter length is 24 characters. For FLMCMD, the default value is TEXT. There is no default value for this parameter for FLMLNK.

$msg_array
An output parameter pointing to the message array. See "Pointer parameter descriptions" on page 368 for more information about $msg_array. This parameter is used for FLMLNK only.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See "SCLM service messages" on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM service return codes" on page 377 for more information.

Possible return codes are:

0
Normal completion.

4
Warning condition. The member, accounting record, or build map was not found. This return code is set whenever any of the data is missing, regardless of whether the request was for ACCT, BMAP, or TEXT.

8
Error condition. The $msg_array parameter contains the error message associated with this condition.

Example of command invocation

```
FLMCMD DELETE,PROJ1,,USER1,SOURCE,FLM01MD2,XXX#04,ACCT
```

This service command deletes the build map and accounting record for the FLM01MD2 member of the SOURCE type in the USER1 group. The project name is PROJ1. The access key for the member is XXX#04.

If the text for the member FLM01MD2 is missing, then the service returns a return code of 4, even though deletion of the text member was requested.

Example of call invocation

This example shows general syntax. Call invocations are language-specific. See Chapter 20, "Sample programs using SCLM services," on page 477 for specific examples.

```
lastrc := FLMLNK('DELETE  ',                  (* service              *)
                  sclm_id,                    (* SCLM ID              *)
                 'USER1   ',                  (* group                *)
                 'SOURCE  ',                  (* type                 *)
                 'FLM01MD2',                  (* member               *)
                 'XXX#04          ',          (* access key           *)
                 'ACCT                  ',    (* type of data to delete *)
                 $msg_array);                 (* message array pointer  *)
```

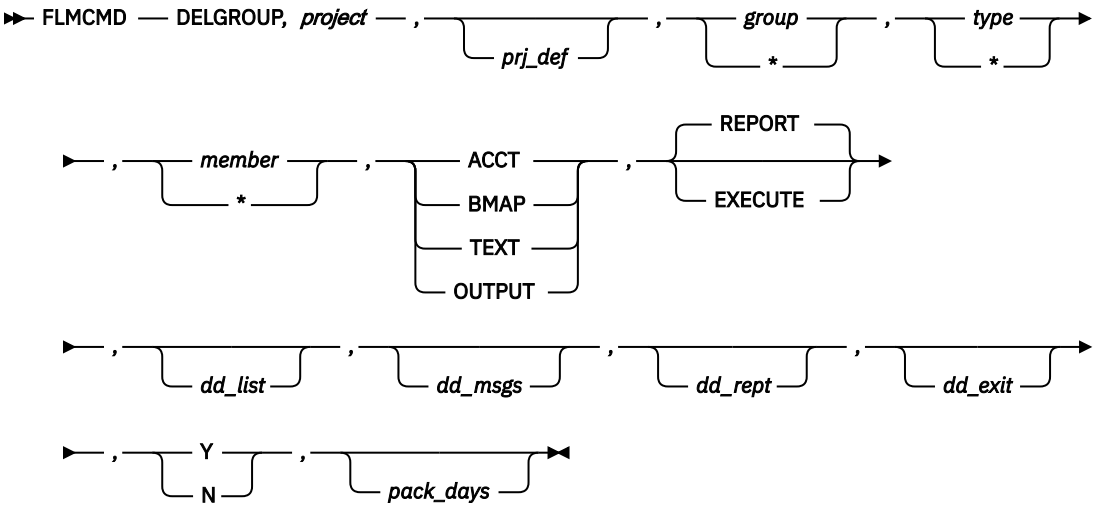
This service call deletes the accounting record and the build map for the FLM01MD2 member of the SOURCE type in the USER1 group. The sclm_id parameter contains a valid SCLM ID returned from the INIT service and the access key is XXX#04. SCLM returns all messages in the $msg_array.

DELGROUP—Delete from Group Database Components

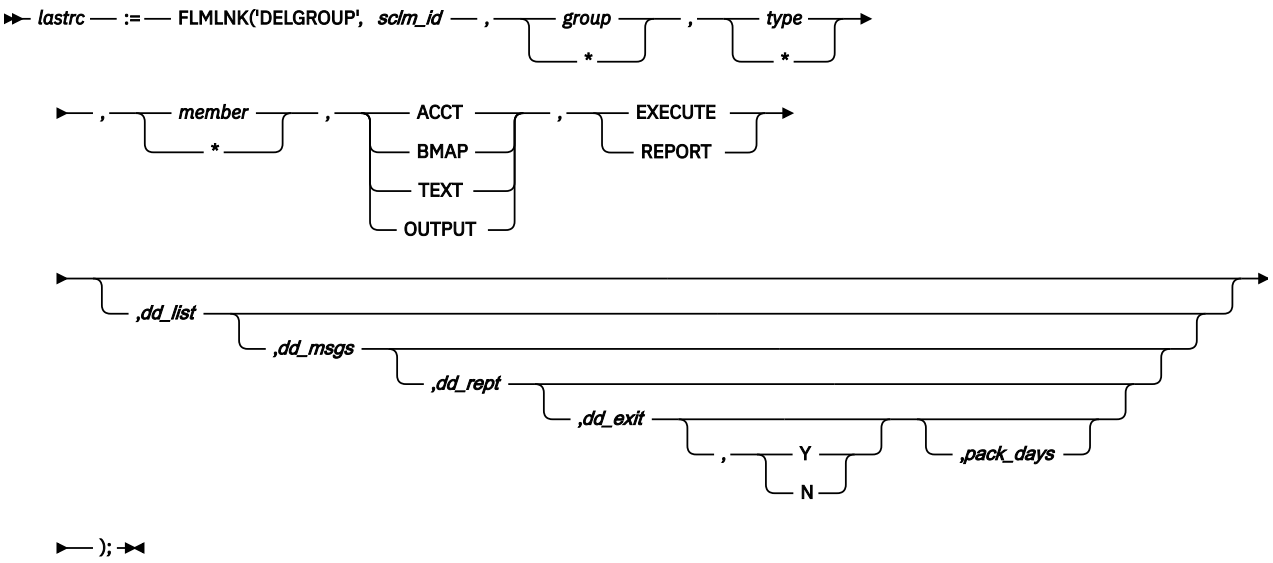
The DELGROUP service deletes SCLM-controlled database components associated with a specified group or groups matching a specified pattern. You can delete a member or members and all associated SCLM accounting information and build map records whose names match the selection criteria. You can further

specify whether you want everything deleted, only build outputs, only accounting information and build map records, or only build map records. You can delete backed-up packages older than a specified period. You can also specify that nothing actually be deleted, but that a deletion report be generated.

Command invocation format



Call invocation format



ISPF interface panel

```
  Menu  SCLM  Utilities  Help
──────────────────────────────────────────────────────────────────────────────
                  SCLM FLMCMD DELGROUP Service - Entry Panel
                                                                        More:     +
Delete from Group Input:
   Project . . . SCLMTEST
   Alternate . . _________
   Group . . . . _________           (Group or pattern to delete)
   Type  . . . . _________           (Type or pattern to delete)
   Member  . . . _________           (Member or pattern to delete)

Access key  . . . . . .  ________________

Data type . . _  1. Account           Delete Mode . . _  1. Execute
                 2. Build map                            2. Report
                 3. Text
                 4. Output

DD Names for output data sets:
   Error message data set  _________ (Blank to write messages to the terminal)
   Report data set  . . .  _________ (Blank to write report to the terminal)
Command ===> ____________________________________________________________________
 F1=HELP      F2=           F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT     F12=SUBMIT
```

Figure 192. DELGROUP Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name used for the delete. It defaults to the project parameter. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

sclm_id
An SCLM ID associated with a given project and project definition. The SCLM ID is generated by the INIT service. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

group|*
The group to be deleted. Only groups that are defined in the project definition will have members deleted. Records in the VSAM data sets for groups that match the pattern but are not in the project definition are not deleted. The maximum parameter length is 17 characters. You can specify an individual group, an asterisk (*) for all groups, or a valid pattern.

Attention: If you specify an asterisk, all groups are deleted, so use extreme caution when using the asterisk.

If you use the Delete from Group Utility panel to invoke Delete from Group, you cannot specify a pattern for the group field. Pattern matches in this field are restricted because of the possible hazards of using a pattern in this field.

type|*
The type containing the members, accounting records, and build maps to be deleted. The maximum parameter length is 17 characters. You can specify an individual type, an asterisk (*) for all types, or a valid pattern. You must specify a type. Only members with types defined in the project definition will be deleted.

The member pattern must also match.

member|*
The name of the members, accounting records, and build maps to be deleted. The maximum parameter length is 17 characters. You can specify an individual member, an asterisk (*) for all members, or a valid pattern. See “Selection parameters” on page 367 for more information about specifying wildcard characters.

The type pattern must also match.

ACCT|BMAP|TEXT|OUTPUT
Indicates which types of data SCLM is to delete.

If you specify BMAP, SCLM deletes only the group's build maps.

If you specify ACCT, SCLM deletes the group's build maps and accounting records.

If you specify TEXT, SCLM deletes the group's build maps, accounting records, and the PDS members associated with those records. If there is no build map or accounting information for a PDS member, the member is not deleted even if you specify the TEXT option.

If you specify OUTPUT, SCLM deletes the group's build outputs that match the selection criteria.

The maximum parameter length is 24 characters.

Because this service deletes information and there is no "Undelete" service, there is no default for this parameter.

Note: SCLM can continue to search for deleted data sets that were once active in the project. SCLM issues warning messages if references to deleted data sets are found.

EXECUTE|REPORT
If you specify EXECUTE, any members that match the selection criteria for the specified delete flag are deleted. A report indicating which members were deleted is produced.

If you specify REPORT, no members are deleted. Instead, SCLM produces a report indicating which members are eligible for deletion. SCLM sends this report to the default output device. Specifying REPORT is a good way to identify the outcome of the delete process before deleting any members. The maximum parameter length is 24 characters. For FLMCMD, the default value is REPORT. There is no default value for FLMLNK. You are required to have update authority to the hierarchy data sets to use the DELGROUP service in either REPORT or EXECUTE mode.

dd_list
The ddname indicating the destination of the purge listing for deletion of intermediate code. You must also specify TEXT or OUTPUT and intermediate code must be deleted to produce this report. If you specify a blank ddname, no listing is produced. Otherwise, before you call the DELGROUP service, you must allocate the ddname. The following attributes should be used: `RECFM=VBA, LRECL=137, BLKSIZE=3120`. The maximum parameter length is 8 characters.

dd_msgs
The ddname indicating the destination of the DELGROUP messages. If you specify a blank ddname, SCLM routes the messages to the default output device. Otherwise, before you call the DELGROUP service, you must allocate the ddname. The following attributes should be used: `RECFM=F, LRECL=80, BLKSIZE=80`. The maximum parameter length is 8 characters.

dd_rept
The ddname indicating the destination of the DELGROUP report. If you specify a blank ddname, SCLM sends the DELGROUP report to the default output device, such as your terminal. Otherwise, before you call the DELGROUP service, you must allocate the ddname; the following attributes should be used: `RECFM=F, LRECL=80, BLKSIZE=80`. The maximum parameter length is 8 characters.

dd_exit
The ddname indicating the destination of the delete user exit data. Specify this parameter only if your project definition defines a notify delete user exit routine. Ask your project manager if your project is using a notify delete user exit routine. If you specify a blank ddname, SCLM routes the delete user exit data to NULLFILE. Otherwise, before you call the DELGROUP service, you must allocate the ddname. The following attributes should be used: `RECFM=FB, LRECL=160, BLKSIZE=3200`. The maximum parameter length is 8 characters.

Y|N
Set to Y to delete a package and any backed up modules within that package.

pack_days
Indicates an age in days up to which packages will not be deleted. If the pack_days parameter is set, only backed up packages that are older than this value will be deleted.

Note:

1. The pack_del and pack_days parameters can only be used for deleting packages.
2. To delete packages you must specify values for group and type; "*" is not allowed. However you can specify "*" in the member name to delete all the members.
3. If the pack_del parameter is set, a DELGROUP report is not produced. If REPORT is specified, entries are added to the message file detailing the packages that will be deleted.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See "SCLM service messages" on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM service return codes" on page 377 for more information.

Possible return codes are:

0
Normal completion. See the SCLM messages for more information.

4
Warning condition. See the SCLM messages for more information.

8
Error condition. See the SCLM messages for more information.

12
Severe error condition. SCLM does not produce messages because there was an error invoking the DELGROUP module.

16
Severe error condition. SCLM does not produce messages because it was unable to retrieve SCLM ID information.

Example of command invocation

```
FLMCMD DELGROUP,PROJ1,,USER1,*,*,ACCT,EXECUTE
```

This service command deletes the build map and accounting records for all types and members that are associated with the USER1 group in the PROJ1 project. SCLM sends messages to the terminal.

Example of call invocation

This example shows general syntax. Call invocations are language-specific. See Chapter 20, "Sample programs using SCLM services," on page 477 for specific examples.

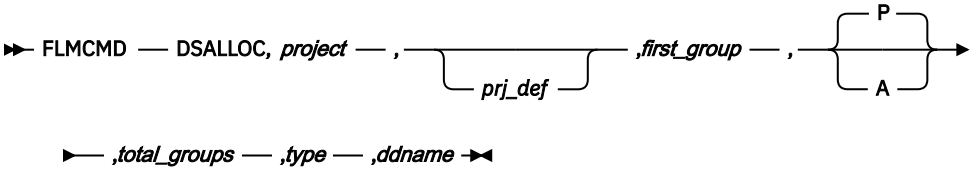
```
lastrc := FLMLNK('DELGROUP',                  (* service        *)
                 sclm_id,                     (* SCLM ID        *)
                 'USER1   ',                  (* group          *)
                 '*                ',         (* type           *)
                 '*                ',         (* member         *)
                 'ACCT                   ',   (* types of data  *)
                 'EXECUTE '                   (* delete members *)
                 ,dd_list                     (* listing        *)
                 ,dd_msgs                     (* messages       *)
                 ,dd_rept);                   (* report         *)
```

This service call deletes the build maps and accounting records for all types and members associated with the USER1 group in the PROJ1 project. The sclm_id parameter contains a valid SCLM ID returned from the INIT service. SCLM sends messages to the terminal.

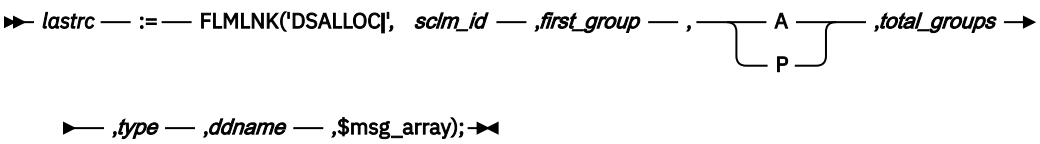
DSALLOC—Allocate Data Sets for Group/Type

The DSALLOC service allocates a ddname that corresponds to a hierarchy view specified by the user. The hierarchy view is a concatenation of the PDS data sets, beginning with the PDS data set for the first_group and adding the PDS for each group above it in the hierarchy. If the ddname already exists, the old ddname is replaced with the new ddname. If unallocated data sets are contained in the hierarchy view, then only the allocated data sets are associated with the ddname. The list of data sets allocated to the ddname does not include extended types.

Command invocation format



Call invocation format



ISPF interface panel

```
  Menu  SCLM  Utilities  Help
_______________________________________________________________________________
                  SCLM FLMCMD DSALLOC Service - Entry Panel

SCLM Library To Allocate:
   Project . . . SCLMTEST
   Alternate . . ________
   First Group   DEV1
   Type  . . . . SOURCE

Total number of groups  . . . . . . ___  (Zero for entire hierarchy)
Type of hierarchy to be allocated   _  1. All groups
                                       2. Primary groups only

DD Name for output data set:
   DD Name to allocate . .  ________




Command ===> ____________________________________________________________________
 F1=HELP      F2=          F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 193. DSALLOC Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name used for the allocate. It defaults to the project parameter. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

sclm_id
The SCLM ID associated with a given project. The INIT service generates the SCLM ID. Maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

first_group
The first group in the hierarchy to be allocated to the ddname. Maximum parameter length is 8 characters. This group defines the desired view of the hierarchy. DSALLOC allocates the data sets SCLM uses to search the hierarchy from the group specified.

A|P
A 1-character value indicating the type of hierarchy to be allocated to the ddname. Acceptable values are:

A
All groups

P
Primary groups only.

For FLMCMD, the default value is P. There is no default value for FLMLNK.

total_groups
The numeric value corresponding to the number of groups for which the allocation is performed. This number includes the first_group. Specify a zero (0); if the entire hierarchy view is wanted. The default value is zero. If this value is greater than the number of groups in the view, all groups in the view are allocated and a warning occurs. The maximum parameter length is 3 characters.

type
The name of the type for which the allocation is performed. Maximum parameter length is 8 characters.

ddname
The ddname for the allocated physical data sets corresponding to the desired hierarchy view. The physical data set names are dynamically allocated to the ddname. You can specify the ddname to be used or leave it blank for the FLMLNK interface. If the ddname already exists, the old ddname is replaced with the new ddname. A blank value is not allowed for FLMCMD. If the ddname is blank, SCLM creates a ddname and uses it to allocate data sets; this name is returned to the user. Maximum length of this field is 8 characters.

$msg_array
An output parameter pointing to the message array. See "Pointer parameter descriptions" on page 368 for more information about $msg_array. This parameter is used for FLMLNK only.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See "SCLM service messages" on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM service return codes" on page 377 for more information.

Possible return codes are:

0
Normal completion.

4
Warning condition. The $msg_array parameter contains the warning message associated with this condition. A warning occurs if the number of data sets allocated to the ddname is less than the number requested in the total_groups parameter.

8
Error condition. The $msg_array parameter contains the error message associated with this condition.

Example of command invocation

```
FLMCMD DSALLOC,PROJ1,,USER1,P,4,SOURCE,APPL
```

This service invocation returns the ddname APPL with the physical data set names corresponding to the hierarchy view specified by the first_group and the total number of groups. If the hierarchy consisted of 4 groups (USER1, INT, TEST, and RELEASE), these 4 physical data set names would be allocated to ddname APPL. A user wanting a ddname corresponding to a single group would specify the same group for the first_group and 1 for the total number of groups.

Example of call invocation

Note: This example shows general syntax. Call invocations are language-specific. See Chapter 20, "Sample programs using SCLM services," on page 477, for specific examples.

```
lastrc := FLMLNK ('DSALLOC ',       (* service              *)
                  sclm_id,          (* SCLM ID              *)
                  'USER1   ',       (* first group          *)
                  'P',              (* hierarchy            *)
                  4,                (* total groups         *)
                  'SOURCE  ',       (* type                 *)
                  ddname,           (* ddname to allocate   *)
                  $msg_array);      (* message array pointer *)
```

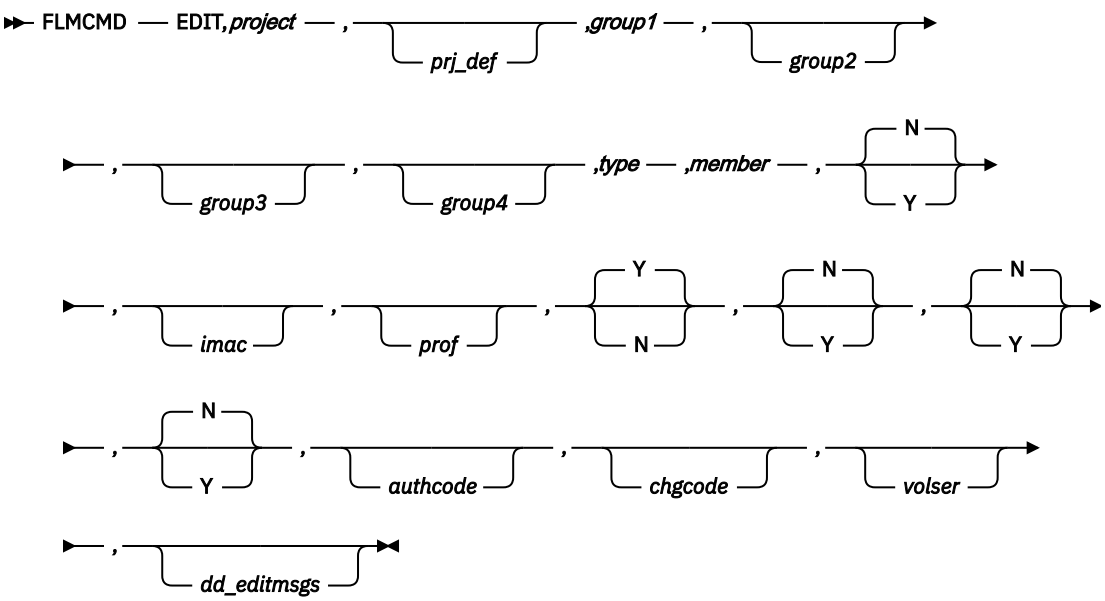
Assume that the ddname for the preceding example is APPL. This service invocation returns the ddname APPL with the physical data set names corresponding to the hierarchy view specified by the first_group and the total number of groups. If the hierarchy consisted of 4 groups (USER1, INT, TEST, and RELEASE), these 4 physical data set names would be allocated to ddname APPL. Note the project is determined by

the sclm_id that is obtained by the INIT service call. A user wanting a ddname corresponding to a single group would specify the same group for the first_group and 1 for the total number of groups.

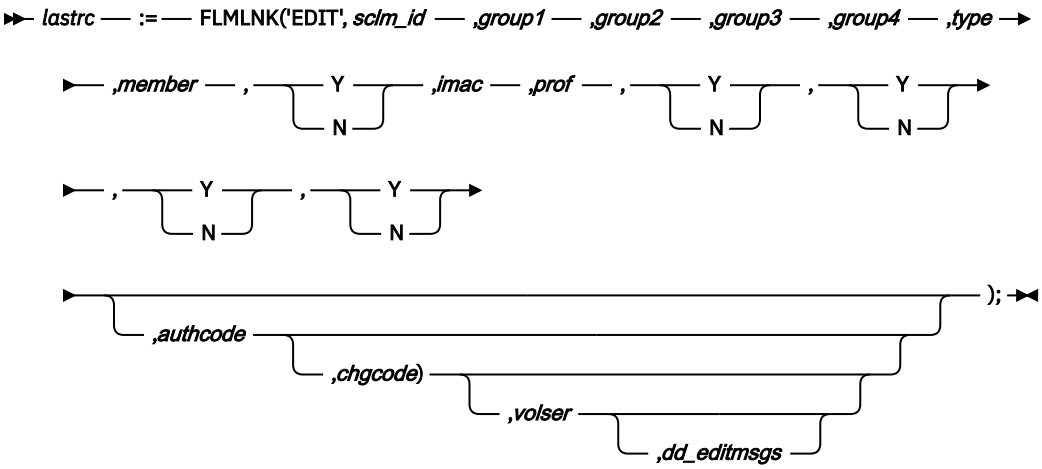
EDIT— Edit a Member of a Controlled Library

The EDIT service brings up an SCLM edit session for the requested member. All of the functions of the SCLM edit panel are available from the edit service, including locking, parsing, and storing SCLM accounting data. The SCLM edit commands, such as SPROF, SCREATE, SREPLACE, and SMOVE, are the same as from the SCLM edit dialog.

Command invocation format



Call invocation format



ISPF interface panel

```
  Menu  SCLM  Utilities  Help
_______________________________________________________________________
                    SCLM FLMCMD EDIT Service - Entry Panel
                                                                More:     +
SCLM Library:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1____  . . . TEST____ . . . RELEASE_ . . . ________
   Type  . . . . SOURCE__       ________       ________
   Member  . . . _________

Initial Macro . . . . . ________
Profile Name  . . . . . ________  (If blank, defaults to data set type)

Options
_  Allocate Hierarchy
/  Confirm Cancel/Move/Replace
_  Mixed Mode
_  Edit on Workstation
_  Preserve VB record length

Command ===> ___________________________________________________________________
 F1=HELP      F2=          F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 194. EDIT Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters.

prj_def
The project definition name to be used for edit. It defaults to project. The maximum parameter length is 8 characters.

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters.

group1
The development group at which the member is to be edited. The maximum parameter length is 8 characters.

group2
Name of the second group in the concatenation. The maximum parameter length is 8 characters.

group3
Name of the third group in the concatenation. The maximum parameter length is 8 characters.

group4
Name of the fourth group in the concatenation. The maximum parameter length is 8 characters.

type
The type containing the member to be edited. The maximum parameter length is 8 characters.

member
The member to be edited. The maximum length for this parameter is 8 characters.

Y|N
Y indicates that SCLM will allocate the entire hierarchy, beginning with group1. If Y is selected, group2, group3 and group4 must be blank. N indicates that SCLM will allocate only group1, group2, group3, and group4 (default).

imac
The name of an initial macro to be run. The maximum parameter length is 8 characters.

prof
The edit profile name to use for the edit session. The maximum parameter length is 8 characters.

Y|N
Y indicates that you will have an opportunity to confirm cancel, move, and replace (default). N indicates that cancel, move, and replace commands will execute without confirmation. The maximum parameter length is 24 characters.

Y|N
Y indicates that mixed edit mode is to be used. N indicates that mixed edit mode is not to be used (default). The maximum parameter length is 24 characters.

Y|N
Y indicates that the data will be edited on the workstation. N indicates that the data will be edited on the host (default). The maximum parameter length is 24 characters.

Y|N
Y indicates that the length of variable blocked data will be preserved. N indicates that blanks at the end of the data are retained (default). The maximum parameter length is 24 characters.

authcode
The authorization code to be used for the edit session. SCLM uses the authorization code for the verification just like the SCLM edit dialog. If you do not enter a blank or do not supply an authcode SCLM uses one of the following default values:

- The authorization code from the existing member, if the member being edited exists in the hierarchy.
- The default authorization code for the group if the member does not exist in the hierarchy.

The maximum parameter length is 8 characters.

chgcode
The default change code for the edit session. If the member's accounting record lists the change code, SCLM updates the date and time stamps for the existing change code entry. The maximum parameter length is 8 characters.

volser
Volume serial for parser listings data set. The maximum parameter length is 8 characters.

dd_editmsgs
DDNAME of the destination of the edit messages. Allocate this ddname with DISP=MOD. The maximum parameter length is 8 characters.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See “SCLM service messages” on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See “SCLM service return codes” on page 377 for more information.

Possible return codes are:

0
Normal completion; data was saved.

4
Normal completion; data was **not** saved.

8
Error condition. See the dd_editmsgs for details.

12
Severe error condition. SCLM does not produce messages because there was an error invoking the edit module.

14
Member in use.

16
Verification error from a user exit routine.

Example of command invocation format

```
FLMCMD EDIT,sclm70,sclm7044,user,,,,SOURCE,A,Y
```

This service command edits SCLM70.USER.SOURCE(A), drawing down member a from the hierarchy, if it is not in group USER. Alternate SCLM7044 is used to determine the hierarchy.

Call invocation format

```
CALL FLMLNK('EDIT ',SLMID,'USER ',BLNK8,BLNK8,BLNK8,
   'SOURCE ','A ',Y,BLNK8,BLNK8,
   Y,N,N,N,'PRIVATE ','R8EDS ',BLNK8,
   DDEDIT)
   RETCODE(R15);
```

This service call edits member A in group USER with type SOURCE, drawing down member EDIT service. The example assumes that the START and INIT services have already completed successfully, so that the SLMID value is valid.

The ddname DDEDIT has been allocated to a data set with valid characteristics.

The authcode is set to 'PRIVATE' and the change code is set to 'R8EDS'.

END— End an SCLM Services Session

The END service stops an SCLM services session. It frees an application ID generated by the START service. Each START service invocation needs a matching END service invocation. This service also calls the FREE service to free any SCLM IDs associated with the given application ID that have not been explicitly freed.

Command invocation format

You cannot use command procedures to call this service.

Call invocation format

Parameters

appl_id
The application ID associated with the SCLM services session you want to stop. You must generate the application ID using the START service. The maximum parameter length is 8 characters.

msg_line
An output parameter that has a buffer containing any END service error message. The maximum parameter length is 80 characters.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See “SCLM service messages” on page 378 for more information.

Other return codes might be produced by the FLMLNK processor.

Possible return codes are:

0
Normal completion.

4
Warning condition. SCLM cannot free an SCLM ID associated with the application ID.

8
Error condition. See the msg_line parameter description for more details.

Example of call invocation

This example shows general syntax. Call invocations are language-specific. See Chapter 20, “Sample programs using SCLM services,” on page 477 for specific examples.

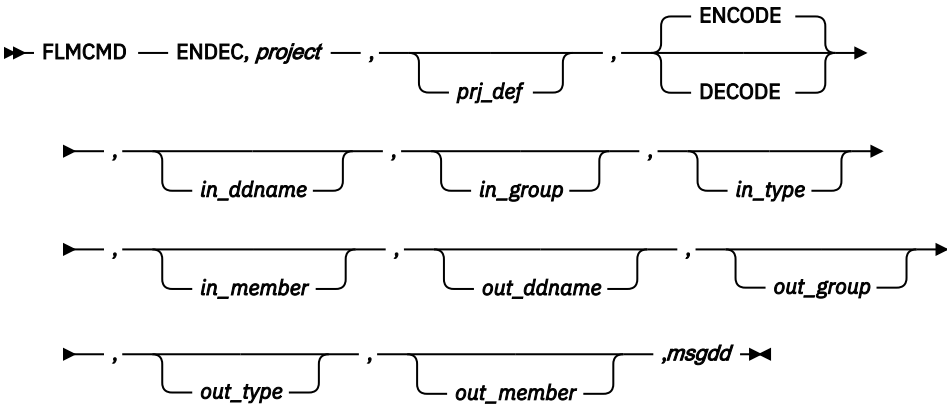
```
lastrc := FLMLNK('END     ',          (* service        *)
                  appl_id,            (* application ID *)
                  msg_line);          (* error messages *)
```

This service call ends the SCLM services session identified by the appl_id parameter. The appl_id parameter contains a valid application ID returned from the START service. SCLM returns messages in the msg_line parameter.

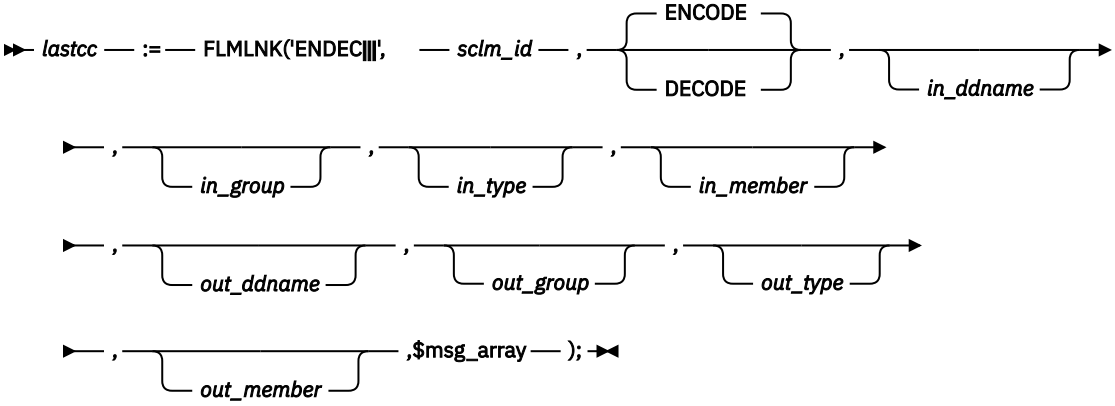
ENDEC— Encode and Decode members

With the ENDEC service, you can encode or decode SCLM members using the terse algorithm. This service can be used initially to encode members or decode SCLM members so they can be used in SCLM language translators and external SCLM processes. You can use the ENDEC service with the ENCODE parameter to reduce the size of members or data sets.

Command invocation format



Call invocation format



Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name. It defaults to the project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

ENCODE|DECODE
ENCODE specifies that the input data set and member are encoded into the output data set and member. If the input is already encoded, the service will receive a return code of 4. DECODE specifies that the input data set and member will be decoded into the output data set and member. If the input is already decoded, the service will receive a return code of 4.

in_ddname
The ddname containing the data set and member to be used as input to the ENDEC service. If specified, the in_group and in_type parameters should not be specified. The maximum parameter length is 8 characters.

in_group
The group to be used to determine the data set to be used as input to the ENDEC service. If specified, the in_type and in_member parameter should also be specified. The maximum parameter length is 8 characters.

in_type
The type to be used to determine the data set to be used as input to the ENDEC service. If specified, the in_group and in_member parameter should also be specified. The maximum parameter length is 8 characters.

in_member
The member under SCLM control to be used as input to the ENDEC service.The maximum parameter length is 8 characters.

out_ddname
The ddname containing the data set and member to be used as output to the ENDEC service. If specified, the out_group and out_type parameters are not specified. The maximum parameter length is 8 characters.

out_group
The group to be used to determine the data set to be used as output to the ENDEC service. If specified, the out_type and out_member parameter are also specified. The maximum parameter length is 8 characters.

out_type
The type to be used to determine the data set to be used as output to the ENDEC service. If specified, the out_group and out_member parameter is also specified. The maximum parameter length is 8 characters.

out_member
The member under SCLM control that is used as output to the ENDEC service. The maximum parameter length is 8 characters.

dd_msgs
The ddname indicating the destination of the messages generated by the ENDEC service. If you specify a blank ddname, SCLM routes the ENDEC messages to the default output device, such as your terminal. Otherwise, before you call the ENDEC service, you must allocate the ddname. The following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

$msg_array
An output parameter pointing to the message array. See "Pointer parameter descriptions" on page 333 for more information about $msg_array. This parameter is used for FLMLNK only.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See “SCLM service messages” on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See “SCLM service return codes” on page 377 for more information.

Possible return codes are:

0
Normal completion. The encoding and decoding was performed.

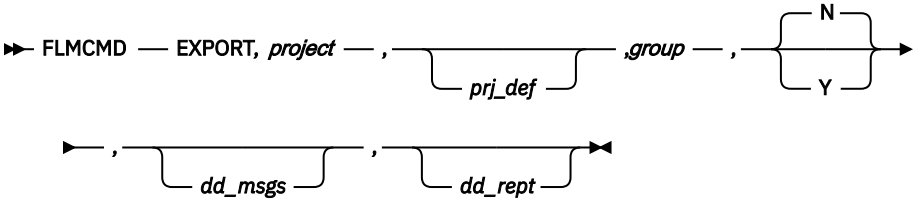
4
Warning if ENCODE was specified, the input data set and member is already encoded. If DECODE was specified, the input data set and member is already decoded.

12
Error completion. Refer to the messages for more information.

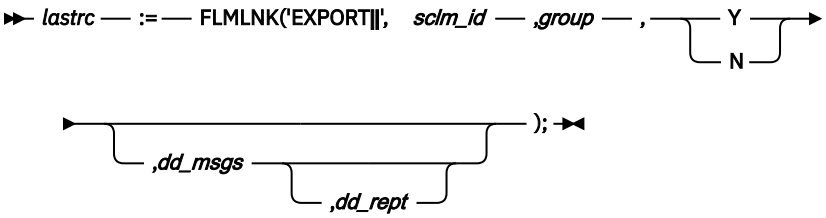
EXPORT—Extract SCLM Accounting Information for a Group

The EXPORT service captures all SCLM accounting and build map information associated with a specified group. You can use this service with the IMPORT service to create a consistent set of data that can be archived or used to create a new release, rename a group, or transport software from one hierarchy to another. Although the SCLM Migration Utility provides a similar function, using the EXPORT and IMPORT services allows you to save build maps. Data presently residing in the group specified is not changed by this service.

Command invocation format



Call invocation format



ISPF interface panel

```
  Menu  SCLM  Utilities  Help
──────────────────────────────────────────────────────────────────────────────
                   SCLM FLMCMD EXPORT Service - Entry Panel

SCLM Export Criteria:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1____

Enter "/" to select option
_  Replace export data

DD Names for output data sets:
   Error message data set  ________ (Blank to write messages to the terminal)
   Report data set  . . .  ________ (Blank to write report to the terminal)




Command ===> ____________________________________________________________________
 F1=HELP      F2=           F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 195. EXPORT Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name used for the export. It defaults to the project definition. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

sclm_id
An SCLM ID associated with a given project and project definition. The SCLM ID is generated by the INIT service. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

group
The group to be exported. The maximum parameter length is 8 characters. The group must be defined in the project definition. The group must have export VSAM data sets defined in the project definition.

Y|N
Indicates whether to purge previously exported data from the export data sets for the group. The export data sets must be empty before new export data can be stored in them. If you specify Y, SCLM attempts to purge the data in the data sets. If you specify Y and the purge fails, the export does not occur. If you specify N, SCLM assumes that the export data sets are empty and does not attempt to purge the data sets. If the export data sets are not empty, the export does not occur. The maximum parameter length is 24 characters. For FLMCMD, the default value is N. There is no default value for FLMLNK.

dd_msgs
The ddname indicating the destination of the export messages. If you specify a blank ddname, SCLM routes the export messages to the default output device, such as your terminal. Otherwise, before you call the EXPORT service, you must allocate the ddname; the following attributes should be used: `RECFM=F, LRECL=80, BLKSIZE=80`. The maximum parameter length is 8 characters.

dd_rept
The ddname indicating the destination of the export report. If you specify a blank ddname, SCLM routes the export report to the default output device, such as your terminal. Otherwise, before you call the EXPORT service, you must allocate the ddname; the following attributes should be used: `RECFM=F, LRECL=80, BLKSIZE=80`. The maximum parameter length is 8 characters.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See “SCLM service messages” on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See “SCLM service return codes” on page 377 for more information.

Possible return codes are:

0
Normal completion. See the SCLM messages for more information.

4
Warning condition. See the SCLM messages for more information.

8
Error condition. See the SCLM messages for more information.

12
Severe error condition. SCLM does not produce messages because there was an error invoking the EXPORT module.

16
Severe error condition. SCLM does not produce messages because it was unable to retrieve SCLM ID information.

Example of command invocation

```
FLMCMD EXPORT,PROJ1,,USER1,Y
```

This service command exports the USER1 group of the PROJ1 project. The export data sets are purged of any existing information before the SCLM accounting information is exported. SCLM sends messages and the report to the terminal.

Example of call invocation

This example shows general syntax. Call invocations are language-specific. See Chapter 20, “Sample programs using SCLM services,” on page 477 for specific examples.

```
lastrc := FLMLNK('EXPORT  ',                (* service             *)
                 sclm_id,                   (* SCLM ID             *)
                 'USER1   ',                (* group               *)
                 'Y                     ',  (* purge exported data *)
                 'EXPMSGS ',                (* messages            *)
                 'EXPREPT ');               (* report              *)
```

This service call exports the USER1 group. The sclm_id parameter contains a valid SCLM ID returned from the INIT service. The export data sets are purged of any existing information before the SCLM accounting information is exported. SCLM sends messages and the report to the terminal.

FREE—Free an SCLM ID

The FREE service frees an SCLM ID generated by the INIT service. Each INIT service invocation needs a matching FREE service invocation. After freeing the SCLM ID, SCLM closes all project data sets and frees the project definition specified on the INIT service.

Command invocation format

You cannot use command procedures to call this service.

Call invocation format

Parameters

sclm_id
The SCLM ID to be freed. The INIT service must generate the SCLM ID. The maximum parameter length is 8 characters.

msg_line
An output parameter that is a buffer containing any FREE service error message. The maximum parameter length is 80 characters.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See “SCLM service messages” on page 378 for more information.

Other return codes might be produced by the FLMLNK processor.

Possible return codes are:

0
Normal completion.

8
Error condition. See the msg_line parameter description for more details.

Example of call invocation

This example shows general syntax. Call invocations are language-specific. See Chapter 20, “Sample programs using SCLM services,” on page 477 for specific examples.

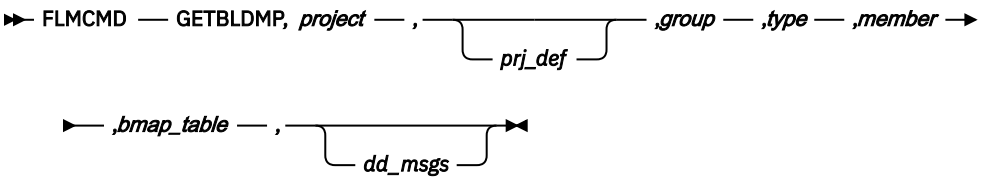
```
lastrc := FLMLNK('FREE    ',        (* service        *)
                 sclm_id,           (* SCLM ID        *)
                 msg_line);         (* error messages *)
```

This service call frees the SCLM ID identified by the sclm_id parameter. The sclm_id parameter contains a valid SCLM ID returned from the INIT service. SCLM returns messages in the msg_line parameter.

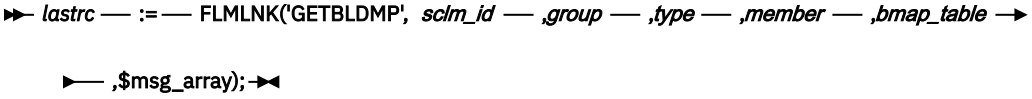
GETBLDMP—Retrieve Build Map Information

The GETBLDMP service retrieves the Build Map information associated with an SCLM-controlled member into an ISPF table. The information is retrieved from the accounting file defined in the project definition for the group specified to the service. The service can search up the hierarchy for the member, or retrieve the information for a specific member. See “ISPF variables” on page 372 for a list of the variables updated by this service.

Command invocation format



Call invocation format



ISPF interface panel

```
  Menu  SCLM  Utilities  Help
_______________________________________________________________________________
                  SCLM FLMCMD GETBLDMP Service - Entry Panel

SCLM Library:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1
   Type  . . . . SOURCE
   Member  . . . ________

DD Name for output data set:
   Error message data set  ________ (Blank to write messages to the terminal)









Command ===> ___________________________________________________________________
 F1=HELP      F2=          F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 196. GETBLDMP Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name. It defaults to the project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

group
The group associated with the accounting record. The maximum parameter length is 8 characters.

type
The type associated with the accounting record. The maximum parameter length is 8 characters.

member
The member under SCLM control. The maximum parameter length is 8 characters. You can specify an individual member, an asterisk for all of them, or a valid pattern.

bmap_table
The name of the ISPF table to contain the build map entries. The table must be open before calling the GETBLDMP service. A TBADD is performed for each build map record.

The following ISPF variables must be used in the table definition in order to have their value stored in the table. All of the variables are optional and only the variables required need to be specified:

ZSBMGRP
Group name where build map was located

ZSBMTYP
The type associated with the accounting record for which the build map was requested. This will be the same as was specified in the *type* parameter.

ZSBMMEM
Member name(s) of build map entries that match the pattern specified in the *member* parameter.

ZSBKWRD
entry type. For example the keyword used in the ARCHDEF or language definition, such as SINC, OBJ, and LOAD.

ZSBGRP
Build map entry group name. Group where the account record for the build map entry is stored in SCLM.

Note : If this parameter is specified in the table, then the account records for each build map entry will be searched to locate the group where the account record exists. If a wildcard is used for member, or the build map is large, this call may take some time.

ZSBMEM
member name

ZSBTYPE
member type

ZSBDATEn
build date (in YYYYMMDD format)

ZSBTIME
build time (in HHMMSS format)

ZSBVER
version

ZSBLINE
unformatted data line (72 characters)

The maximum parameter length is 8 characters (except for ZSBLINE).

dd_msgs
The ddname indicating the destination of the messages generated by the GETBLDMP service. If you specify a blank ddname, SCLM routes the GETBLDMP messages to the default output device, such as your terminal. Otherwise, before you call the GETBLDMP service, you must allocate the ddname. The following attributes should be used: `RECFM=F, LRECL=80, BLKSIZE=80`. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

$msg_array
An output parameter pointing to the message array. See “Pointer parameter descriptions” on page 368 for more information about $msg_array. This parameter is used for FLMLNK only.

Example of an AUTHCODE report

This sample REXX code is an example of how to invoke the GETBLDMP service.

```
/* REXX */

   address ispexec
   'CONTROL ERRORS RETURN'
   'TBEND   BMAPTABL'
   'CONTROL ERRORS CANCEL'
   tbnames = 'NAMES(ZSBMGRP ZSBMTYP ZSBMMEM ZSBKWRD ZSBMEM ZSBTYPE
                    ZSBDATE ZSBTIME ZSBVER  ZSBLINE ZSBGRP)'
   'TBCREATE BMAPTABL' tbnames 'NOWRITE'
   'TBVCLEAR BMAPTABL'

   parmbmap = 'ADMIN10,ADMIN10,DEV1,COBOL,RDBK*'
   Address TSO 'FLMCMD GETBLDMP,'||parmbmap||',BMAPTABL'
   Say 'Return code from Build Map lookup = 'rc
   If RC < 5 Then
   Do
      'TBTOP  BMAPTABL'
      'TBSKIP BMAPTABL'

      Say ''
      Say 'BM Group BM Type  BM Memb ' ,
          'Keyword  Member   Type     Group    Date/Time Modified  Version'
```

```
      Do While (RC = 0)
         Say SUBSTR(ZSBMGRP,1,8) SUBSTR(ZSBMTYP,1,8) SUBSTR(ZSBMMEM,1,8) ,
             SUBSTR(ZSBKWRD,1,8) SUBSTR(ZSBMEM,1,8) SUBSTR(ZSBTYPE,1,8),
             SUBSTR(ZSBGRP,1,8) ZSBDATE'  'ZSBTIME '  ' ZSBVER '  '
         'TBSKIP BMAPTABL'
      End
      'TBEND BMAPTABL'
   End
```

This example builds the temporary ISPF table BMAPTABL with all of the available parameters. The service call is invoked for COBOL members beginning with RDBK, starting at the DEV1 group in the hierarchy. Once complete the temporary ISPF table is read to list out the build map entries.

Here is the output:

```
Return code from Build Map lookup = 0

BM Group BM Type  BM Memb  Keyword  Member   Type     Group  Date/Time Modified
Version
DEV1     COBOL    RDBKC02  SINC     RDBKC02  COBOL    DEV1   20070625  155226   1
DEV1     COBOL    RDBKC02  OBJ      RDBKC02  OBJ      DEV1   20070625  155357   1
DEV1     COBOL    RDBKC02  LIST     RDBKC02  LIST     DEV1   20070625  155357   1
DEV1     COBOL    RDBKC02  OUT1     RDBKC02  DBRM     DEV1   20070625  155357   1
DEV1     COBOL    RDBKC02  I1*      ORDRSET  COPYBOOK TEST   20070607  114654   1
DEV1     COBOL    RDBKC02  I1*      RDBKTB01 COPYBOOK TEST   20070522  165926   1
TEST     COBOL    RDBKC01  SINC     RDBKC01  COBOL    DEV1   20070612  143325   3
TEST     COBOL    RDBKC01  OBJ      RDBKC01  OBJ      TEST   20070625  155347   14
TEST     COBOL    RDBKC01  LIST     RDBKC01  LIST     TEST   20070625  155347   14
TEST     COBOL    RDBKC01  OUT1     RDBKC01  DBRM     TEST   20070625  155347   13
TEST     COBOL    RDBKC01  I1*      ORDRSET  COPYBOOK TEST   20070607  114654   1
TEST     COBOL    RDBKC01  I1*      RDBKTB01 COPYBOOK TEST   20070522  165926   1
TEST     COBOL    RDBKC01  I1*      RDBKDC01 DCLGEN   TEST   20070612  143307   1
```

Return codes

Additional special services messages are written to the FLMMSGS ddname. See “SCLM service messages” on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See “SCLM service return codes” on page 377 for more information.

Possible return codes are:

0
Normal completion. A build map record was found that exactly matched the specified criteria and the information was stored successfully.

4
Normal completion. A build map record was found at a higher level. The information was stored successfully.

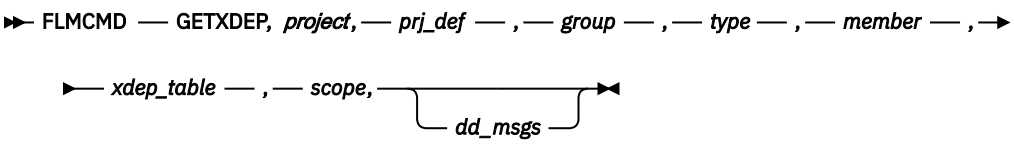
8
Error completion. No account record was found for the specified member.

12
Error completion. Refer to the messages for more information.

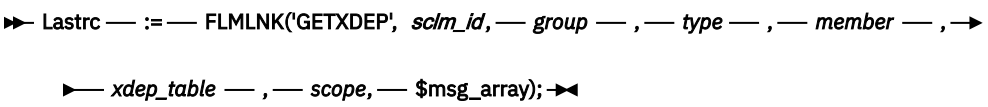
GETXDEP—return cross-dependency information

The GETXDEP service returns information from the Cross-dependency database associated with the specified SCLM-managed member. The information is returned in an ISPF table. One row is returned for each child-parent dependency found.

Command invocation format



Call invocation format



Parameters

project
The project name.

prj_def
The project definition name.

group
The group associated with the accounting record.

type
The type associated with the accounting record.

member
The member under SCLM control.

xdep_table
The name of the ISPF table to contain the cross-dependency entries. The table must be open before calling the GETXDEP service. A TBADD is performed for each cross-dependency record found. The following ISPF variables must be defined to the table in order to have their value stored in the table:

ZSFGROUP
Initial group (same on all records).

ZSFSCOPE
Scope (same on all records).

ZSFLEVEL
Nesting level.

ZSFCNAME[Note 1]
Child object - name.

ZSFCPGRP[Note 1]
Child object - group.

ZSFCTYPE[Note 1]
Child object - type.

ZSFCLANG[Note 1]
Child object - language.

ZSFCBGRP[Note 1]
Child object - group (buildmap).

ZSFCBLNG[Note 1]
Child object - language (buildmap).

ZSFCNUMP[Note 1]
Child object - number of parents.

ZSFPKWRD
Parent object - include keyword.

ZSFPNAME
Parent object - name.

ZSFPTYPE
Parent object - account type.

ZSFPGRP
Parent object - account group.

ZSFPBGRP
Parent object - buildmap group.

ZSFPLANG
Parent object - account language.

ZSFPBLNG
Parent object - buildmap language.

ZSFPNUMP
Parent object - number of parents.

ZSFLIMIT
Limited list = blank, "CIRC", or "MAX".

Note:

1. The ZSFC** fields are repeated for each of their parent rows.

scope
Valid values are:

NEXT
Just return immediate parents.

ALL
Return all parents.

LMOD
Stop at LECMAP archdefs.

TOPSRC
Stop at CCMAP archdefs.

dd_msgs
The ddname indicating the destination of the messages generated by the GETXDEP service. If you specify a blank ddname, SCLM routes the GETXDEP messages to the default output device, such as your terminal. Otherwise, before you call the GETXDEP service, you must allocate the ddname. The following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

$msg_array
An output parameter pointing to the message array. See the section "Pointer Parameter Descriptions" in Chapter 16,"Invoking the SCLM services" for more information about $msg_array. This parameter is used for FLMLNK only.

Return codes

0
GETXDEP completed successfully.

4
The parent chain was truncated when the maximum nesting level was exceeded, or a circular reference was detected. ZSFLIMIT is non-blank on truncated rows.

8
No parent data was found.

12
Cross-dependency database is not active.

16
Error updating ISPF table.

20
Error reading the Cross-dependency database.

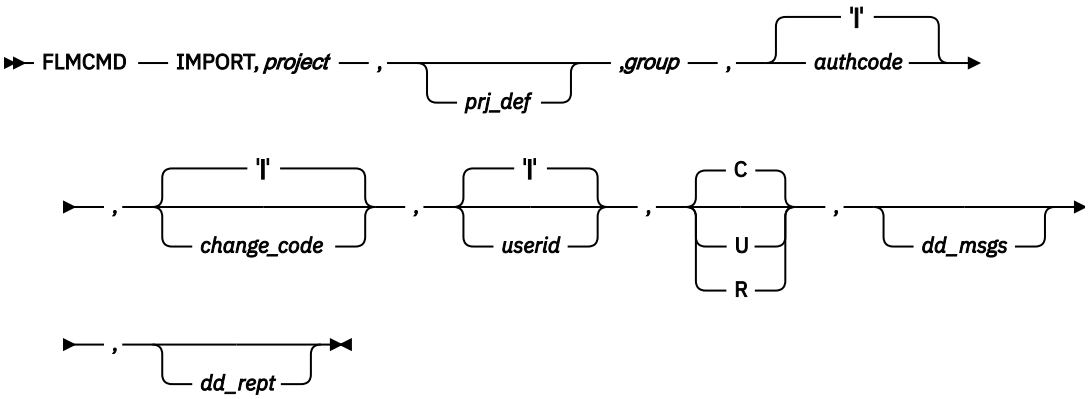
IMPORT—Import SCLM Accounting Information to Current Project

The IMPORT service reintroduces the exported SCLM accounting information into the context of the current project, after first verifying that this data corresponds to the current contents of the SCLM-controlled data sets.

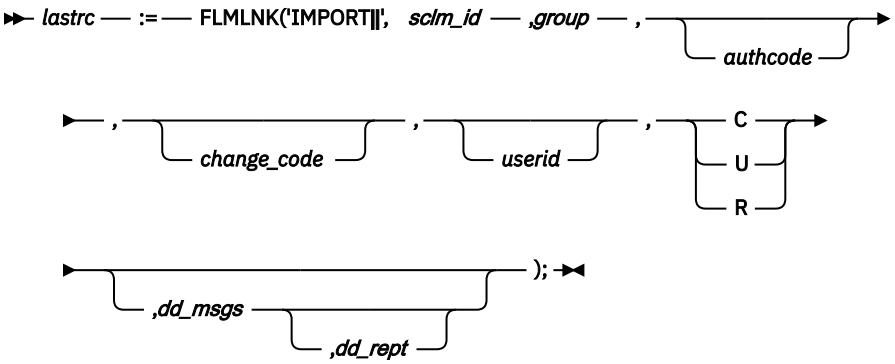
Like the SCLM editor, the IMPORT service verifies authorization codes and prohibits simultaneous updates of members. The group specified to receive the import must be a development group. The IMPORT service also ensures that all the software components to be imported are available and have correct accounting information. Finally, the IMPORT service verifies that each software component is either new or based directly on the version that exists in the higher group.

Note: Upon completion, the IMPORT service purges the EXPORT database of all records that were successfully imported.

Command invocation format



Call invocation format



ISPF interface panel

```
  Menu  SCLM  Utilities  Help
_______________________________________________________________________________
                    SCLM FLMCMD IMPORT Service - Entry Panel

SCLM Import Criteria:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1

Mode  . . _  1. Conditional
             2. Unconditional
             3. Report
Authorization code  . .  ________ (If blank, the default auth code is used)
Change code . . . . . .  ________

DD Names for output data sets:
   Error message data set  ________  (Blank to write messages to the terminal)
   Report data set . . . . ________  (Blank to write report to the terminal)



Command ===> ___________________________________________________________________
 F1=HELP      F2=          F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 197. IMPORT Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name used for the import. It defaults to the project parameter. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

sclm_id
An SCLM ID associated with a given project and project definition. The SCLM ID is generated by the INIT service. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

group
The group into which data is being imported. The maximum parameter length is 8 characters. The group must be defined in the project definition as a development group. As well, export accounting data sets must be defined for the group for import to work.

authcode
The authorization code to be used for the lock. SCLM uses the authorization code for the verification steps described in “LOCK—Lock a Member or Assign an Access Key” on page 430. If you do not supply an authcode for FLMCMD, or if you specify a blank for either FLMCMD or FLMLNK, SCLM uses the authorization code from the exported accounting information. The maximum parameter length is 8 characters.

change_code
If you have a change code verification routine, when you specify this parameter, you must ensure that the change code is valid. When you specify a valid change code, the IMPORT service adds the change code to each editable member's accounting record and updates the change code date and time to the change date and time from the exported accounting record. If you specify a change code that is already listed in a member's exported accounting record, the IMPORT service does not add a duplicate change code to the accounting record. It uses the one from the exported accounting record. For FLMCMD, the default value is blank; unless a change code is specified, the IMPORT service will not perform verification. There is no default value for FLMLNK.

userid
If you supply a value for this parameter, SCLM replaces the USERID field in each exported accounting record with the value supplied. If you do not specify a value, SCLM uses the user ID from the exported accounting information. The maximum parameter length is 8 characters. If no value is specified for FLMCMD or blank is specified for either FLMCMD or FLMLNK, the user ID from the exported accounting information will be used.

C|U|R
Indicates the import mode, where C=conditional, U=unconditional, and R=report. The maximum parameter length is 24 characters.

When you specify C, the IMPORT service attempts to import the specified group only when each accounting record and build map record successfully passes all the necessary verifications. The IMPORT service fails if any one of these records cannot pass verification. Thus, when you specify conditional mode, the IMPORT service imports all records or none. The IMPORT service deletes the record from the export database once it has been imported successfully into the specified group.

When you specify U, the IMPORT service performs the same set of verifications, but attempts to import the group even if one or more records do not pass verification. In this case, the IMPORT service imports only those records that passed verification and leaves the records that failed verification in the export database. In addition, IMPORT attempts to store an accounting record with a predecessor baseline date/time verification error. The IMPORT service deletes the record from the export database once it has been imported successfully into the specified group.

When you specify R, the IMPORT service performs the verification and reports the eligibility of members for import. For FLMCMD, the default value is C. There is no default value for FLMLNK.

dd_msgs
The ddname indicating the destination of the import messages. If you specify a blank ddname, SCLM routes the import messages to the default output device, such as your terminal. Otherwise, before you call the IMPORT service, you must allocate the ddname; the following attributes should be used: `RECFM=F, LRECL=80, BLKSIZE=80`. The maximum parameter length is 8 characters.

dd_rept
The ddname indicating the destination of the import report. If you specify a blank ddname, SCLM routes the import messages to the default output device, such as your terminal. Otherwise, before you call the IMPORT service, you must allocate the ddname; the following attributes should be used: `RECFM=F, LRECL=80, BLKSIZE=80`. The maximum parameter length is 8 characters.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See "SCLM service messages" on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM service return codes" on page 377 for more information.

Possible return codes are:

0
Normal completion. See the SCLM messages for more information.

4
Warning condition. See the SCLM messages for more information.

8
Error condition. See the SCLM messages for more information.

12
Severe error condition. SCLM does not produce messages because there was an error invoking the IMPORT module.

16
Severe error condition. SCLM does not produce messages because it was unable to retrieve SCLM ID information.

Example of command invocation

```
FLMCMD IMPORT,PROJ1,,USER1,,,,C
```

This service command imports data into the USER1 group in the PROJ1 project in conditional mode. SCLM sends messages and listings to the terminal.

Example of call invocation

This example shows general syntax. Call invocations are language-specific. See Chapter 20, “Sample programs using SCLM services,” on page 477 for specific examples.

```
lastrc := FLMLNK('IMPORT  ',                (* service            *)
                 sclm_id,                   (* SCLM ID            *)
                 'USER1   ',                (* group              *)
                 '        ',                (* authorization code *)
                 '        ',                (* change code        *)
                 '        ',                (* user ID            *)
                 'C                  ',     (* mode               *)
                 'MESSAGES',                (* messages           *)
                 'REPORT ');                (* report             *)
```

This service call imports the USER1 group in conditional mode. The sclm_id parameter contains a valid SCLM ID returned from the INIT service. The ddnames for the messages and report (MESSAGES and REPORT respectively) must be allocated before calling FLMLNK.

INIT—Generate an SCLM ID

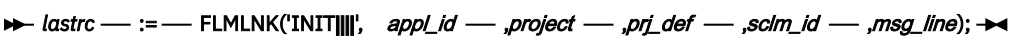
The INIT service initializes an SCLM ID. During this process, it also initializes the specified project definition. The INIT service also checks to make sure that the project definition is current. The project definition macros must be reassembled after installing SCLM 3.5. If the macros have not been reassembled, SCLM issues an error message. After the INIT service generates an SCLM ID, it can be passed to other SCLM services, such as DELETE and LOCK. Each INIT service invocation needs a matching FREE service invocation.

Note: SCLM maintains allocations of data sets in the hierarchy between uses of SCLM services. This enhances the performance of SCLM; however, if data sets in the hierarchy are created or deleted, the FREE service will need to be invoked to release the existing allocations and a new INIT service invoked to regain access to the project definition.

Command invocation format

You cannot use command procedures to call this service.

Call invocation format



Parameters

appl_id
The application ID with which the generated SCLM ID is to be associated. The application ID must be generated by the START service. The maximum parameter length is 8 characters.

project
The project name. The maximum parameter length is 8 characters.

prj_def
The project definition name to be initialized for the SCLM ID. The maximum parameter length is 8 characters.

sclm_id
The generated SCLM ID. Each time you invoke the INIT service, it generates a unique SCLM ID. The maximum parameter length is 8 characters.

msg_line
An output parameter that is a buffer containing any INIT service error message. The maximum parameter length is 80 characters.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See "SCLM service messages" on page 378 for more information.

Other return codes might be produced by the FLMLNK processor. See "SCLM service return codes" on page 377 for more information.

Possible return codes are:

0
Normal completion.

8
Error condition. See the msg_line parameter description for more details.

Example of call invocation

This example shows general syntax. Call invocations are language-specific. See Chapter 20, "Sample programs using SCLM services," on page 477 for specific examples.

```
lastrc := FLMLNK('INIT    ',       (* service                *)
                  appl_id,         (* application ID          *)
                  'PROJ1   ',      (* project name            *)
                  'PROJ1   ',      (* project definition name *)
                  sclm_id,         (* SCLM ID                 *)
                  msg_line);       (* error messages          *)
```

This service call initializes an SCLM ID for the PROJ1 project using the PROJ1 project definition. The appl_id parameter contains a valid application ID returned from the START service. SCLM returns messages in the msg_line parameter.

LOCK—Lock a Member or Assign an Access Key

The LOCK service locks a member in a development library, assigns the member an access key, or both. In most cases, LOCK allows one member to be modified by only one user at a time. Locking a member also ensures that updates to the member can occur only in the specified development library until you unlock or promote the member. The member to be locked does not have to exist in a development library or anywhere in the SCLM project hierarchy.

Suppose you are creating a new member on your programmable workstation. You can use LOCK to reserve the member name for future use.

You can assign an access key to the member to make the member even more secure than just locking it does. If you assign an access key to a member, you must, thereafter, provide that access key to further modify the member. When using access keys, remember:

- Access keys have no effect on the BUILD, DBACCT, DBUTIL, PARSE, and RPTARCH services.
- You must supply the correct member access key when you call the DELETE, SAVE, STORE, and UNLOCK services.
- Before you can promote a member, you must call the UNLOCK service to remove a member's access key. The PROMOTE service promotes any member that has a blank access key.
- If you have successfully completed the SAVE or STORE service for a member, the member remains locked. You can still use the LOCK service to assign an access key to the member.

In most cases, LOCK allows one member to be modified by only one user at a time (see Note). When you edit a member in one development library, LOCK prohibits others from editing the same member in their development libraries. Another user cannot edit the member until you delete the member and its accounting information from your group or you promote the member to a common group.

Note: Depending upon the software configuration management plan for a project, a temporary copy of a member could exist in two development libraries at the same time. See “Step 3: Establish authorization codes” on page 8 for more information, or see the project manager for the project.

The LOCK service provides the following capabilities:

- Verifying a group

  LOCK verifies that the group specified is valid. Group verification allows SCLM to control all source modifications to the higher groups of the hierarchy through the promote function.

- Verifying an authorization code

  The project administrator defines a list of *authorization codes* to each group in the project's database. An authorization code is an identifier that SCLM uses to control authority to update and promote members within a hierarchy.

  The LOCK service can only lock those members in the group that are assigned one of the authorization codes defined to the group. See “FLMGROUP macro” on page 540 for more information.

- Verifying predecessors

  The LOCK service guarantees that the member to be locked in the development library is the most current version of the member within the hierarchical view. *Predecessors* of the member are previous versions of a member existing within the same hierarchical view.

  The LOCK service ensures that the member to be locked does not overlay changes to a predecessor. LOCK does this by verifying that the predecessor of each version of the member within the hierarchical view has not been modified.

- Verifying build output

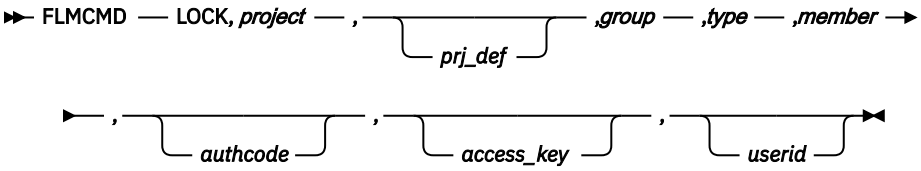
  You cannot lock members that are outputs of a build. This verification prevents accidental modification of a build output member, such as text files and compiler listings. (These members are referred to as "noneditable" elsewhere in this document.)

- Verifying access keys

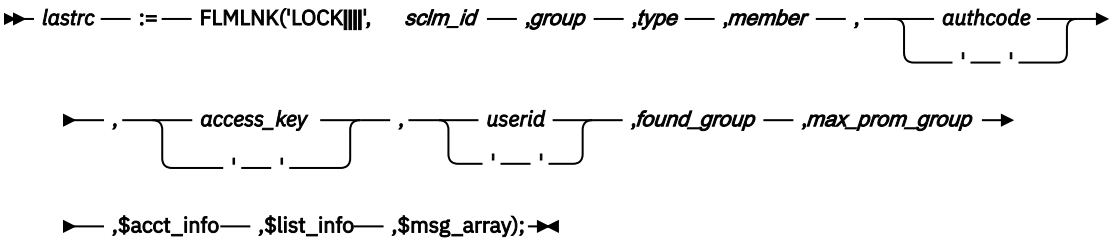
  The LOCK service also prevents you from accidentally modifying or deleting a member you do not control. The access key that you store with the accounting information for a member provides this verification. Locking a member with an access key allows you to prevent others from accidentally modifying or promoting the member if they make changes while working outside of SCLM.

  Use the access key as a signal to other developers, not as a security measure. For example, you can use the access key to indicate the location of the member or the reason it was locked.

Command invocation format



Call invocation format



ISPF interface panel

```
  Menu  SCLM  Utilities  Help
_______________________________________________________________________________
                    SCLM FLMCMD LOCK Service - Entry Panel

SCLM Library:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1____
   Type  . . . . SOURCE__
   Member  . . . ________

Authorization code  . . ________  (If blank, the default auth code is used)
Access key  . . . . . . ________________
User id . . . . . . . . ________  (If blank, your user id is used)









Command ===> ____________________________________________________________________
 F1=HELP      F2=          F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 198. LOCK Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name to be used for the lock. It defaults to project. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

group
The group in which the member is to be locked. The specified group must be a development library. The maximum parameter length is 8 characters.

type
The type containing the member to be locked. The maximum parameter length is 8 characters.

member
The member to be locked. The maximum parameter length is 8 characters.

authcode
The authorization code to be used for the lock. If you do not supply an authcode, SCLM uses one of the following default values:

- The authorization code from the existing member if the member being locked exists in the hierarchy
- The default authorization code for the group if the member does not exist in the hierarchy.

The maximum parameter length is 8 characters.

access_key
The access key to be assigned to the member. It defaults to blank. The maximum parameter length is 16 characters. You must use the access key for any further manipulation of the member until you use the UNLOCK service to remove the access key.

userid
User ID of the person requesting the lock. It defaults to the current system user ID. The maximum parameter length is 8 characters.

found_group
An output parameter that indicates the group in which the first occurrence of the member exists within the hierarchy. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

max_prom_group
An output parameter that indicates the highest group in the hierarchy to which the member can be promoted. This member's maximum promotable group is based on the authorization code you use for the lock. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

$acct_info
An output parameter pointing to a record containing the static portion of the member's accounting record. See “$acct_info” on page 368 for more details. This parameter is used for FLMLNK only.

$list_info
An output parameter pointing to an array of records that contains the dynamic portion of the member's accounting record. See “$list_info” on page 371 for more details. This parameter is used for FLMLNK only.

$msg_array
An output parameter pointing to the message array. See “Pointer parameter descriptions” on page 368 for more information about $msg_array. This parameter is used for FLMLNK only.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See “SCLM service messages” on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See “SCLM service return codes” on page 377 for more information.

Possible return codes are:

0
Normal completion. If a member is already locked, and no information concerning the lock has changed (the change code, or language, for example), then no action will be taken, but the return code will still be 0. No audit or versioning records will be written in this case.

8
Error condition. The $msg_array parameter contains the error message associated with this condition.

Example of command invocation

```
FLMCMD LOCK,PROJ1,,USER1,SOURCE,FLM01MD2,,XXX#04
```

This service command locks the FLM01MD2 member of the SOURCE type in the USER1 group. The project name is PROJ1. The access key to be assigned to the member is XXX#04. The authcode and user ID parameters are defaults.

Example of call invocation

This example shows general syntax. Call invocations are language-specific. See Chapter 20, “Sample programs using SCLM services,” on page 477 for specific examples.

```
lastrc := FLMLNK('LOCK    ',            (* service                  *)
                 sclm_id,               (* SCLM ID                  *)
                 'USER1   ',            (* group                    *)
                 'SOURCE  ',            (* type                     *)
                 'FLM01MD2',            (* member                   *)
                 'TESTAC  ',            (* authorization code       *)
                 'XXX#04          ',    (* access key               *)
                 '        ',            (* user ID                  *)
                 found_group,           (* found group              *)
                 max_prom_group,        (* maximum promotable group *)
                 $acct_info,            (* accounting information pointer *)
                 $list_info,            (* list information pointer *)
                 $msg_array);           (* message array pointer    *)
```

This service call locks the FLM01MD2 member of the SOURCE type in the USER1 group. The sclm_id parameter contains a valid SCLM ID returned from the INIT service. The authorization code to be used for the lock verification is TESTAC and the access key is XXX#04. USERID is the user requesting the lock. SCLM returns all messages in the $msg_array parameter.

MIGRATE—Create Accounting for Selected Members

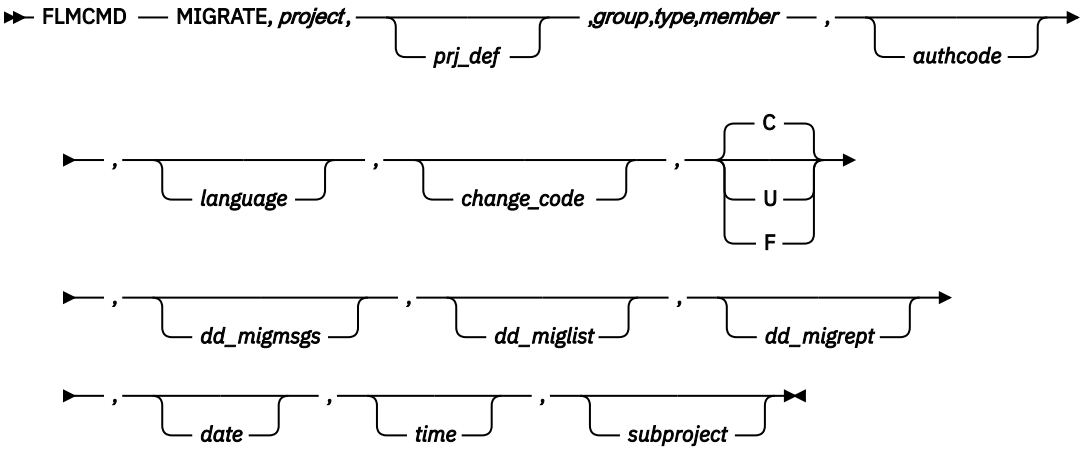
The MIGRATE service creates or updates SCLM accounting information for members in a development library that match a given pattern.

MIGRATE checks each member whose name matches the pattern for valid SCLM accounting information. If a selected member does not have valid accounting information or if Forced mode is specified, MIGRATE invokes the SAVE service to lock, parse, and store the member. All of the rules and restrictions that apply to the SAVE service also apply to the MIGRATE service.

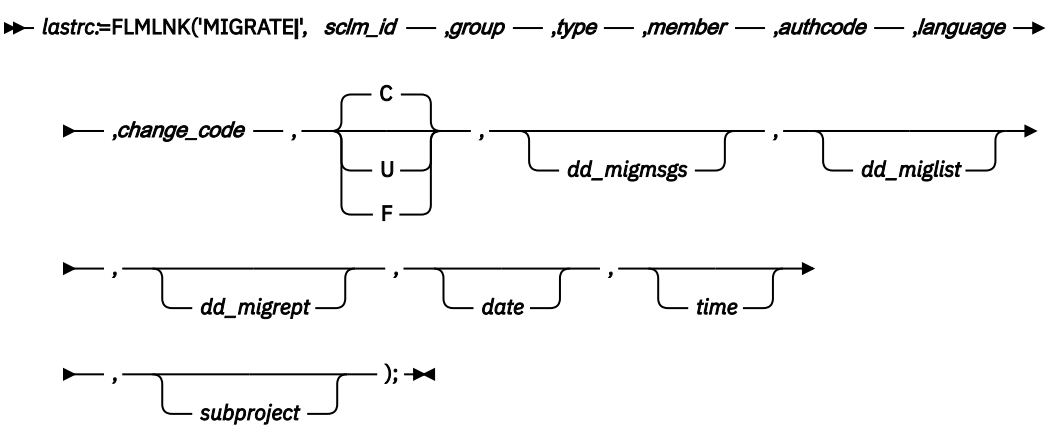
Note: The MIGRATE service does not parse a member correctly if the member is packed. Make sure that the pack mode is off in the member's profile.

For more information about the SAVE, LOCK, PARSE, and STORE services, see their service descriptions in this chapter.

Command invocation format



Call invocation format



ISPF interface panel

```
  Menu  SCLM  Utilities  Help
_______________________________________________________________________________
                  SCLM FLMCMD MIGRATE Service - Entry Panel
                                                                        More:     +
SCLM Library:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1
   Type  . . . . SOURCE
   Member  . . . ________

Authorization code  . . ________  (If blank, the default auth code is used)
Language  . . . . . . . COB
Change code . . . . . . ________

Mode  . . _  1. Conditional
             2. Unconditional
             3. Forced

Accounting date for migrate . . . .  __________ (Blank to use current date)
Accounting time for migrate . . . .  _________ (Blank to use current time)
Command ===> ____________________________________________________________________
 F1=HELP      F2=          F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 199. MIGRATE Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters.

prj_def
The project definition name to be used for the lock, parse, and store of migrated members. It defaults to the project parameter. The maximum parameter length is 8 characters.

group
The group in which the migration is to occur. The specified group must be a development group. The maximum parameter length is 8 characters.

type
The type containing the members. The maximum parameter length is 8 characters.

member
A pattern used to select the members to be migrated. The maximum parameter length is 10 characters. You must specify a valid member name or valid pattern, or an error message appears.

authcode
The authorization code to be used for locking selected members. If you do not supply an authcode or the authcode is blank, SCLM uses default values as follows:

- The authorization code from the existing member if the member being migrated exists in the hierarchy
- The default authorization code for the group if the member does not exist in the hierarchy.

The maximum parameter length is 8 characters.

language
The language of the member. The maximum parameter length is 8 characters. You must specify the language the first time you save a member.

change_code
A change_code to be added to the information obtained by parsing the member. If the member's accounting record lists the change_code, SCLM updates the date and time stamps for the existing change_code entry. The maximum parameter length is 8 characters.

C|U|F
Indicates the migrate mode (C=Conditional; U=Unconditional; F=Forced). The maximum parameter length is 24 characters. The default value for FLMCMD is C. There is no default value for FLMLNK.

dd_migmsgs
The ddname indicating the destination of the messages generated by the MIGRATE service. If you specify a blank ddname, SCLM routes the MIGRATE service messages to the default output device, such as your terminal. Otherwise, before you call the MIGRATE service, you must allocate the ddname. The following attributes should be used: `RECFM=F, LRECL=80, BLKSIZE=80`. The maximum parameter length is 8 characters.

dd_miglist
The ddname indicating the destination of the parser listings. If you specify a blank ddname, SCLM does not generate the parser listings. The maximum parameter length is 8 characters.

If the parser for the specified language does not produce a listing, specify a blank ddname. The language parsers supplied by SCLM do not produce a listing. If the parser for the specified language does produce a listing and you specified a ddname, allocate the ddname with the attributes required by the parser. Project-specific parsers can produce a listing. See “FLMTRNSL macro” on page 557 for more information about project-defined parsers.

dd_migrept
The ddname indicating the destination of the migrate report. If you specify a blank ddname, SCLM routes the migrate report to the default output device, such as your terminal. Otherwise, before you call the MIGRATE service, you must allocate the ddname; the following attributes should be used: `RECFM=FBA, LRECL=80, BLKSIZE=3120`. The maximum parameter length is 8 characters.

date
The date to assign to the accounting record and member statistics. Use this field if you want to keep audit records and versions from another library system. The date defaults to the current date. This parameter is required if the "time" parameter is entered. The parameter length is 10 characters. The date, with a 4-character year, must be specified in the national language format.

time
The time to assign to the accounting record and member statistics. This parameter is required if the "date" parameter is entered. The time must be specified in the national language format. The parameter length is 8 characters.

subproject
The name of the subproject to be assigned to the member. This parameter must only be specified if the SCLM project has been defined with subprojects. For more information about subprojects, see Chapter 17, “SCLM security,” on page 337.

The maximum parameter length is 8 characters.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See “SCLM service messages” on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See “SCLM service return codes” on page 377 for more information.

Possible return codes are:

0
Normal completion.

4
Warning condition. See the SCLM messages for more information.

8
Error condition. See the SCLM messages for more information.

Example of command invocation

```
FLMCMD MIGRATE,PROJ1,,USER1,SOURCE,MOD*,TESTAC,COBOL,CC001234,,,PARSEDD
```

This service command migrates (locks, parses, and stores accounting information) members with names beginning MOD (such as MOD1, MOD2, or MODULE) of the type SOURCE in the USER1 group. The project name is PROJ1 and the authorization code is TESTAC. Change code CC001234 is to be added to the information obtained by parsing the member with the COBOL parser.

SCLM copies parser listings to the PARSEDD ddname only if errors occur. You must allocate the PARSEDD ddname before you call the service.

Messages generated by the MIGRATE service appear on the default output device. This is probably the terminal if you are running under a foreground TSO session.

Example of call invocation

Note: This example shows general syntax. Call invocations are language-specific. See Chapter 20, "Sample programs using SCLM services," on page 477 for specific examples.

```
lastrc := FLMLNK('MIGRATE ',   (* service              *)
        scln_id,               (* application ID       *)
        'RELEASE  ',           (* group                *)
        'SOURCE   ',           (* type                 *)
        '*        ',           (* all members          *)
        '         ',           (* default authcode     *)
        'HLASM    ',           (* language             *)
        'INIT     ',           (* change code          *)
        'C        ',           (* conditional          *)
        'MSGSDD   ',           (* messages ddname      *)
        'LISTDD   ',           (* list ddname          *)
        'REPTDD   ';           (* report ddname        *)
```

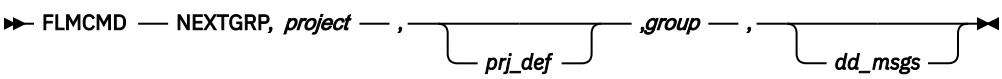
This service call migrates all members of the SOURCE type and RELEASE group in the project. Each member's accounting record has the default authcode, as defined in the project definition. All members have a language of HLASM and a change code of INIT. Any messages are written to the data set allocated to MSGSDD, and the migrate report appears in the data set allocated to REPTDD. Any parser errors are written to the data set allocated to LISTDD.

This service call initializes an SCLM ID for the PROJ1 project using the PROJ1 definition. The appl_id parameter contains a valid application ID returned from the START service. SCLM returns messages in the msg_line parameter.

NEXTGRP— Retrieve the Next Group in an SCLM Hierarchy

The NEXTGRP service returns the name of the next group in a given hierarchy. For a given group, the next group is returned in the SHARED pool variable ZSNXTGRP. An indicator whether the group is key or non-key is returned in SHARED pool variable ZSNGPKEY. The possible values for ZSNGPKEY are KEY for key groups, and NONKEY for non-key groups.

Command invocation format



Call invocation format

ISPF interface panel

```
  Menu  SCLM  Utilities  Help
_______________________________________________________________________________
                  SCLM FLMCMD NEXTGRP Service - Entry Panel

SCLM Library Input:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1____

DD Name for output data set:
   Error message data set ________  (Blank to write messages to the terminal)









Command ===> ____________________________________________________________________
 F1=HELP      F2=           F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT     F12=SUBMIT
```

Figure 200. NEXTGRP Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters.

prj_def
The project definition name to be used for NEXTGRP. It defaults to the project parameter. The maximum parameter length is 8 characters.

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters.

group
The group for which the "next" group is to be found. The maximum parameter length is 8 characters.

dd_msgs
The ddname indicating the destination of the messages generated by the NEXTGRP service. The maximum parameter length is 8 characters.

Return codes

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See “SCLM service return codes” on page 377 for more information.

Possible return codes are:

0
Normal completion. NEXTGRP completed successfully. Variables are set.

4
Warning condition. The group is already the top group. No variables are set.

8
Error condition. Invalid project, prj_def, or group name.

12
Severe error condition. SCLM might not produce messages because there was an error invoking the NEXTGRP module. For some conditions, messages are available.

Example of command invocation

The following REXX exec begins at group USER and finds each successive group in the hierarchy defined by the SCLM7010 alternate of the SCLM70 project.

```
/* REXX exec to find the next groups in a hierarchy     */
TRACE off
address ispexec
group = 'USER'
done = 'false'
address 'TSO' 'alloc fi(ddm) da(sclm.msgs) shr mod'
do until done = 'true'
   'select cmd(FLMCMD NEXTGRP,SCLM70,SCLM7010,'group',ddm)'
   if rc > 0 then
      do
        done = 'true'
      end
   else
     do
       'vget (zsnxtgrp,zsngpkey) shared'
       say 'For group' group 'the next group is' zsnxtgrp zsngpkey
       group = zsnxtgrp
    end
end
address 'TSO' 'free fi(ddm)'
```

Executing this example produces this output:

```
For group USER the next group is STGE KEY
For group STGE the next group is DEV KEY
For group DEV the next group is INT KEY
For group INT the next group is REL KEY
For group REL the next group is BASE KEY
```

Example of call invocation

Note: This example shows general syntax. Call invocations are language-specific. See Chapter 20, “Sample programs using SCLM services,” on page 477 for specific examples.

This program fragment uses the NEXTGRP service to find the group that USER promotes into. The variables ZSNXTGRP and ZSNGPKEY are VDEFINEd to local program variables, and the values set by the NEXTGRP service are retrieved from the shared pool by the VGET service. The example assumes that the START and INIT services have already completed successfully, so that the SLMID value is valid. The ddname DDMSGS has been allocated to a data set with valid characteristics.

```
CALL FLMLNK('NEXTGRP ',SLMID,'USER ',DDMSGS)
    RETCODE(R15);
    EVAL(8),' ',' ');
CALL ISPLINK ('VDEFINE ', 'ZSNXTGRP', ZSNXTGRP, 'CHAR ',
    EVAL(8),' ',' ');
CALL ISPLINK ('VDEFINE ', 'ZSNGPKEY', ZSNGPKEY, 'CHAR ',
CALL ISPLINK ('VGET ', 'ZSNGPKEY', 'SHARED ');
CALL ISPLINK ('VGET ', 'ZSNXTGRP', 'SHARED ');
```

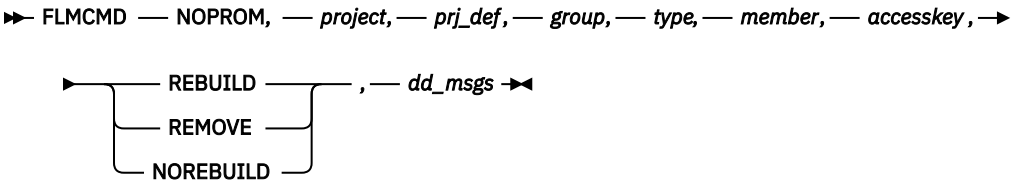
NOPROM—Change Promote Processing

The NOPROM service modifies the accounting status of a member's accounting record to change the processing of a member when it is copied to the next level during promotion.

For more information on this option, see Chapter 15, “Leaving a Member Behind on Promotion,” on page 319.

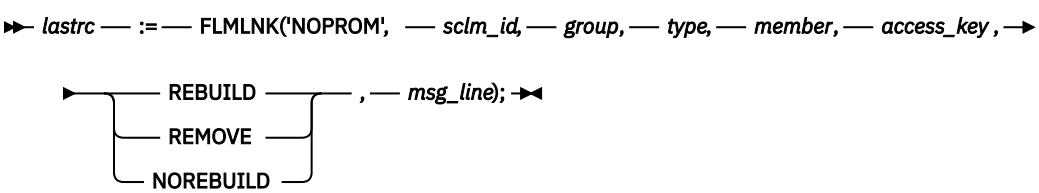
Command invocation format

The Format of the FLMCMD NOPROM service is:



Call invocation format

The Format of the FLMLNK NOPROM service is:



Parameters

project
The project name. The maximum parameter length is 8 characters.

prj_def
The project definition name. It defaults to the project name. The maximum parameter length is 8 characters.

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters.

group
The group at which the member accounting status is to be changed. The maximum parameter length is 8 characters.

type
The type of member whose accounting status is to be changed. The maximum parameter length is 8 characters.

member
Specifies the member whose accounting status is to be changed.

access_key
The access key that is assigned to the member. If the member is locked and the access key is not provided, then the accounting record cannot be updated. The maximum parameter length is 16 characters.

NOREBUILD|REBUILD|REMOVE
NOREBUILD indicates that SCLM is to update the accounting record to specify an accounting status of NOPROM-N. After building the appropriate member, SCLM while promoting copies the build maps containing the non-promoted member but the member itself is left behind. If the build map references the member not being promoted, this option causes the build map to be copied during the promote, regardless of the FLMLRBLD macro.

REBUILD indicates that SCLM is to update the accounting record to specify an accounting status of NOPROM-R. After building the appropriate member, SCLM while promoting does not copy the build maps containing the non-promoted member or the member itself. Once the copy phase of the promote is complete, the build function is invoked at the level being promoted into. Because the build maps are missing for members that reference the member not being promoted, those members, and any dependent members, are rebuilt. This option allows you to rebuild at the next level using the editable member found at that level or above it in the hierarchy.

REMOVE indicates that SCLM is to update the accounting record to specify an accounting status of EDITABLE. SCLM performs a normal promote of the member.

The maximum parameter length is 10 characters.

dd_msgs
The ddname indicating the destination of the messages generated by the NOPROM service. If you specify a blank ddname, SCLM routes the NOPROM messages to the default output device, such as your terminal. Otherwise, before you call the NOPROM service, you must allocate the ddname. Use the following attributes: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

msg_line
Services that only write one message have a msg_line parameter. Define a program variable that is 80 characters to hold the contents of this message line. This parameter only applies to services called through the FLMLNK interface.

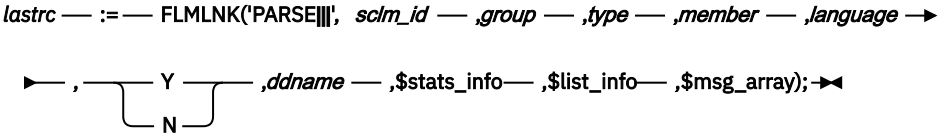
PARSE—Parse a Member for Statistical and Dependency Information

The PARSE service parses a member for statistical and dependency information. SCLM returns two buffers containing the member's vital information that you can pass on to the STORE service. When the STORE service receives this information, it places it in the member's accounting record.

Command invocation format

You cannot use command procedures to call this service.

Call invocation format



Parameters

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters.

group
The group in which the member is to be parsed. The maximum parameter length is 8 characters. Note that a member can be parsed in any group; the specified group does not have to be a development library.

type
The type containing the member to be parsed. The maximum parameter length is 8 characters.

member
The member to be parsed. The maximum parameter length is 8 characters.

language
The language used to identify the parser that will be invoked for the member. The maximum parameter length is 8 characters.

Y|N
Y indicates that parser listings are to be copied to the ddname parameter only if parser errors occur. N indicates that all parser listings are to be copied to the ddname. The maximum parameter length is 24 characters.

If the parser for the specified language does not produce a listing, specify Y. (The language parsers supplied by SCLM do not produce a listing.) If the parser for the specified language does produce a listing, specify either value. For more efficient performance, specify Y. Project-specific parsers can produce a listing.

ddname
The ddname indicating the destination of the parser listings. If you specify a blank ddname, SCLM does not generate parser listings. The maximum parameter length is 8 characters.

If the parser for the specified language does not produce a listing, specify a blank ddname. The parsers supplied by SCLM do not produce a listing. If the parser for the specified language does produce a listing and you specify a ddname, allocate the ddname with the attributes the parser requires. Project-specific parsers can produce a listing.

$stats_info
An output parameter pointing to a record containing the member's statistical information derived from parsing the member. See “$stats_info” on page 370 for more details.

$list_info
An output parameter pointing to an array of records that contains the member's include, change code, and user entry information derived from parsing the member. See “$list_info” on page 371 for more details.

$msg_array
An output parameter pointing to the message array. See “Pointer parameter descriptions” on page 368 for more information about $msg_array.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See “SCLM service messages” on page 378 for more information.

Other return codes might be produced by the FLMCMD processor. See “SCLM service return codes” on page 377 for more information.

Possible return codes are:

0
Normal completion.

4
Warning condition. A parser error occurred.

8
Error condition. The $msg_array parameter contains the error message associated with this condition.

Example of call invocation

This example shows general syntax. Call invocations are language-specific. See Chapter 20, “Sample programs using SCLM services,” on page 477 for specific examples.

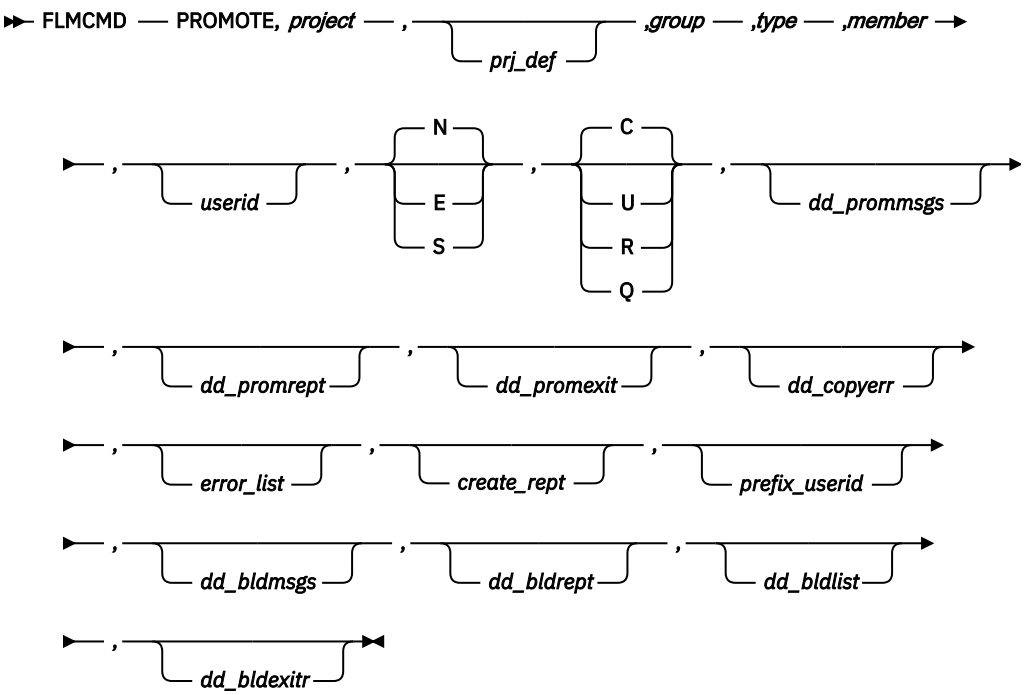
```
lastrc := FLMLNK('PARSE   ',              (* service            *)
              sclm_id,                    (* SCLM ID            *)
              'USER1   ',                 (* group              *)
              'SOURCE  ',                 (* type               *)
              'FLM01MD2',                 (* member             *)
              'PASCAL  ',                 (* language           *)
              'Y                     ',   (* listings           *)
              'PARSEDD ',                 (* ddname of listings *)
              $stats_info,                (* statistical information pointer *)
              $list_info,                 (* list information pointer *)
              $msg_array);                (* message array pointer *)
```

This service call parses the FLM01MD2 member of the SOURCE type in the USER1 group. The sclm_id contains a valid SCLM ID returned from the INIT service. SCLM uses the PASCAL parser and copies the parser listings to the PARSEDD ddname only if errors occur. You must allocate the PARSEDD ddname before you call FLMLNK. SCLM returns the parse results in the $stats_info and $list_info parameters and all messages in the $msg_array parameter.

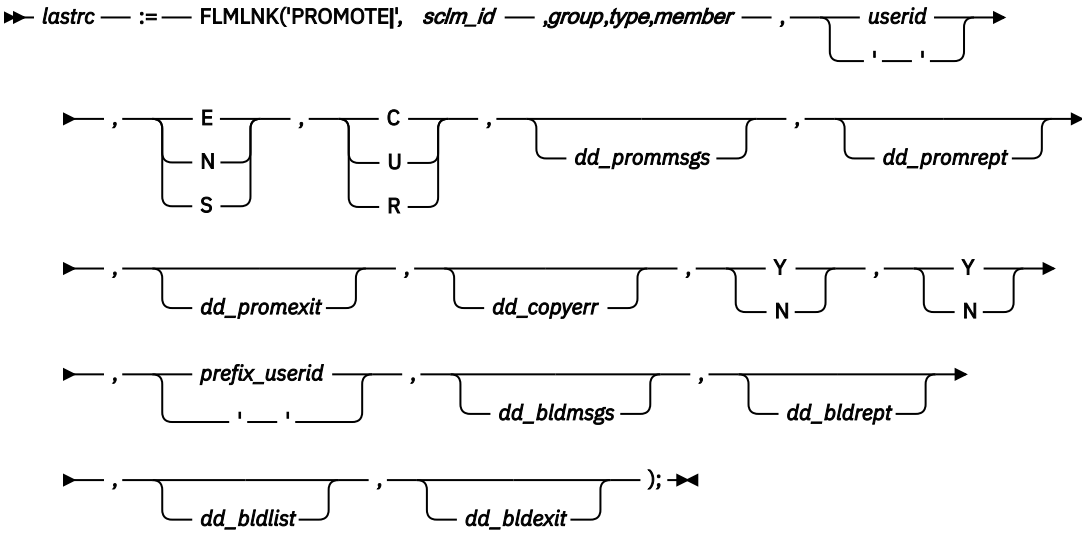
PROMOTE—Promote a Member from One Library to Another

The PROMOTE service moves data, that is, promotes data through the project database according to a project's architecture definition and project definition. Before SCLM can promote a member, it must have a blank access key and must have successfully completed the BUILD service. If a member has an access key, you must call the UNLOCK service to reset the access key before you can promote the member.

Command invocation format



Call invocation format



Notes:

[1] Comma separators are only required up to and before the last optional parameter specified.

ISPF interface panel

```
  Menu  SCLM  Utilities  Help
──────────────────────────────────────────────────────────────────────
                  SCLM FLMCMD PROMOTE Service - Entry Panel
                                                                More:     +
SCLM Library:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1____
   Type  . . . . SOURCE__
   Member  . . . ________

User id . . . . . . . . . . . . . ________ (If blank, your user id is used)
Prefix for temporary data sets   ________ (Blank to default to user id)

Mode  . . _  1. Conditional       Scope . . _  1. Limited
             2. Unconditional                  2. Normal
             3. Forced                         3. Subunit
             4. Report                         4. Extended

DD Names for output data sets:
   Error message data set ________  (Blank to write messages to the terminal)
Command ===> ____________________________________________________________
 F1=HELP      F2=          F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 201. PROMOTE Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name to be used for the promote. It defaults to project. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

group
The group the promote occurs from. The maximum parameter length is 8 characters.

type
The type containing the member to be promoted. The maximum parameter length is 8 characters.

member
The name of the architecture member or source member to be promoted. The maximum parameter length is 8 characters.

userid
The user ID of the person requesting the promote. If no value is specified for FLMCMD or a blank (' ') is specified for FLMLNK, it defaults to your TSO prefix or user ID if no TSO prefix has been created. The maximum parameter length is 8 characters.

E|N|S
Indicates the promote scope (E=extended, N=normal, S=subunit). The maximum parameter length is 24 characters. The default value for FLMCMD is N. There is no default value for FLMLNK.

C|R|U|Q
Indicates the promote mode (C=conditional, R=Report, U=Unconditional, Q=Report Unconditional). The maximum parameter length is 24 characters. The default value for FLMCMD is C. There is no default value for FLMLNK. Q is not supported for FLMLNK.

dd_prommsgs
The ddname indicating the destination of the promote messages. If you specify a blank ddname, SCLM routes the promote messages to the default output device, such as your terminal. Otherwise, before you call the PROMOTE service, you must allocate the ddname. The following attributes should be used: `DISP=MOD, RECFM=F, LRECL=80, BLKSIZE=80`. The maximum parameter length is 8 characters.

dd_promrept
The ddname indicating the destination of the promote report. If you specify a blank ddname, SCLM routes the promote report to the default output device, such as your terminal. Otherwise, before you call the PROMOTE service, you must allocate the ddname. The following attributes should be used: `RECFM=FBA, LRECL=80, BLKSIZE=3120`. The maximum parameter length is 8 characters.

dd_promexit
The ddname indicating the destination of the promote user exit data. Specify this parameter only if your project administrator defined a promote user exit routine in your project definition. Ask your project manager if your project is using a promote user exit routine. If you specify a blank ddname, SCLM routes the promote user exit data to NULLFILE. Otherwise, before you call the PROMOTE service, you must allocate the ddname. The following attributes should be used: `RECFM=FB, LRECL=160, BLKSIZE=3200`. The maximum parameter length is 8 characters.

dd_copyerr
The ddname indicating the destination of the promote copy error information. The promote copy error information consists of system messages indicating the cause of copy errors during promote processing.

If you specify a blank ddname, SCLM routes the promote copy error information to the default output device, such as your terminal. Otherwise, before you call the PROMOTE service, you must allocate the ddname. The maximum parameter length is 8 characters.

Note: The remaining parameters are applicable only if the project has a language with rebuild on promote specified (an FLMLRBLD statement).

Y|N
Y indicates that build translator listings are to be copied to the dd_bldlist ddname only if errors occur. N indicates that all translator listings are to be copied to the dd_bldlist ddname. For FLMCMD, the default is Y. There is no default for FLMLNK. The maximum parameter length is 24 characters. This parameter only applies if the project definition requests automatic rebuild when a member is promoted into the 'to group'.

Y|N
Y indicates that a build report is to be produced and routed to the bldrept ddname. N indicates that a build report is not to be produced. For FLMCMD, the default is Y. There is no default for FLMLNK. The maximum parameter length is 24 characters. This parameter only applies if the project definition requests automatic rebuild when a member is promoted into the 'to group'.

prefix_userid
This is the data set name prefix to be used when locating and cataloging temporary data sets. If no value is specified for FLMCMD or a blank (' ') is specified for FLMLNK, it defaults to the user Id parameter. The maximum parameter length is 17 characters. This parameter only applies if the project definition requests automatic rebuild when a member is promoted into the 'to group'.

dd_bldmsgs
This is the ddname indicating the destination of the build messages. If you specify a blank ddname, SCLM routes the build messages to the default output device, such as your terminal. Otherwise, before you call the BUILD service, you must allocate the ddname. The following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. You cannot specify a blank ddname for FLMLNK. This parameter only applies if the project definition requests automatic rebuild when a member is promoted into the 'to group'. The maximum parameter length is 8 characters.

dd_bldrept
This is the ddname indicating the destination of the build report. If you specify a blank ddname, SCLM routes the build report to the default output device, such as your terminal. Otherwise, before you call the BUILD service, you must allocate the ddname. The following attributes should be used: RECFM=FBA, LRECL=80, BLKSIZE=3120. The maximum parameter length is 8 characters. This parameter only applies if the project definition requests automatic rebuild when a member is promoted into the 'to group'.

dd_bldlist
This is the ddname indicating the destination of the build listings. If you specify a blank ddname, SCLM does not generate the build listings. Otherwise, before you call the BUILD service, you must allocate the ddname. The following attributes should be used: DISP=MOD, RECFM=VBA, LRECL=137, BLKSIZE=3120. The maximum parameter length is 8 characters. This parameter only applies if the project definition requests automatic rebuild when a member is promoted into the 'to group'.

dd_bldexit
This is the ddname indicating the destination of the build user exit data. Specify this parameter only if your project definition defines a build user exit routine. Ask your project manager if your project is using a build user exit routine. If you specify a blank ddname, SCLM routes the build user exit data to NULLFILE. Otherwise, before you call the BUILD service you must allocate the ddname. The following attributes should be used: RECFM=FB, LRECL=160, BLKSIZE=3200. The maximum parameter length is 8 characters. This parameter only applies if the project definition requests automatic rebuild when a member is promoted into the 'to group'.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See “SCLM service messages” on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See “SCLM service return codes” on page 377 for more information.

Possible return codes are:

0
Normal completion. See the SCLM messages for more information.

4
Warning condition. See the SCLM messages for more information. The location of the messages file is determined by the dd_prommsgs parameter.

8
Error condition. See the SCLM messages for more information.

10
Promote completed successfully. Build was requested in the project definition, but the build failed. See the build messages file allocated to the dd_bldmsgs parameter for more information.

12
Severe error condition. SCLM does not produce messages because there was an error invoking the promote module.

16
Severe error condition. SCLM does not produce messages because SCLM cannot retrieve SCLM ID information.

Example of command invocation

```
FLMCMD PROMOTE,PROJ1,,USER1,ARCHDEF,FLM01CMD,,,U
```

This service command promotes the FLM01CMD member of the ARCHDEF type and all of its dependent members from the USER1 group to the next group in the hierarchy. The project name is PROJ1. The promote scope is normal (by default) and the promote mode is unconditional. SCLM sends messages, reports, and listings to the terminal.

Example of call invocation

This example shows general syntax. Call invocations are language-specific. See Chapter 20, “Sample programs using SCLM services,” on page 477 for specific examples.

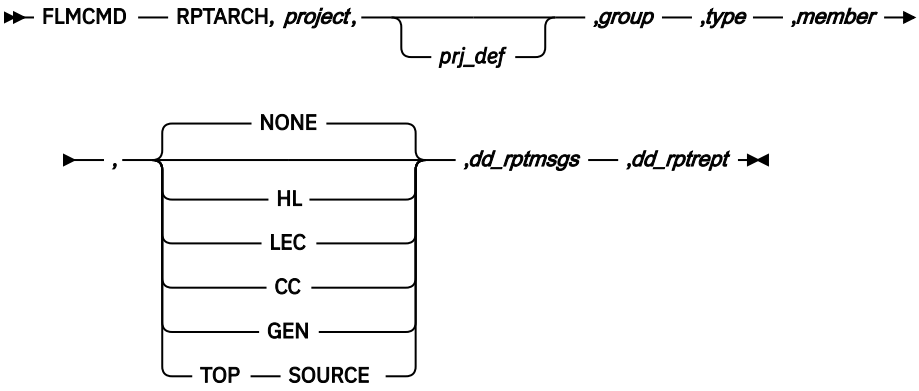
```
lastrc := FLMLNK('PROMOTE ',                    (* service        *)
                 sclm_id,                       (* SCLM ID        *)
                 'USER1   ',                    (* group          *)
                 'ARCHDEF ',                    (* type           *)
                 'FLM01CMD',                    (* member         *)
                 '        ',                    (* user ID        *)
                 'E                       ',   (* scope          *)
                 'R                       ',   (* mode           *)
                 'PROMMSGS',                    (* messages       *)
                 'PROMREPT',                    (* report         *)
                 'PROMEXIT',                    (* user exit data *)
                 'COPYDD  ');                   (* copy errors    *)
```

This service call performs a report-only promote on the FLM01CMD member of the ARCHDEF type in the USER1 group. The sclm_id parameter contains a valid SCLM ID returned from the INIT service and USERID identifies who is requesting the promote. The promote scope is extended. You must allocate the ddnames (PROMMSGS, PROMREPT, PROMEXIT, and COPYDD, respectively) before you call FLMLNK.

RPTARCH—Generate an SCLM Architecture Report

The RPTARCH service provides a list of all the components in a given application. The report generator examines the requested architecture and all of its references, and then constructs an indented report of the architecture. The report lists software components in each type referenced by the architecture to help you eliminate unnecessary code.

Command invocation format



Call invocation format

You cannot use call procedures to start this service.

ISPF interface panel

```
  Menu  SCLM  Utilities  Help
_______________________________________________________________________
                  SCLM FLMCMD RPTARCH Service - Entry Panel

SCLM Library:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1
   Type  . . . . SOURCE
   Member  . . . ________

Report Cutoff    _  1. HL
                    2. LEC
                    3. CC
                    4. Generic
                    5. Top Source
                    6. None

DD Names for output data sets:
   Error message data set ________  (Blank to write messages to the terminal)
   Report data set  . . . ________  (Blank to write report to the terminal)
Command ===> ___________________________________________________________
 F1=HELP      F2=          F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 202. RPTARCH Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters.

prj_def
The project definition name to be used for generating the architecture report. It defaults to project. The maximum parameter length is 8 characters.

group
The group the report is to be generated from. The maximum parameter length is 8 characters. If information is not found at the specified group, RPTARCH searches up the hierarchy to the next layer.

type
The type containing the member to be reported on. The maximum parameter length is 8 characters.

member
The member to be reported on. The maximum parameter length is 8 characters.

HL|LEC|CC|GEN|TOP SOURCE|NONE
Indicates the cutoff (determines depth) for the architecture report.

The architecture report contains the following if you specify:

HL
The HL architecture members in the application represented by the architecture member you specified with the member parameter.

LEC
The HL and LEC architecture members in the application represented by the architecture member you specified with the member parameter.

CC
The HL, LEC, and CC architecture members in the application represented by the architecture member you specified with the member parameter.

GEN
The HL and generic architecture members in the application represented by the architecture member you specified with the member parameter.

TOP SOURCE
The HL, LEC, CC, and generic architecture members and top source members in the application represented by the architecture member you specified with the member parameter.

NONE
The HL, LEC, CC, and generic architecture members in each of the types and all source members down to the lowest include group in the application represented by the architecture member you specified with the member parameter.

The maximum parameter length is 24 characters. The default value is NONE.

dd_rptmsgs
The ddname indicating the destination of the RPTARCH service messages. If you specify a blank ddname, SCLM routes the RPTARCH service messages to the default output device, such as your terminal. Otherwise, before you call the RPTARCH service, you must allocate the ddname. The following attributes should be used: `RECFM=F, LRECL=80, BLKSIZE=80`. The maximum parameter length is 8 characters.

dd_rptrept
The ddname indicating the destination of the architecture report. If you specify a blank ddname, SCLM routes the architecture report to the default output device, such as your terminal. Otherwise, before you call the RPTARCH service, you must allocate the ddname; the following attributes should be used: `RECFM=FBA, LRECL=80, BLKSIZE=3120`. The maximum parameter length is 8 characters.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See “SCLM service messages” on page 378 for more information.

Other return codes might be produced by the FLMCMD processor. See “SCLM service return codes” on page 377 for more information about these.

Possible return codes are:

0
Normal completion. See the SCLM messages for more information.

4
Warning condition. See the SCLM messages for more information.

8
Error condition. See the SCLM messages for more information.

Example of command invocation

```
FLMCMD RPTARCH,PROJ1,,USER1,SOURCE,FLM01MD1,NONE
```

This service command generates an architecture report for the FLM01MD1 member of the SOURCE type in the USER1 group. The project name is PROJ1. The report cutoff is NONE, and SCLM sends messages and the architecture report to your terminal.

SAVE—Lock, Parse, and Store a Member

The SAVE service locks and parses a member, and stores that member's statistical, dependency, and historical information all in one service call. The SAVE service calls the LOCK, PARSE, and STORE services.

Note: The SAVE service does not parse a member correctly if the member is packed. Make sure that the pack mode is off in the member's profile.

Before you start the SAVE service, the member must exist in the development library you specify. (The LOCK, SAVE, or STORE service can be complete for the member, but this is not necessary.) Upon completion of the SAVE service, the member has been locked and its access key has been set. (You must supply the correct access key for previously locked members.) A typical development scenario follows:

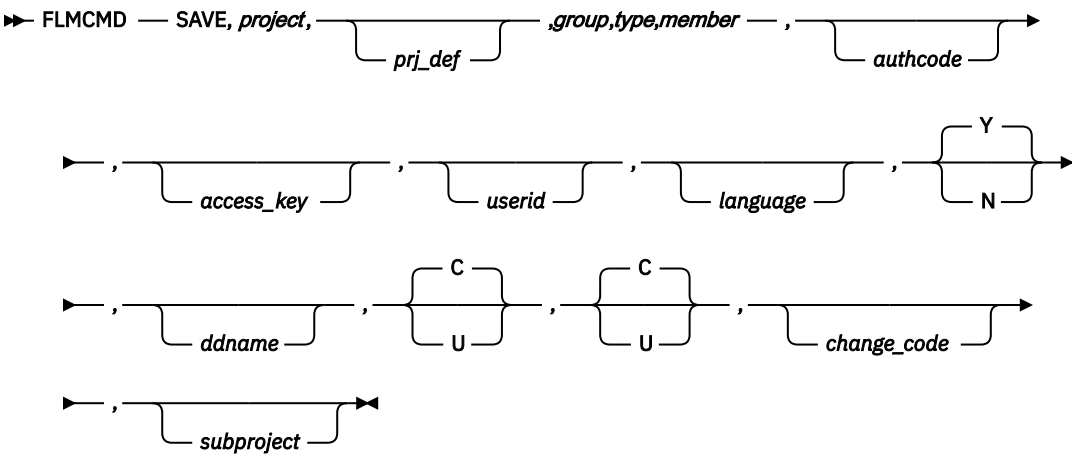
1. Update or create the member.
2. Start the SAVE service to parse the member and store the member's statistical, dependency, and historical information.

For more information about the LOCK, PARSE, and STORE services, see their service descriptions in this chapter.

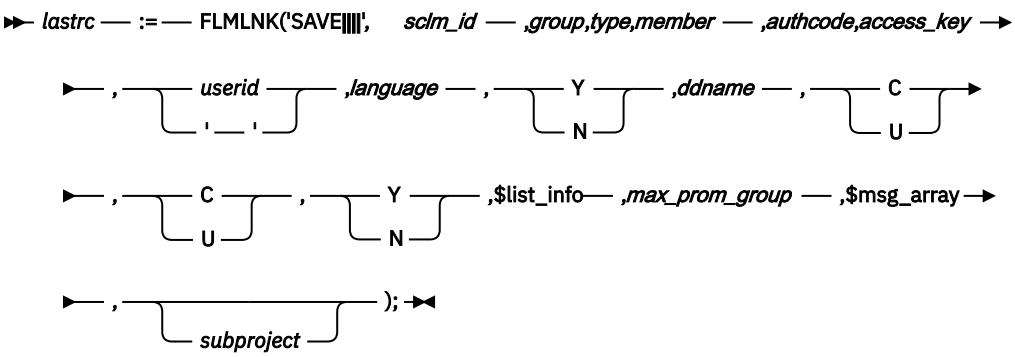
Note: Use of the SAVE service causes SCLM to delete all previously stored $list_info data from the member's dependency and historical information. Each invocation of the SAVE service creates a new set of statistical, dependency, and historical information for the member.

If you need pre-existing historical information, such as user entry data, do not invoke the SAVE service. Use the LOCK, PARSE, and STORE services instead.

Command invocation format



Call invocation format



ISPF interface panel

```
  Menu  SCLM  Utilities  Help
──────────────────────────────────────────────────────────────────────────────
                    SCLM FLMCMD SAVE Service - Entry Panel
                                                                      More:     +
SCLM Library:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1____
   Type  . . . . SOURCE__
   Member  . . . ________

Authorization code  . . ________  (If blank, the default auth code is used)
Access key  . . . . . . ________
User id . . . . . . . . ________  (If blank, your user id is used)
Change code . . . . . . ________
Language  . . . . . . . COB_____

                                     Compilation
Mode  . . _  1. Conditional          Unit Mode . . . _  1. Conditional
             2. Unconditional                           2. Unconditional

Command ===> ____________________________________________________________________
 F1=HELP      F2=          F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 203. SAVE Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name to be used for the lock, parse, and store. It defaults to the project parameter. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

sclm_id
An SCLM ID associated with a given project and project definition. The SCLM ID is generated by the INIT service. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

group
The group in which the lock, parse, and store are to occur. The specified group must be a development library. The maximum parameter length is 8 characters.

type
The type containing the member. The maximum parameter length is 8 characters.

member
The member to be locked and parsed, and whose accounting information is to be stored. The maximum parameter length is 8 characters.

authcode
The authorization code to be used for the lock. If you do not supply an authcode, SCLM uses default values as follows:

- The authorization code from the existing member if the member being locked exists in the hierarchy
- The default authorization code for the group if the member does not exist in the hierarchy.

The maximum parameter length is 8 characters.

access_key
The access key to be assigned to the member. The access key is required for any further manipulation of the member until you use the UNLOCK service to remove the access key. It defaults to blank. The maximum parameter length is 16 characters.

userid
User ID of the person requesting the SAVE service. It defaults to the current system user ID. The maximum parameter length is 8 characters.

language
The language of the member. The maximum parameter length is 8 characters. You must specify the language the first time you save a member; after that a language name is optional. If not specified, the language will default to the language already defined for the member. Specify a different language name if you wish to change the name of the language defined for the member. Parsers will be called based on the current value specified.

Y|N
Y indicates that SCLM is to copy parser listings to the ddname parameter only if parser errors occur. N indicates that SCLM is to copy all parser listings to the ddname. The maximum parameter length is 24 characters.

If the parser for the specified language does not produce a listing, specify Y. The language parsers supplied by SCLM do not produce a listing. If the parser for the specified language does produce a listing, you can specify either value. For more efficient performance, specify Y. Project-specific parsers can produce a listing. The default value for FLMCMD is Y. There is no default value for FLMLNK.

ddname
The ddname indicating the destination of the parser listings. If you specify a blank ddname, SCLM does not generate the parser listings. The maximum parameter length is 8 characters.

If the parser for the specified language does not produce a listing, specify a blank ddname. The language parsers supplied by SCLM do not produce a listing. If the parser for the specified language does produce a listing and you specified a ddname, allocate the ddname with the attributes required by the parser. Project-specific parsers can produce a listing.

C|U
Specify C to indicate that the member's statistical and dependency information is not to be saved in the event of a parser error; that is, the STORE service is not to be called if the PARSE service completes with a return code of 4. Specify U to indicate that the member's statistical and dependency information is to be saved even in the event of a parser error. The maximum parameter length is 24 characters. The default value for FLMCMD is C. There is no default value for FLMLNK.

C|U
Specify C to indicate that a compilation unit cannot be drawn down into a different member. Specify U to indicate that a compilation unit can be drawn down into a different member. The maximum parameter length is 24 characters. The default value for FLMCMD is C. There is no default value for FLMLNK.

change_code
A change_code to be added to the information obtained by parsing the member. If the member's accounting record lists the change_code, SCLM updates the date and time stamps for the existing change_code entry. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

subproject
The name of the subproject to be assigned to the member. This parameter must only be specified if the SCLM project has been defined with subprojects. For more information about subprojects, see Chapter 17, "SCLM security," on page 337.

The maximum parameter length is 8 characters.

Y|N
Y tells SCLM to verify change code records appearing in $list_info with the change code verification routine specified in the project definition. N tells SCLM not to verify change code records. The maximum parameter length is 24 characters.

This parameter is only valid for the FLMLNK call invocation. SCLM always verifies change_code records for the FLMCMD command format.

Specify N if your project definition does not specify a change_code verification routine. Ask your project manager if your project is using a change_code verification routine.

$list_info
An input or output parameter pointing to an array of records that contains change_code information. SCLM adds any change codes appearing in the array to the information it obtains by parsing the member. If you are not adding change_code information to the parser information, SCLM can pass a fullword zero buffer address. The array contains only change_code records.

SCLM deletes all information associated with the member (such as user entry data) previously stored through the STORE service with the $list_info parameter.

SCLM ignores the Date and Time Stamp fields on all change_code entries in the $list_info array. The SAVE service assigns the last change date and time from the member's accounting record to all change_codes it finds in the array. Note that SCLM does not update the array itself.

SCLM adds all change_code data listed in $list_info to the existing change_code data in the member's accounting record. If the member's accounting record already lists the change_code, SCLM updates the date and time stamps for the existing change_code entry.

This parameter is used for FLMLNK only. See "Pointer parameter descriptions" on page 368 for more details on $list_info.

max_prom_group
An output parameter indicating the highest group in the hierarchy to which the member can be promoted. Based on the authorization code you used for the lock, SCLM determines the highest group that you can promote this member to. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

$msg_array
An output parameter pointing to the message array. See "Pointer parameter descriptions" on page 368 for more information about $msg_array. This parameter is used for FLMLNK only.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See "SCLM service messages" on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM service return codes" on page 377 for more information.

Possible return codes are:

0
Normal completion.

4
Warning condition. The $msg_array parameter determines the location of this message array.

8
Error condition. The $msg_array parameter determines the location of this message array.

Example of command invocation

```
FLMCMD SAVE,PROJ1,,USER1,SOURCE,FLM01MD1,,XXX#05,,PASCAL,,,,,CC001234
```

This service command locks, parses, and stores the information for the member FLM01MD1 of the type SOURCE in the USER1 group. The project name is PROJ1 and the access key is XXX#05. Change code CC001234 is to be added to the information obtained by parsing the member with the PASCAL parser. All other parameters are default values.

Example of call invocation

This example shows general syntax. Call invocations are language-specific. See Chapter 20, “Sample programs using SCLM services,” on page 477 for specific examples.

```
$list_info := NIL;  (* Sets the buffer address to X'00000000' *)

lastrc := FLMLNK('SAVE    ',                  (* service            *)
                 sclm_id,                     (* SCLM ID            *)
                 'USER1   ',                  (* group              *)
                 'SOURCE  ',                  (* type               *)
                 'FLM01MD1',                  (* member             *)
                 'TESTAC  ',                  (* authorization code *)
                 'XXX#05          ',          (* access key         *)
                 '        ',                  (* user ID            *)
                 'PASCAL  ',                  (* language           *)
                 'Y                       ', (* listings           *)
                 'PARSEDD ',                  (* ddname of listings *)
                 'U                       ', (* statistical and dependency info *)
                 'C                       ', (* compilation unit   *)
                 'Y                       ', (* change codes       *)
                 $list_info,                  (* list information pointer *)
                 max_prom_group,              (* maximum promotable group *)
                 $msg_array);                 (* message array pointer    *)
```

This service call locks, parses, and stores the information for member FLM01MD1 of the SOURCE type in the USER1 group. The sclm_id parameter contains a valid SCLM ID returned from the INIT service. The authorization code to be used for the lock verification is TESTAC and the access key is XXX#05. The PASCAL parser parses the member.

SCLM copies parser listings to the PARSEDD ddname only if errors occur. If a parser error does occur, the STORE still completes, SCLM does not draw down compilation units into a different member, and the service verifies all change codes found in $list_info. SCLM returns all messages produced in the $msg_array parameter. You must allocate the PARSEDD ddname before you call FLMLNK.

SCLMINFO—Return Project Information

The SCLMINFO service returns information about an SCLM project. This information is retrieved by reading the SCLM project definition load library.

The project information is returned in the following ISPF variables:

ZSCIPROJ
Project specified by the user

ZSCIPDEF
Alternate specified by the user

ZSCIGRP
Group information for the project

ZSCITYPE
Type information for the project

ZSCILANG
Language information for the project

ZSCISVER
SCLM version ID for the project

ZSCITMST
Timestamp (date and time when the project was generated)

ZSCIACTF
Accounting file names for the project

ZSCIAUT
Authorization code information for the project

ZSCINPAT
Data set name patterns information for the project

ZSCIGRP contains a list of all the groups specified for this project. The following information is returned for each group:

```
LV=001 GR=RELEASE
```

You must parse the variable to retrieve the information for each group.

ZSCITYP contains a list of all the types specified for this project. The following information is returned for each type:

```
TY=ARCHDEF XT=nnnnnn
```

You must parse the variable to retrieve the information for each type. The XT= information will only be displayed if you have specified the "EXTEND=" on the FLMTYPE macro when defining the project.

ZSCILANG contains a list of all the languages specified for this project. Here is an example of the information that is returned for each language:

```
LA=HLAS     LD=HLASM TRANSLATOR
```

You must parse the variable to retrieve the name and description of each language.

ZSCIACTF contains the concatenation of the names of all accounting files specified for this project. Each name is padded to 44 bytes.

ZSCIAUT contains a list of all the authorization codes specified at each group for this project. Here is an example of the information that is returned for each group:

```
GR=TEST     AC=P        D
```

You must parse the variable to retrieve the authorization codes for each group.

With the PTFs for APAR OA21104 applied, ZSCINPAT contains a list of all the data set patterns used for each group for this project. Even if flexible data set naming has not been used on the FLMCNTRL, or FLMALTC macros, this variable will be returned with the default pattern of @@FLMPRJ.@@FLMGRP.@@FLMTYP specified for each group. Here is an example of the information that is returned for each group:

```
GR=PROD     NP=@@FLMPRJ.V2.@@FLMGRP.@@FLMTYP
```

You must parse the variable to retrieve the data set pattern for each group.

Command invocation format

Call invocation format

ISPF interface panel

```
  Menu  SCLM  Utilities  Help
_______________________________________________________________________
                  SCLM FLMCMD SCLMINFO Service - Entry Panel

SCLM Hierarchy:
   Project . . . SCLMTEST
   Alternate . . ________

DD Name for output data set:
   Error message data set ________  (Blank to write messages to the terminal)









Command ===> _____________________________________________________________
 F1=HELP      F2=           F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 204. SCLMINFO Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name. It defaults to the project name. The maximum parameter length is 8 characters.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See “SCLM service messages” on page 378 for more information.

Other return codes might be produced by the FLMCMD processor. See “SCLM service return codes” on page 377 for more information.

Possible return codes are:

0
Normal completion. The information was retrieved successfully.

12
Error completion. Refer to the messages for more information.

START—Generate an Application ID for a Services Session

The START service initializes an SCLM services session. It generates an application ID that identifies the services session. You can use the application ID to call the INIT service to initialize an SCLM ID. Each START service invocation needs a matching END service invocation.

Command invocation format

You cannot use command procedures to call this service.

Call invocation format

Parameters

appl_id
The generated application ID identifying the SCLM services session. Each time you invoke the START service, SCLM generates a unique application ID in this output parameter. The maximum parameter length is 8 characters.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See “SCLM service messages” on page 378 for more information.

Other return codes might be produced by the FLMLNK processor. See “SCLM service return codes” on page 377 for more information.

Possible return codes are:

0
Normal completion.

12
Severe error condition. The maximum application ID limit was exceeded.

16
Severe error condition. An invalid version of the SCLM table was loaded.

20
Severe error condition. An invalid version of the multicultural support table was loaded.

24
Severe error condition. SCLM is unable to load the SCLM table.

28
Severe error condition. SCLM is unable to load the multicultural support table or the SCLM I/O load module.

32
Severe error condition. An invalid parameter list was passed to the requested service.

34
Severe error condition. An invalid service was requested.

36
Severe error condition. The version of the FLMLNK subroutine does not match the version of the SCLM services module.

Example of call invocation

This example shows a general syntax. Call invocations are language-specific. See Chapter 20, “Sample programs using SCLM services,” on page 477 for specific examples.

```
lastrc := FLMLNK('START   ',   (* service        *)
                  appl_id);     (* application ID *)
```

This service call initializes an SCLM services session.

STORE—Store Member Information in an Accounting Record

The STORE service saves a member's statistical, dependency, and historical information in an accounting record in the project database. SCLM usually obtains statistical and dependency information by parsing the member, and it is a required input to the STORE service. SCLM retains the historical information in the project database and automatically generates it for the member.

Before you call the STORE service, you must lock the member using the LOCK service, and the member must exist in the development library you specify. After the STORE service ends, the member remains locked and the access key also remains unchanged. A typical development scenario follows:

1. Use the LOCK service to lock the member. The member may or may not yet exist.
2. Update or create the member.
3. Parse the member using the PARSE service.
4. Save the member's statistical, dependency, and historical information using the STORE service.

The STORE service removes duplicate dependency information for each member. For example, if a member is referenced as an include ten times, the STORE service records the reference only once in the accounting information.

When the STORE service receives dependency information, it replaces the existing dependency information rather than appending to it.

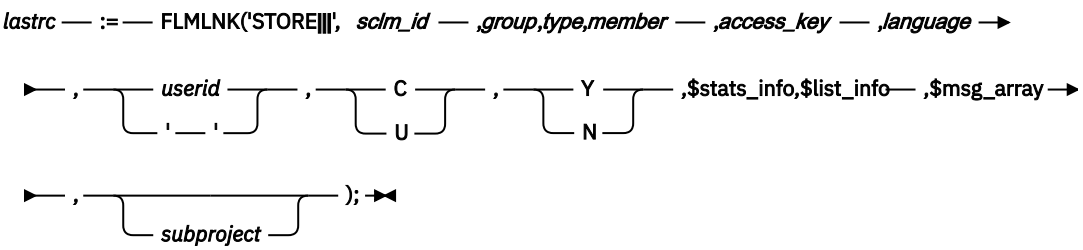
Change code information can relate problem report (PR) numbers, change request (CR) numbers, and other information to individual source members. The STORE service can validate change codes you input to the STORE service before it enters them into the accounting records and saves the member.

Like dependency information, all existing user data entries are replaced with the new user data the STORE service receives. User data entries are stored directly into the accounting information for the member. Duplicate entries passed to the STORE service are preserved in the accounting information.

Command invocation format

You cannot use command procedures to call this service.

Call invocation format



Parameters

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters.

group
The group in which the store is to occur. The specified group must be a development library. The maximum parameter length is 8 characters.

type
The type containing the member whose information is to be stored. The maximum parameter length is 8 characters.

member
The member whose information is to be stored. The maximum parameter length is 8 characters.

access_key
The access key assigned to the member with the LOCK service. If you supply an incorrect access key, the service fails. The maximum parameter length is 16 characters.

language
The language of the member. If you used the PARSE service to parse the member, this language should be the same as the one specified as input to the PARSE service. The maximum parameter length is 8 characters. However, if the language is different, you can generate your own $stats_info and write an accounting record. You can also use the statistics retrieved from the PARSE service, and it will create a new accounting record with the updated information.

userid
The user ID of the person requesting the STORE service. It defaults to the current system user ID. The maximum parameter length is 8 characters.

C|U
C indicates conditional; SCLM does not draw down a compilation unit into a different member. U indicates unconditional; SCLM can draw down a compilation unit into a different member. The maximum parameter length is 24 characters.

Y|N
Y tells SCLM to verify change code records appearing in $list_info with the change code verification routine specified in the project definition. N tells SCLM not to verify change code records. The maximum parameter length is 24 characters.

Ask your project manager if your project is using a change code verification routine. If it is not, specify N.

$stats_info
A pointer to a record containing the member's statistical information. You must have a valid buffer address.

Note: If you used the PARSE service to generate the record, you must copy the buffer to the calling program's local storage before calling the STORE service. Failure to copy the buffer to local storage causes unpredictable results.

See “Pointer parameter descriptions” on page 368 for more details on the $stats_info parameter and copying the record contents.

$list_info
A pointer to an array of records that contains the member's include, change code and user entry information. If the member has none of this information, you can pass a fullword zero buffer address.

All include and user entry information data listed in $list_info replaces existing accounting record data for the member. If you want to maintain existing information (such as user entry history) for the member, it must appear in the $list_info parameter.

SCLM ignores the Date and Time Stamp fields on all change code entries in the $list_info array. The STORE service assigns the current system date and time to all change codes it finds in the array. Note that SCLM does not update the array itself.

SCLM adds all change code data listed in $list_info to the existing change code information in the member's accounting record. If the change code is already listed in the member's accounting record, SCLM updates the date and time stamps for the existing change code entry.

The order of the include entries in $list_info determine the order in which the build function processes the member's dependencies.

Note that SCLM does not permit duplicate record entries in the $list_info array. If it encounters duplicate records, it flags an error.

Note: If you used the PARSE service to generate the array, you must copy the buffer to the calling program's local storage before you call the STORE service. Failure to copy the buffer to local storage causes unpredictable results. See “Pointer parameter descriptions” on page 368 for more information about the $list_info parameter and copying the array contents.

$msg_array
An output parameter pointing to the message array. See “Pointer parameter descriptions” on page 368 for more information about $msg_array.

subproject
The name of the subproject to be assigned to the member. This parameter must only be specified if the SCLM project has been defined with subprojects. For more information about subprojects, see Chapter 17, “SCLM security,” on page 337.

The maximum parameter length is 8 characters.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See “SCLM service messages” on page 378 for more information.

Other return codes might be produced by the FLMLNK processor. See “SCLM service return codes” on page 377 for more information.

Possible return codes are:

0
Normal completion.

4
Warning condition. The $msg_array parameter determines the location of this message array.

8
Error condition. The $msg_array parameter contains the error message associated with this condition.

Example of call invocation

This example shows general syntax. Call invocations are language-specific. See Chapter 20, “Sample programs using SCLM services,” on page 477 for specific examples.

```
lastrc := FLMLNK('STORE   ',              (* service          *)
             sclm_id,                     (* SCLM ID          *)
```

```
              'USER1   ',                  (* group            *)
              'SOURCE  ',                  (* type             *)
              'FLM01MD2',                  (* member           *)
              'XXX#04          ',          (* access key       *)
              'PASCAL  ',                  (* language         *)
              '        ',                  (* user ID          *)
              'C                       ',  (* compilation unit *)
              'Y                       ',  (* change codes     *)
              $stats_info,                 (* statistical information pointer *)
              $list_info,                  (* listing information pointer     *)
              $msg_array);                 (* message array pointer           *)
```

This service call stores the statistical and dependency information (obtained from $stats_info and $list_info) in the accounting record for member FLM01MD2 in the project database. The sclm_id parameter contains a valid SCLM ID returned from the INIT service.

The member FLM01MD2 must exist in the SOURCE type in the USER1 group and must have previously been locked with an access key of XXX#04. The member is identified as a PASCAL member.

SCLM does not draw down compilation units into a different member and it verifies all change codes found in $list_info. SCLM returns all messages in the $msg_array array.

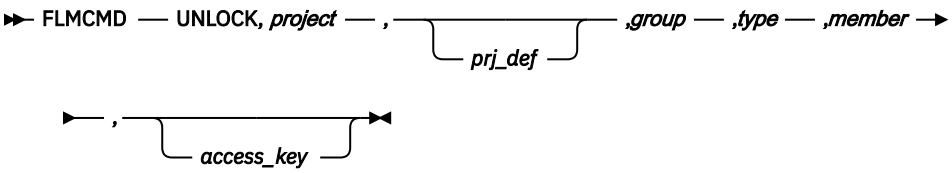
UNLOCK—Unlock a Member in a Development Library

The UNLOCK service makes a locked member available for updates by another user. If an access key was assigned to the member when it was locked, the UNLOCK service resets the access key to blank.

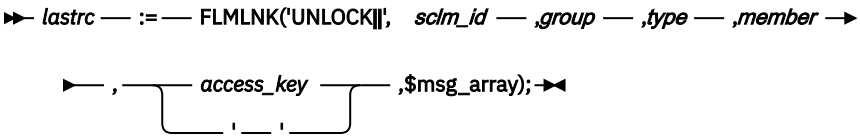
If SAVE or STORE completes successfully for a member and that member has an access key, you can reset the access key by calling the UNLOCK service.

Before you can promote a member, you must call the UNLOCK service to remove its access key. The PROMOTE service does not promote any member that has an access key. For more information about the LOCK service and access keys, see “LOCK—Lock a Member or Assign an Access Key” on page 430.

Command invocation format



Call invocation format



ISPF interface panel

```
  Menu  SCLM  Utilities  Help
_______________________________________________________________________________
                   SCLM FLMCMD UNLOCK Service - Entry Panel

SCLM Library:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1
   Type  . . . . SOURCE
   Member  . . . ________         (Blank or pattern for member selection list)

Access key  . . . . . .  ________________









Command ===> ___________________________________________________________________
 F1=HELP      F2=          F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 205. UNLOCK Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name to be used for the unlock. It defaults to project. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

group
The group in which the member is to be unlocked. The specified group must be a development library. The maximum parameter length is 8 characters.

type
The type containing the member to be unlocked. The maximum parameter length is 8 characters.

member
The member to be unlocked. The maximum parameter length is 8 characters.

access_key
The access key assigned (with the LOCK or SAVE service) to the member. If you supply an incorrect access key, the unlock fails. The maximum parameter length is 16 characters.

For the FLMCMD format, the default is blank. For the FLMLNK format, you MUST specify an access key parameter. If you do not want to specify an access key on the FLMLNK, you must pass blanks as the parameter value.

$msg_array
An output parameter pointing to the message array. See “Pointer parameter descriptions” on page 368 for more information about $msg_array. This parameter is used for FLMLNK only.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See "SCLM service messages" on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM service return codes" on page 377 for more information.

Possible return codes are:

0
Normal completion.

4
Warning condition. The $msg_array parameter determines the location of this message array.

8
Error condition. The $msg_array parameter contains the error message associated with this condition.

Example of command invocation

```
FLMCMD UNLOCK,PROJ1,,USER1,SOURCE,FLM01MD1,XXX#05
```

This service command unlocks the FLM01MD1 member of the SOURCE type in the USER1 group. The project name is PROJ1. The access key value for the member is XXX#05.

Example of call invocation

This example shows general syntax. Call invocations are language-specific. See Chapter 20, "Sample programs using SCLM services," on page 477 for language-specific examples.

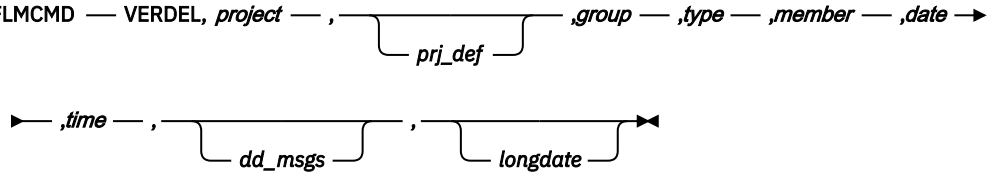
```
lastrc := FLMLNK('UNLOCK  ',             (* service               *)
                 sclm_id,                (* SCLM ID               *)
                 'USER1   ',             (* group                 *)
                 'SOURCE  ',             (* type                  *)
                 'FLM01MD1',             (* member                *)
                 'XXX#05        ',       (* access key            *)
                 $msg_array);            (* message array pointer *)
```

This service call unlocks the FLM01MD1 member of the SOURCE type in the USER1 group. The sclm_id parameter contains a valid SCLM ID returned from the INIT service. The access key value for the member is XXX#05. SCLM returns all messages in the $msg_array parameter.

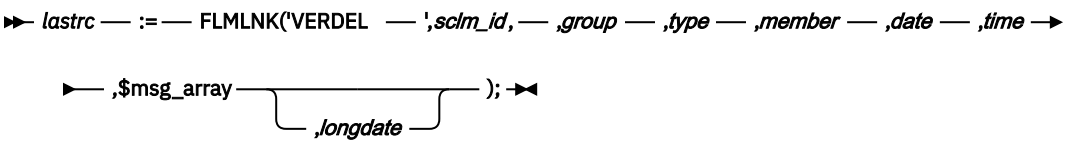
VERDEL—Delete Version and Audit Information

The VERDEL service deletes the information about a versioned or audited member from SCLM. The information is deleted from the auditing data set defined in the project definition and from the versioning PDS associated with the audit record, if it exists. The partitioned data set used for storing the versions is also updated for deletion of the version. The date and time specified to the service must exactly match the date and time of the audit and version information to delete. Use the VERINFO service to obtain the dates and times of audit and version information.

Command invocation format



Call invocation format



ISPF interface panel

```
  Menu  SCLM  Utilities  Help
_______________________________________________________________________
                    SCLM FLMCMD VERDEL Service - Entry Panel

SCLM Library:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1____
   Type  . . . . SOURCE__
   Member  . . . ________

Date of audit or version  . . . . .  __________ (In ZDATEF format)
Time of audit or version  . . . . .  ___________ (In HH:MM:SS.hh format)

DD Names for output data sets:
   Error message data set ________  (Blank to write messages to the terminal)




Command ===> ___________________________________________________________________
 F1=HELP      F2=          F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 206. VERDEL Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name. It defaults to the project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

group
The group associated with the version or audit record. The maximum parameter length is 8 characters.

type
The type associated with the version or audit record. The maximum parameter length is 8 characters.

member
The member that has the version or audit record. The maximum parameter length is 8 characters.

date
The date of the version or audit record. The date must be specified in the format given in the ZDATEF ISPF variable. Either the date or the longdate parameter is required. If both are given, the date parameter is used. The length of this parameter is 8 characters.

time
The time of the version or audit record. The format for the time is HH:MM:SS.hh or HH:MM:SS,hh where HH is the hour from a 24 hour clock, MM is the minute, SS is the seconds and hh is the hundredths of seconds. The length of this parameter is 11 characters.

dd_msgs
The ddname indicating the destination of the messages generated by the VERDEL service. If you specify a blank ddname, SCLM routes the VERDEL messages to the default output device, such as your terminal. Otherwise, before you call the VERDEL service, you must allocate the ddname. The following attributes should be used: `RECFM=F, LRECL=80, BLKSIZE=80.` The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

$msg_array
An output parameter pointing to the message array. See "Pointer parameter descriptions" on page 368 for more information about $msg_array. This parameter is used for FLMLNK only.

longdate
The date of the version or audit record. The longdate, with a 4-character year, must be specified in the national language format. Either the date or the longdate parameter is required. If both are given, the date parameter is used. The length of this parameter is 10 characters.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See "SCLM service messages" on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM service return codes" on page 377 for more information.

Possible return codes are:

0
Normal completion. The audit and version information were deleted.

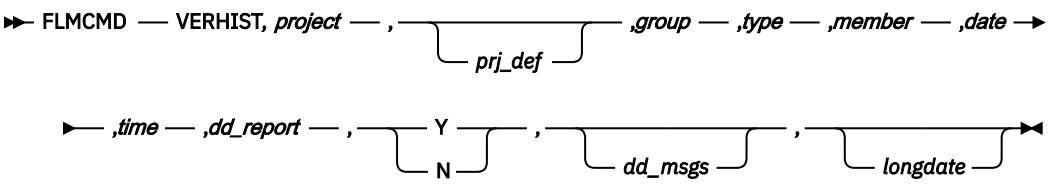
8
Error completion. No audit and version information was deleted. No audit record was found that matches the specified criteria.

12
Error completion. Refer to the messages for more information.

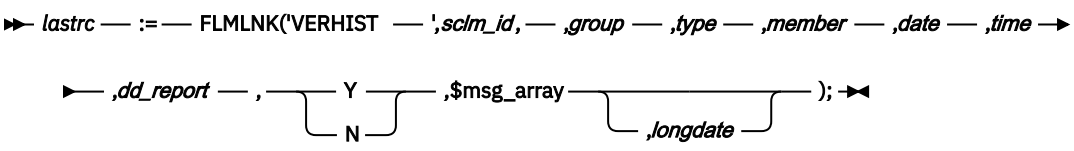
VERHIST—Retrieve Versioned Member Information

The VERHIST service retrieves a report of the detailed changes for a versioned member into a sequential data set. The VERINFO service can be used to identify the available versions, and then this service can be invoked to extract the member tagged with the changes. The output is the same as would be provided by the Compare selection in the Audit and Version Utility (option 3.8).

Command invocation format



Call invocation format



Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name. It defaults to the project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

group
The group associated with the version or audit record. The maximum parameter length is 8 characters.

type
The type associated with the version or audit record. The maximum parameter length is 8 characters.

member
The member that has the version or audit record. The maximum parameter length is 8 characters.

date
The date of the version or audit record. If omitted or specified as blanks the longdate is used. The date must be specified in the format given in the ZDATEF ISPF variable. The length of this parameter is 8 characters.

time
The time of the version or audit record. The format for the time is HH:MM:SS.hh or HH:MM:SS,hh where HH is the hour from a 24 hour clock, MM is the minute, SS is the seconds, and hh is the hundredths of seconds. The length of this parameter is 11 characters.

dd_report
The name of a predefined DD that references a data set where the version report for the specified member is to be written. The data set can be fixed or variable format with a logical record length at least 133. The specified DD will be released before the service returns. The maximum parameter length is 8 characters.

Y|N
Specifies if the VIEW service is to be invoked to review the resultant report data set or not.

dd_msgs
The ddname indicating the destination of the messages generated by the VERHIST service. If you specify a blank ddname, SCLM routes the VERHIST messages to the default output device, such as your terminal. Otherwise, before you call the VERHIST service, you must allocate the ddname. The

following attributes should be used: `RECFM=F, LRECL=80, BLKSIZE=80.` The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

$msg_array
An output parameter pointing to the message array. See "Pointer parameter descriptions" on page 368 for more information about $msg_array. This parameter is used for FLMLNK only.

longdate
The date of the version or audit record. If omitted or specified as blanks the date parameter is used. The longdate must be specified in the national language format with a 4-character year. The length of this parameter is 10 characters.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See "SCLM service messages" on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM service return codes" on page 377 for more information.

Possible return codes are:

0
Normal completion. An audit record exactly matching the specified criteria was found and the version report was stored successfully.

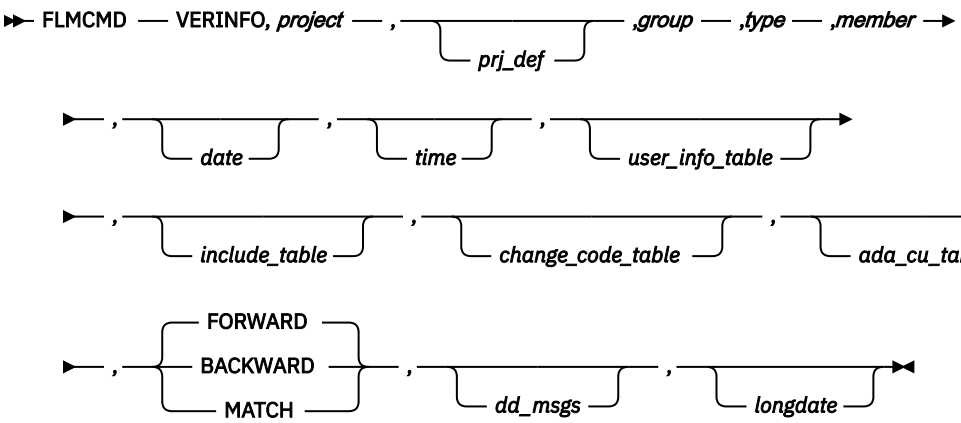
8
Error completion. No audit record was found for the specified member.

12
Error completion. Refer to the messages for more information.

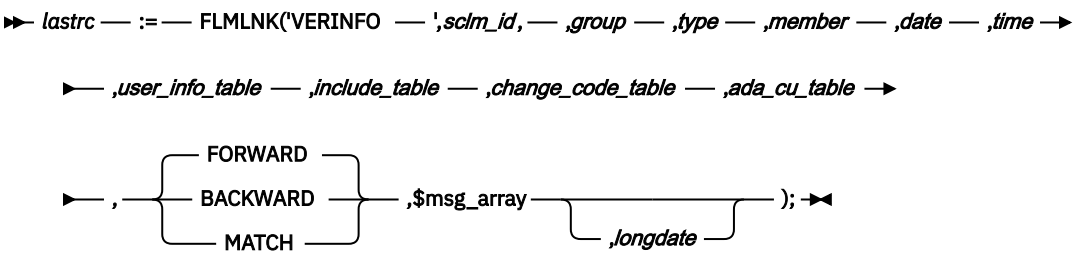
VERINFO—Retrieve Version and Audit Information

The VERINFO service retrieves the information about a versioned or audited member into ISPF variables and tables. The service can search a group for the next or previous matching audit record, or retrieve a specific audit record. See "ISPF variables" on page 372 for a list of the variables updated by this service.

Command invocation format



Call invocation format



ISPF interface panel

```
  Menu  SCLM  Utilities  Help
──────────────────────────────────────────────────────────────────────
                 SCLM FLMCMD VERINFO Service - Entry Panel
                                                                More:     +
SCLM Library:
   Project . . . SCLMTEST
   Alternate . . ________
   Group . . . . DEV1____
   Type  . . . . SOURCE__
   Member  . . . ________

Date of audit or version  . . . . .  __________ (In ZDATEF format)
Time of audit or version  . . . . .  ___________ (In HH:MM:SS.hh format)
Search type . . _  1. Search
                   2. Forward
                   3. Match


Names of open tables for service output:
   User Info . . . . . . . ________
   Includes  . . . . . . . ________
Command ===> ____________________________________________________________
 F1=HELP      F2=          F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
 F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 207. VERINFO Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name. It defaults to the project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

group
The group associated with the version or audit record. The maximum parameter length is 8 characters.

type
The type associated with the version or audit record. The maximum parameter length is 8 characters.

member
The member that has the version or audit record. The maximum parameter length is 8 characters.

date
The date of the version or audit record. If omitted or specified as blanks the longdate is used. If both the date and longdate are omitted or specified as blanks, the date will default to 00/00/00. The date must be specified in the format given in the ZDATEF ISPF variable. The length of this parameter is 8 characters.

time
The time of the version or audit record. If omitted or specified as blanks the time will default to 00:00:00.00. The format for the time is HH:MM:SS.hh or HH:MM:SS,hh where HH is the hour from a 24 hour clock, MM is the minute, SS is the seconds and hh is the hundredths of seconds. The length of this parameter is 11 characters.

user_info_table
The name of the ISPF table to contain the user entries from the audit record. The table must be open before calling the VERINFO service. A TBADD will be performed for each user entry in the audit record. The maximum parameter length is 8 characters. The following ISPF variables must be used in the table definition in order to have their value stored in the table:

- ZSUNUM - the user entry number
- ZSUENTRY - the user entry data

include_table
The name of the ISPF table to contain the includes from the audit record. The table must be open before calling the VERINFO service. A TBADD will be performed for each include in the audit record. The maximum parameter length is 8 characters. The following ISPF variables must be used in the table definition in order to have their value stored in the table:

- ZSIMBR - the include member name
- ZSIISET - the include set name

change_code_table
The name of the ISPF table to contain the change codes from the audit record. The table must be open before calling the VERINFO service. A TBADD will be performed for each change code in the audit record. The maximum parameter length is 8 characters. The following ISPF variables must be used in the table definition in order to have their value stored in the table:

- ZSCCODE - the change code
- ZSCDATE - the change code date in 2-character date format
- ZSCDAT4 - the change code date in 4-character date format
- ZSCTIME - the change code time

ada_cu_table
The name of the ISPF table to contain the ADA compilation units from the audit record. The table must be open before calling the VERINFO service. A TBADD will be performed for each ADA compilation unit in the audit record. The maximum parameter length is 8 characters. The following ISPF variables must be used in the table definition in order to have their value stored in the table:

- ZSDNAME- the ADA compilation unit name
- ZSDTYPE - the ADA compilation unit type

FORWARD|BACKWARD|MATCH

FORWARD indicates that if the type name, member name, date, or time do not exactly match an audit record, the information from the next audit record for the group is to be returned. This is the default.

BACKWARD indicates that if the type name, member name, date, or time do not exactly match an audit record, the information from the previous audit record for the group is to be returned.

MATCH indicates that the type name, member name, date, and time must exactly match the type name, member name, date, and time in an audit record.

To retrieve all of the audit records within a group use FORWARD and start with the type name and member name set to blanks and the date and time set to all zeros. If an audit record is found increment the last digit of the time by one before calling the VERINFO service again. Repeat this process until the service indicates that no record was found.

The maximum parameter length is 8 characters.

dd_msgs
The ddname indicating the destination of the messages generated by the VERINFO service. If you specify a blank ddname, SCLM routes the VERINFO messages to the default output device, such as your terminal. Otherwise, before you call the VERINFO service, you must allocate the ddname. The following attributes should be used: `RECFM=F, LRECL=80, BLKSIZE=80.` The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

$msg_array
An output parameter pointing to the message array. See "Pointer parameter descriptions" on page 368 for more information about $msg_array. This parameter is used for FLMLNK only.

longdate
The date of the version or audit record. If omitted or specified as blanks the date parameter is used. If both the date and longdate are omitted or specified as blanks, the date will default to 00/00/00. The longdate must be specified in the national language format with a 4-character year. The length of this parameter is 10 characters.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See "SCLM service messages" on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM service return codes" on page 377 for more information.

Possible return codes are:

0
Normal completion. An audit record exactly matching the specified criteria was found and the information was stored successfully.

8
Error completion. No audit record was found for the specified member.

- If FORWARD was specified, then there are no audit records for the group which match or follow the specified type, member, date, and time.
- If BACKWARD was specified, then there are no audit records for the group which match or precede the specified type, member, date, and time.
- If MATCH was specified, then there is not an audit record with the specified group, type, and member name.

12
Error completion. Refer to the messages for more information.

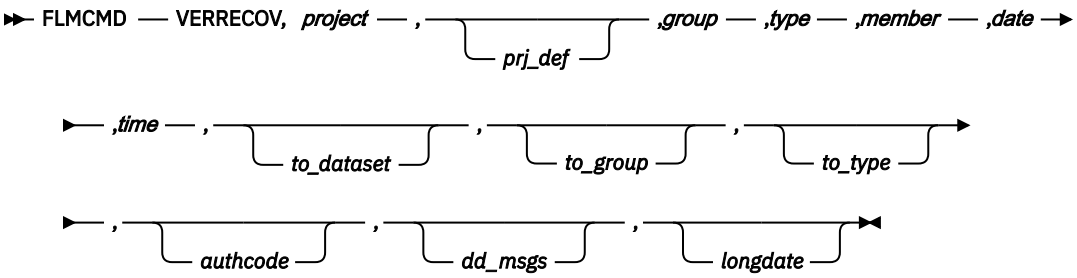
VERRECOV—Recover a Version

The VERRECOV service recovers a version of a member from the version data set. For retrieval of a member into the hierarchy the information is recovered from the auditing data set defined in the project definition for the group specified to the service. The date and time specified to the service must exactly match the date and time of the audit record with version information to recover. Use the VERINFO service to obtain the dates and times of audit and version information. The VERINFO service sets variable ZSVMBR, which tells the name of the version member. If ZSVMBR is blank after a VERINFO call, then

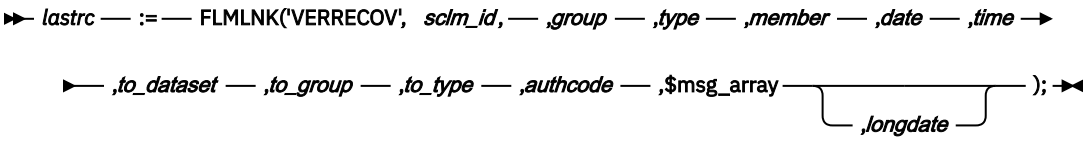
there is an audit record but no version of the member with this date and time. If ZSVMBR is not blank, then there is a version to recover.

The recovery can be done to a data set outside of SCLM control by specifying the to_dataset name parameter. To recover into the SCLM project specify the to_group, to_type and optionally the authcode. When recovering into SCLM, a lock is first done to lock the member at the specified group and type. If the lock fails no recovery will be performed. Either the to_dataset must be specified or the to_group and to_type must be specified to indicate the location of the recovered member.

Command invocation format



Call invocation format



ISPF interface panel

```
  Menu  SCLM  Utilities  Help
──────────────────────────────────────────────────────────────────────────────
                  SCLM FLMCMD VERRECOV Service - Entry Panel
                                                                More:     +
 SCLM Library:
    Project . . . SCLMTEST
    Alternate . . ________
    Group . . . . DEV1____
    Type  . . . . SOURCE__
    Member  . . . ________

 Date of audit or version  . . . . .  __________ (In ZDATEF format)
 Time of audit or version  . . . . .  ___________ (In HH:MM:SS.hh format)

 Non-SCLM controlled retrieve output data set:
    Sequential Data Set
Name . . . ____________________________________________

 SCLM controlled retrieve output library:
    Group . . . . ________
    Type  . . . . ________

 Command ===> ____________________________________________________________
  F1=HELP      F2=           F3=END       F4=DATASETS  F5=FIND      F6=CHANGE
  F9=SWAP     F10=LEFT     F11=RIGHT    F12=SUBMIT
```

Figure 208. VERRECOV Service panel

Parameters

project
The project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

prj_def
The project definition name. It defaults to the project name. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

sclm_id
An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters. This parameter is used for FLMLNK only.

group
The group associated with the version or audit record. The maximum parameter length is 8 characters.

type
The type associated with the version or audit record. The maximum parameter length is 8 characters.

member
The member that has the version or audit record. The maximum parameter length is 8 characters.

date
The date of the version or audit record. The date must be specified in the format given in the ZDATEF ISPF variable. The length of this parameter is 8 characters.

time
The time of the version or audit record. The format for the time is HH:MM:SS.hh or HH:MM:SS,hh where HH is the hour from a 24 hour clock, MM is the minute, SS is the seconds and hh is the hundredths of seconds. The length of this parameter is 11 characters.

to_dataset
The name of the data set to hold the recovered member. The data set must be a PDS without the member name specified. The data set name must be fully qualified without quotes. If the data set is a PDS the member name will be the name of the member being recovered. If this parameter is specified then the to_group and to_type parameters must not be specified. The maximum parameter length is 44 characters.

to_group
The name of the group to hold the recovered member. The group must be a development group (lowest level of the hierarchy). This parameter requires that the to_type also be specified. If this parameter is specified then the to_dataset must not be specified. The maximum parameter length is 8 characters.

to_type
The name of the type to hold the recovered member. This parameter requires that the to_group also be specified. If this parameter is specified then the to_dataset must not be specified. The maximum parameter length is 8 characters.

authcode
The authorization code to be used for locking the member in the hierarchy. The authorization code must be valid for the group specified in the to_group parameter. If this parameter is not specified and to_group is specified then SCLM will attempt to lock the member with the authorization code that is in the audit record. This parameter requires that the to_group and to_type also be specified. If this parameter is specified then the to_dataset must not be specified. The maximum parameter length is 8 characters.

dd_msgs
The ddname indicating the destination of the messages generated by the VERRECOV service. If you specify a blank ddname, SCLM routes the VERRECOV messages to the default output device, such as your terminal. Otherwise, before you call the VERRECOV service, you must allocate the ddname. The following attributes should be used: `RECFM=F, LRECL=80, BLKSIZE=80.` The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

$msg_array
An output parameter pointing to the message array. See "Pointer parameter descriptions" on page 368 for more information about $msg_array. This parameter is used for FLMLNK only.

longdate
The date of the version or audit record. If omitted or specified as blanks, the date parameter is used. If both the date and longdate are omitted or specified as blanks, the date will default to 00/00/00. The longdate, with a 4-character year, must be specified in the national language format. The length of this parameter is 10 characters.

Return codes

Additional special services messages are written to the FLMMSGS ddname. See "SCLM service messages" on page 378 for more information.

Other return codes might be produced by the FLMCMD or the FLMLNK processor. See "SCLM service return codes" on page 377 for more information.

Possible return codes are:

0
Normal completion. The audit and version information were recovered.

8
Error completion. No audit and version information was recovered. No audit record was found that matches the specified criteria.

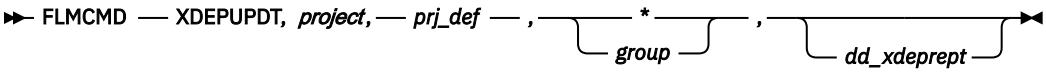
10
Error completion. No audit and version information was recovered. The member could not be locked with the specified authorization code.

12
Error completion. Refer to the messages for more information.

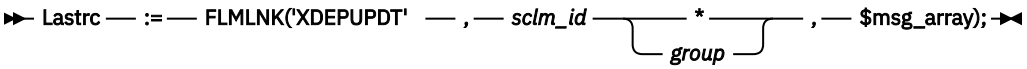
XDEPUPDT—Update Cross-dependency Information

Command invocation format

The XDEPUPDT service populates the cross dependency (XDEP) database. You can use the XDEPUPDT service to create the XDEP database initially, and then let SCLM automatically update the database as the application changes by using the XDEPDYN=Y parameter of the FLMCNTRL macro. If you set XDEPDYN=N, you need to run the XDEPUPDT service whenever you need to refresh the database.



Call invocation format



Parameters

project
The project name.

prj_def
The project definition name.

group
The group pattern to process. This should normally be specified as '*' to process ALL groups.

dd_xdeprept
The ddname indicating the destination of the messages generated by the XDEPUPDT service. If you specify a blank ddname, SCLM routes the XDEPUPDT messages to the default output device, such as your terminal. Otherwise, before you call the XDEPUPDT service, you must allocate the ddname. The following attributes should be used: RECFM= F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

$msg_array
An output parameter pointing to the message array. See the section "Pointer Parameter Descriptions" in Chapter 16,"Invoking the SCLM services" for more information about $msg_array. This parameter is used for FLMLNK only.

Chapter 20. Sample programs using SCLM services

This chapter contains the following information:

- An example of Pascal program invocations that call the following SCLM services in this order:
  - START
  - INIT
  - LOCK
  - PARSE
  - STORE
  - BUILD
  - FREE
  - END
- A PL/I example that illustrates SCLM service procedures.

Command interface examples written in REXX for the Audit and Versioning Services can be found in ISP.SISPSAMP members FLMSACCT (ACCTINFO), FLMSVERI (VERINFO), FLMSVERR (VERRECOV) and FLMSVERD (VERDEL).

The source code for the Pascal sample programs is found in ISP.SISPSAMP members FLMSRV1, FLMSRV1D, and FLMSRV1S. The source code for the PL/I sample program is found in ISP.SISPSAMP member FLMPLSM.

Pascal example

You can use the following sample Pascal programs to migrate and build a component registered with SCLM. SCLM prompts you for responses as it processes the component. The program prolog contains a description of the required ddnames to be allocated before you start the program.

Note: All requested input parameters must be entered in uppercase characters.

Main program FLMSRV1

```
PROGRAM FLMSRV1 ;

(*********************************************************************)
(*                                                                   *)
(* This program allows you to call SCLM services from a              *)
(* Pascal program.                                                   *)
(*                                                                   *)
(*********************************************************************)
(*********************************************************************)
(********   ALL REQUESTED INPUT PARAMETERS MUST BE ENTERED  **********)
(********   IN UPPERCASE.                                  **********)
(*********************************************************************)
(*********************************************************************)
(*                                                                   *)
(* The function of this program is to register a software component  *)
(* with SCLM and then build it.                                      *)
(* The member in the SCLM controlled library (PDS) to be processed   *)
(* is referenced by the variables project.group.type(member).        *)
(* You must allocate the following ddnames as specified below:       *)
(*                                                                   *)
(* PRSLIST - for parser listings (RECFM=VBA,LRECL=137,BLKSIZE=3120)  *)
(* BLDMSGS - for build messages  (RECFM=F,  LRECL=80, BLKSIZE=80)    *)
(* BLDREPT - for build report    (RECFM=FBA,LRECL=80, BLKSIZE=3120)  *)
(* BLDLIST - for build listings  (RECFM=VBA,LRECL=137,BLKSIZE=3120)  *)
(* BLDEXIT - for build user exit (RECFM=FB, LRECL=160,BLKSIZE=3200)  *)
(*********************************************************************)
(*********************************************************************)
(*              Declare program and interface constants              *)
```

© Copyright IBM Corp. 1990, 2019

```
(*********************************************************************)
CONST

  (* Declare the maximum number of records the accounting record *)
  (* list information array can hold.                            *)
  max_list_info_entries = 200 ;

  (* Declare the required ddnames as constants. *)
  bldmsgs = 'BLDMSGS' ;
  bldrept = 'BLDREPT' ;
  bldlist = 'BLDLIST' ;
  bldexit = 'BLDEXIT' ;

(* Include SCLM Interface common type declarations. *)
%INCLUDE FLMSRV1D ;

(* Include SCLM Interface procedure definitions. *)
%INCLUDE FLMSRV1S ;

(*********************************************************************)
(*              Declare program local variables                      *)
(*********************************************************************)
VAR

  $acct_info                : $acct_info_type       ;
  $list_info                : $list_info_type       ;
  $list_info_copy           : $list_info_type       ;
  $stats_info               : $stats_info_type      ;
  $stats_info_copy          : $stats_info_type      ;
  $msg_array                : $msg_array_type       ;
  breport_check             : char24                ;
  build_scope               : char24                ;
  build_mode                : char24                ;
  access_key                : char16                ;
  appl_id                   : char8                 ;
  authcode                  : char8                 ;
  ddname                    : char8                 ;
  error_listings_only       : char24                ;
  found_group               : char8                 ;
  language                  : char8                 ;
  group                     : char8                 ;
  listing_check             : char24                ;
  max_prom_group            : char8                 ;
  msg_line                  : char80                ;
  prefix_userid             : char17                ;
  project                   : char8                 ;
  project_def               : char8                 ;
  retncode                  : INTEGER               ;
  pds_type                  : char8                 ;
  member                    : char8                 ;
  SCLM_id                   : char8                 ;
  sub_drawdown_mode         : char24                ;
  userid                    : char8                 ;
  verify_cc                 : char24                ;

(*********************************************************************)
(*                    Define the main program                        *)
(*********************************************************************)

BEGIN

  (* Initialize terminal I/O. *)
  TERMIN (INPUT)  ;
  TERMOUT(OUTPUT) ;

  (* Initialize some working variables. *)
  $stats_info_copy := NIL ;
  $list_info_copy  := NIL ;

  (* Get the PDS/member name of the component to process. *)
  WRITELN ('Enter the name of the project to process.' );
  READLN (project);
  WRITELN ('Enter the name of the project definition to process.' );
  READLN (project_def);
  IF
    (project_def = ' ')
  THEN
    project_def := project;
  WRITELN ('Enter the name of the development group to process.');
  READLN  (group);
  WRITELN ('Enter the name of the type to process.');
  READLN  (pds_type);
```

```
WRITELN ('Enter the name of the member to process.');
READLN  (member);
WRITELN ('Enter the language of the source member to register.');
READLN  (language);
(* Issue a request to begin an SCLM service session. *)
SRVSTART ( appl_id,
           retncode );

(* Continue processing only if the request succeeded. *)
IF
  retncode <> 0
THEN
  WRITELN ('SCLM service START failed, error code = ', retncode:-3 )
ELSE BEGIN
  (* Issue a request to initialize an SCLM ID. *)
  msg_line    := ' '     ;
  SRVINIT ( appl_id,
            project,
            project_def,
            SCLM_id,
            msg_line,
            retncode );

  (* Continue processing only if the request succeeded. *)
  IF
    retncode <> 0
  THEN BEGIN
    WRITELN ('SCLM service INIT failed, error code = ', retncode:-3 );
    WRITELN ( msg_line );
  END

  ELSE BEGIN

    (* Issue a request to lock the component. *)
    authcode   := ' ' ;
    $acct_info := NIL ;
    $list_info := NIL ;
    $msg_array := NIL ;
    SRVLOCK ( SCLM_id,
              group,
              pds_type,
              member,
              authcode,
              ' ',              (* access_key *)
              userid,
              found_group,
              max_prom_group,
              $acct_info,
              $list_info,
              $msg_array,
              retncode );

    (* If the lock failed, print associated error messages. *)
    IF
      retncode <> 0
    THEN BEGIN
      WRITELN ('SCLM service LOCK failed, error code = ',            ;
               retncode:-3);                                         ;
      PUTMSGS ( $msg_array );
    END
  ELSE BEGIN

      (* Display some of the accounting record fields *)
      WRITELN ('The component has been locked.' );
      WRITELN ('The component last changed date is: ',
                $acct_info@.change_date );
      WRITELN ('The component last changed time is: ',
                $acct_info@.change_time );
      WRITELN ('The component change-userid is: ',
                $acct_info@.change_userid );
      WRITELN ('The component version number is: ',
                $acct_info@.member_version:-3 );
    END;
    (* Continue processing only if the member has been locked. *)
    IF
      retncode = 0
    THEN BEGIN

      (* Issue a request to parse the component to obtain *)
      (* the statistical information SCLM requires.         *)
      $stats_info := NIL ;
```

```
  SRVPARSE ( SCLM_id,
             group,
             pds_type,
             member,
             language,
             'Y',          (* error_listings_only = yes *)
             'PRSLIST',    (* ddname *)
             $stats_info,
             $list_info,
             $msg_array,
             retncode );

  (* If the parse failed, print associated error messages. *)
  IF
    retncode <> 0
  THEN BEGIN
    WRITELN ('SCLM service PARSE failed, ',
             'error code = ',retncode:-3 );
    PUTMSGS ( $msg_array );
  END
  ELSE BEGIN

    (* Copy all buffered service output into new buffers so    *)
    (* subsequent service calls do not delete the information. *)
    WRITELN ('The component has been parsed.' );
    NEW ( $stats_info_copy );
    $stats_info_copy@ := $stats_info@ ;

    NEW ( $list_info_copy );
    COPYLIST ($list_info, $list_info_copy );
  END;
END;
(* Continue processing only if the member has been parsed. *)
IF
  retncode = 0
THEN BEGIN

  (* Issue a request to register the component with SCLM    *)
  $stats_info := $stats_info_copy ;
  $list_info  := $list_info_copy  ;

  SRVSTORE ( SCLM_id,
             group,
             pds_type,
             member,
             ' ',                (* access_key *)
             language,
             userid,
             'C',                (* sub_drawdown_mode = cond. *)
             'N',                (* verify_cc = no *)
             $stats_info,
             $list_info,
             $msg_array,
             retncode );

  (* If the store failed, print associated error messages. *)
  IF
    retncode <> 0
  THEN BEGIN
    WRITELN ('SCLM service STORE failed, ',
             'error code = ',retncode:-3 );
    PUTMSGS ( $msg_array );
  END;
END;

(* Continue processing only if the member has been stored. *)
IF
  retncode = 0
THEN BEGIN

  (* Issue a request to build the component *)                    )
  (* registered with SCLM.                  *)                    )
  WRITELN ('The component has been stored.' );
  prefix_userid := STR(userid) ;

  SRVBUILD ( SCLM_id,
             group,
             pds_type,
             member,
             userid,
             'N',              (* build_scope = normal      *)
             'C',              (* build_mode  = conditional *)
```

```
                    'N',              (* listing_check = no        *)
                    'Y',              (* breport_check = yes       *)
                    prefix_userid,
                    bldmsgs,          (* dd_bldmsgs *)
                    bldrept,          (* dd_bldrept *)
                    bldlist,          (* dd_bldlist *)
                    bldexit,          (* dd_bldexit *)
                    retncode );
        (* If the build failed, print error messages. *)
        IF
          retncode <> 0
        THEN BEGIN
          WRITELN ('SCLM service BUILD failed, ',
                   'error code = ',retncode:-3 );
          WRITELN ('See the data set allocated to ddname=BLDMSGS ',
                   'for associated error messages.' );
        END
        ELSE
          WRITELN ('The component has been built.' );
      END;

      (* Issue a request to free the SCLM ID. *)
      SRVFREE ( SCLM_id,
                msg_line,
                retncode );
    END;                    (* INIT succeeded *)

    (* Issue a request to end this SCLM service session. *)
    SRVEND ( appl_id,
             msg_line,
             retncode );
  END;                      (* START succeeded *)

  (* Free buffer memory if it is still allocated. *)
  IF
    $stats_info_copy <> NIL
  THEN
    DISPOSE ( $stats_info_copy );

  IF
    $list_info_copy <> NIL
  THEN
    DISPOSE ( $list_info_copy );

END.                        (* Main Program *)
```

Included member FLMSRV1D

```
(*********************************************************************)
(* FLMSRV1D                                                          *)
(*                                                                   *)
(* This member is included by program FLMSRV1                        *)
(*                                                                   *)
(*********************************************************************)
(*********************************************************************)
(*                 Declare Common SCLM Interface Types               *)
(*********************************************************************)
TYPE

  (* Declare arrays of various sizes. *)
  char2   = PACKED ARRAY (. 1..  2 .) OF CHAR ;
  char4   = PACKED ARRAY (. 1..  4 .) OF CHAR ;
  char6   = PACKED ARRAY (. 1..  6 .) OF CHAR ;
  char8   = PACKED ARRAY (. 1..  8 .) OF CHAR ;
  char12  = PACKED ARRAY (. 1.. 12 .) OF CHAR ;
  char16  = PACKED ARRAY (. 1.. 16 .) OF CHAR ;  (* type = ALPHA *)
  char17  = PACKED ARRAY (. 1.. 17 .) OF CHAR ;
  char24  = PACKED ARRAY (. 1.. 24 .) OF CHAR ;
  char43  = PACKED ARRAY (. 1.. 43 .) OF CHAR ;
  char80  = PACKED ARRAY (. 1.. 80 .) OF CHAR ;
  char110 = PACKED ARRAY (. 1..110 .) OF CHAR ;
  char128 = PACKED ARRAY (. 1..128 .) OF CHAR ;

  (* Declare a pointer to an SCLM message array. *)
  $msg_array_type = @ msg_array_type ;
   msg_array_type = PACKED ARRAY (. 1 .. 9999 .) OF char80 ;

  (* Declare a pointer to the static portion *)
  (* of an SCLM accounting record.           *)
```

```
$acct_info_type = @ acct_info_type ;
acct_info_type  =
  RECORD
    acct_group               : char8   ;
    acct_type                : char8   ;
    acct_member              : char8   ;
    SCLM_version             : char2   ;
    accounting_status        : CHAR    ;
    change_date              : char8   ;
    change_time              : char6   ;
    change_group             : char8   ;
    change_userid            : char8   ;
    member_version           : INTEGER ;
    language                 : char8   ;
    authorization_code       : char8   ;
    authorization_code_change : char8  ;
    access_key               : char16  ;
    creation_date            : char8   ;
    creation_time            : char6   ;
    map_date                 : char8   ;
    map_time                 : char6   ;
    predecessor_date         : char8   ;
    predecessor_time         : char6   ;
    promote_date             : char8   ;
    promote_time             : char6   ;
    promote_userid           : char8   ;
    db_qual                  : char8   ;
    translator_version       : char8   ;
    map_name                 : char8   ;
    map_type                 : char8   ;
    language_version         : char8   ;
    total_lines              : INTEGER ;
    comment_lines            : INTEGER ;
    non_comment_lines        : INTEGER ;
    blank_lines              : INTEGER ;
    prolog_lines             : INTEGER ;
    total_stmts              : INTEGER ;
    comment_stmts            : INTEGER ;
    control_stmts            : INTEGER ;
    assignment_stmts         : INTEGER ;
    non_comment_stmts        : INTEGER ;
    number_of_user_entries   : INTEGER ;
    number_of_includes       : INTEGER ;
    number_of_changecodes    : INTEGER ;
    number_of_cus            : INTEGER ;
  END;

(* Declare a pointer to the statistical portion *)
(* of an SCLM accounting record.                *)
$stats_info_type = @ stats_info_type ;
 stats_info_type =
  RECORD
    total_lines             : INTEGER ;
    comment_lines           : INTEGER ;
    non_comment_lines       : INTEGER ;
    blank_lines             : INTEGER ;
    prolog_lines            : INTEGER ;
    total_stmts             : INTEGER ;
    comment_stmts           : INTEGER ;
    control_stmts           : INTEGER ;
    assignment_stmts        : INTEGER ;
    non_comment_stmts       : INTEGER ;
  END;

(* Declare an SCLM list-info change code entry. *)
change_code_record_type =
  RECORD
    change_code  : char8 ;
    date         : char8 ;
    time         : char6 ;
  END;

(* Declare an SCLM list-info EXTD entry. *)
extd_record_type =
  RECORD
    extd_group   : char8  ;
    extd_type    : char8  ;
    extd_name    : char43 ;
    date         : char8  ;
    time         : char6  ;
  END;
```

```
  (* Declare an SCLM list-info compilation unit entry. *)
  cu_record_type =
    RECORD
      cu_name               : char110 ;
      cu_type               : CHAR    ;
      generic_flag          : CHAR    ;
      depend_cu_name        : char110 ;
      depend_cu_type        : CHAR    ;
      depend_cu_depend_type  : CHAR    ;
    END;

  (* Declare an SCLM accounting record list-info entry. *)
  include_record_type =
    RECORD
      member      : char8 ;
      include_set : char8 ;
    END;

  (* Declare an SCLM accounting record list-info entry overlay. *)
  list_info_record_type =
    RECORD
      record_kind : char4 ;
      CASE INTEGER OF
        1: ( member              : char8                   );
        2: ( compool             : char8                   );
        3: ( change_code_record  : change_code_record_type );
        4: ( user_entry          : char128                 );
        5: ( cu_record           : cu_record_type          );
        6: ( extd_record         : extd_record_type        );
        7: ( include_record       : include_record_type     );

    END;

  (* Declare a pointer to an SCLM accounting record list-info array. *)
  $list_info_type = @ list_info_type ;
   list_info_type = PACKED ARRAY (.1..max_list_info_entries.)
                     OF list_info_record_type ;
```

Included member FLMSRV1S

```
(*********************************************************************)
(* FLMSRV1S   SCLM SERVICE INTERFACE PROCEDURE DEFINITIONS           *)
(*                                                                   *)
(* This member is included by program FLMSRV1                        *)
(*                                                                   *)
(*********************************************************************)

(*********************************************************************)
(*                  SCLM START Service Interface                     *)
(*********************************************************************)
PROCEDURE SRVSTART ( VAR appl_id   : char8    ;
                     VAR rc        : INTEGER );

  FUNCTION FLMLNK    ( CONST service  : char8  ;
                       VAR   appl_id  : char8 ): INTEGER ;
                     FORTRAN ;

BEGIN
  rc := FLMLNK ('START   ', appl_id );
END;

(*********************************************************************)
(*                   SCLM INIT Service Interface                     *)
(*********************************************************************)
PROCEDURE SRVINIT ( CONST appl_id     : char8     ;
                    CONST project     : char8     ;
                    CONST project_def : char8     ;
                    VAR   SCLM_id     : char8     ;
                    VAR   msg_line    : char80    ;
                    VAR   rc          : INTEGER ) ;

  FUNCTION FLMLNK ( CONST service     : char8     ;
                    CONST appl_id     : char8     ;
                    CONST project     : char8     ;
                    CONST project_def : char8     ;
                    VAR   SCLM_id     : char8     ;
                    VAR   msg_line    : char80  ) : INTEGER ;
                    FORTRAN ;
```

```
BEGIN
  rc := FLMLNK ('INIT    ', appl_id, project, project_def, SCLM_id,
                msg_line );
END;

(*********************************************************************)
(*                 SCLM FREE Service Interface                       *)
(*********************************************************************)
PROCEDURE SRVFREE ( CONST SCLM_id  : char8     ;
                    VAR   msg_line : char80    ;
                    VAR   rc       : INTEGER ) ;

  FUNCTION FLMLNK ( CONST service  : char8     ;
                    CONST SCLM_id  : char8     ;
                    VAR   msg_line : char80  ) : INTEGER ;
                    FORTRAN ;

BEGIN
  rc := FLMLNK ('FREE    ', SCLM_id, msg_line );
END;

(*********************************************************************)
(*                  SCLM END Service Interface                       *)
(*********************************************************************)
PROCEDURE SRVEND  ( CONST appl_id  : char8     ;
                    VAR   msg_line : char80    ;
                    VAR   rc       : INTEGER ) ;

  FUNCTION FLMLNK ( CONST service  : char8     ;
                    CONST appl_id  : char8     ;
                    VAR   msg_line : char80  ) : INTEGER ;
                    FORTRAN ;

BEGIN
  rc := FLMLNK ('END     ', appl_id, msg_line );
END;

(*********************************************************************)
(*                 SCLM BUILD Service Interface                      *)
(*********************************************************************)
PROCEDURE SRVBUILD ( CONST  SCLM_id        : char8     ;
                     CONST  group          : char8     ;
                     CONST  pds_type       : char8     ;
                     CONST  member         : char8     ;
                     CONST  userid         : char8     ;
                     CONST  build_scope    : char24    ;
                     CONST  build_mode     : char24    ;
                     CONST  listing_check  : char24    ;
                     CONST  breport_check  : char24    ;
                     CONST  prefix_userid  : char17    ;
                     CONST  dd_bldmsgs     : char8     ;
                     CONST  dd_bldrept     : char8     ;
                     CONST  dd_bldlist     : char8     ;
                     CONST  dd_bldexit     : char8     ;
                     VAR    rc             : INTEGER ) ;

   FUNCTION FLMLNK ( CONST  service        : char8     ;
                     CONST  SCLM_id        : char8     ;
                     CONST  group          : char8     ;
                     CONST  pds_type       : char8     ;
                     CONST  member         : char8     ;
                     CONST  userid         : char8     ;
                     CONST  build_scope    : char24    ;
                     CONST  build_mode     : char24    ;
                     CONST  listing_check  : char24    ;
                     CONST  breport_check  : char24    ;
                     CONST  prefix_userid  : char17    ;
                     CONST  dd_bldmsgs     : char8     ;
                     CONST  dd_bldrept     : char8     ;
                     CONST  dd_bldlist     : char8     ;
                     CONST  dd_bldexit     : char8   ) : INTEGER ;
                     FORTRAN ;

BEGIN
  rc := FLMLNK ( 'BUILD   ', SCLM_id, group, pds_type, member, userid,
                 build_scope, build_mode, listing_check, breport_check,
                 prefix_userid,
                 dd_bldmsgs, dd_bldrept, dd_bldlist, dd_bldexit );
END;

(*********************************************************************)
(*                 SCLM LOCK Service Interface                       *)
```

```
(*****************************************************************)
PROCEDURE SRVLOCK ( CONST  SCLM_id         : char8           ;
                    CONST  group           : char8           ;
                    CONST  pds_type        : char8           ;
                    CONST  member          : char8           ;
                    CONST  authcode        : char8           ;
                    CONST  access_key      : char16          ;
                    CONST  userid          : char8           ;
                    VAR    found_group     : char8           ;
                    VAR    max_prom_group  : char8           ;
                    VAR    $acct_info      : $acct_info_type ;
                    VAR    $list_info      : $list_info_type ;
                    VAR    $msg_array      : $msg_array_type ;
                    VAR    rc              : INTEGER )       ;

  FUNCTION FLMLNK ( CONST  service         : char8           ;
                    CONST  SCLM_id         : char8           ;
                    CONST  group           : char8           ;
                    CONST  pds_type        : char8           ;
                    CONST  member          : char8           ;
                    CONST  authcode        : char8           ;
                    CONST  access_key      : char16          ;
                    CONST  userid          : char8           ;
                    VAR    found_group     : char8           ;
                    VAR    max_prom_group  : char8           ;
                    VAR    $acct_info      : $acct_info_type ;
                    VAR    $list_info      : $list_info_type ;
                    VAR    $msg_array      : $msg_array_type):
                                                    INTEGER ;
                    FORTRAN ;

BEGIN
  rc := FLMLNK ( 'LOCK    ', SCLM_id, group, pds_type, member, authcode,
                 access_key, userid,
                 found_group, max_prom_group,
                 $acct_info, $list_info, $msg_array );
END;

(*****************************************************************)
(*                 SCLM PARSE Service Interface                  *)
(*****************************************************************)
PROCEDURE SRVPARSE ( CONST  SCLM_id             : char8            ;
                     CONST  group               : char8            ;
                     CONST  pds_type            : char8            ;
                     CONST  member              : char8            ;
                     CONST  language            : char8            ;
                     CONST  error_listings_only : char24           ;
                     CONST  ddname              : char8            ;
                     VAR    $stats_info         : $stats_info_type ;
                     VAR    $list_info          : $list_info_type  ;
                     VAR    $msg_array          : $msg_array_type  ;
                     VAR    rc                  : INTEGER )        ;
   FUNCTION FLMLNK ( CONST  service             : char8            ;
                     CONST  SCLM_id             : char8            ;
                     CONST  group               : char8            ;
                     CONST  pds_type            : char8            ;
                     CONST  member              : char8            ;
                     CONST  language            : char8            ;
                     CONST  error_listings_only : char24           ;
                     CONST  ddname              : char8            ;
                     VAR    $stats_info         : $stats_info_type ;
                     VAR    $list_info          : $list_info_type  ;
                     VAR    $msg_array          : $msg_array_type ):
                                                          INTEGER ;
                     FORTRAN ;

BEGIN
  rc := FLMLNK ('PARSE   ', SCLM_id, group, pds_type, member, language,
                 error_listings_only, ddname,
                 $stats_info, $list_info, $msg_array );
END;

(*****************************************************************)
(*                 SCLM STORE Service Interface                  *)
(*****************************************************************)
PROCEDURE SRVSTORE (CONST  SCLM_id            : char8           ;
                    CONST  group              : char8           ;
                    CONST  pds_type           : char8           ;
                    CONST  member             : char8           ;
                    CONST  access_key         : char16          ;
                    CONST  language           : char8           ;
                    CONST  userid             : char8           ;
```

```
                    CONST  sub_drawdown_mode  : char24           ;
                    CONST  verify_cc          : char24           ;
                    CONST  $stats_info        : $stats_info_type ;
                    CONST  $list_info         : $list_info_type  ;
                    VAR    $msg_array         : $msg_array_type  ;
                    VAR    rc                 : INTEGER )        ;

  FUNCTION FLMLNK ( CONST  service            : char8            ;
                    CONST  SCLM_id            : char8            ;
                    CONST  group              : char8            ;
                    CONST  pds_type           : char8            ;
                    CONST  member             : char8            ;
                    CONST  access_key         : char16           ;
                    CONST  language           : char8            ;
                    CONST  userid             : char8            ;
                    CONST  sub_drawdown_mode  : char24           ;
                    CONST  verify_cc          : char24           ;
                    CONST  $stats_info        : $stats_info_type ;
                    CONST  $list_info         : $list_info_type  ;
                    VAR    $msg_array         : $msg_array_type) :
                                                INTEGER          ;
                    FORTRAN ;

BEGIN
  rc := FLMLNK ('STORE   ', SCLM_id, group, pds_type, member,
                access_key, language, userid, sub_drawdown_mode,
                verify_cc, $stats_info, $list_info, $msg_array );
END;

(*********************************************************************)
(*      Procedure to print the contents of an SCLM $msg_array.       *)
(*********************************************************************)
PROCEDURE PUTMSGS ( VAR  $msg_array : $msg_array_type );

VAR
  indx : INTEGER ;

BEGIN                   (* Procedure PUTMSGS *)

  (* Print message header information. *)
  WRITELN ('Message array information...');

  (* If the pointer is valid, print the information. *)
  IF
    $msg_array <> NIL
  THEN BEGIN

    (* Loop through the list information. *)
    indx := 1 ;
    WHILE
      $msg_array@(.indx.) <> 'END'
    DO BEGIN
      WRITELN ( $msg_array@(.indx.)) ;
      indx := indx + 1               ;
    END;
  END;                  (* if $msg_array <> nil *)

  (* Reset "$msg_array" to NIL. *)
  $msg_array := NIL;

END;                    (* Procedure PUTMSGS *)

(*********************************************************************)
(*  Procedure to copy an accounting record list information array. *)
(*********************************************************************)
PROCEDURE COPYLIST ( CONST $list_info      : $list_info_type   ;
                     VAR   $list_info_copy : $list_info_type ) ;

VAR
  indx : INTEGER ;

BEGIN                   (* Procedure COPYLIST *)

  (* Only perform the copy if the input list is not nil. *)
  IF
    $list_info <> NIL
  THEN BEGIN

    (* Allocate storage for the copy list if the caller *)
    (* has not yet done this.                           *)
    IF
      $list_info_copy = NIL
```

```
    THEN
      NEW ( $list_info_copy );

    (* Loop through the list information, copying entry-by-entry. *)
    indx := 1 ;
    REPEAT
      $list_info_copy@(.indx.) := $list_info@(.indx.) ;
      indx := indx + 1                                ;
    UNTIL
      ($list_info@(.indx-1.).record_kind = 'END ')
                    OR
      (indx > max_list_info_entries) ;

    (* Check for overflow condition. *)
    IF
      indx > max_list_info_entries
    THEN BEGIN
      WRITELN ('*** ERROR *** List information array overflowed!' );
      WRITELN ('*** ERROR *** Increase size of program constant.' );
    END;
  END;                  (* if $list_info <> nil *)
END;                    (* Procedure COPYLIST *)
```

PL/I example

Here is a sample PL/I program for SCLM service procedures.

```
 FLMPLSM: PROC (PARMS) OPTIONS(MAIN NOEXECOPS);
 /*********************************************************************/
 /*                                                                   */
 /* PL/I PROGRAM WHICH CALLS SCLM SERVICES                            */
 /*                                                                   */
 /* PROCEDURES IN THIS PROGRAM:                                       */
 /*                                                                   */
 /* -SCLSTRT  START  SCLM SESSION                                     */
 /* -SCLINIT  INIT   SCLM_ID                                          */
 /* -SCLEDIT  EDIT   SCLM SOURCE MEMBER                               */
 /* -SCLFREE  FREE   SCLM_ID                                          */
 /* -SCLEND   END    SCLM SESSION                                     */
 /*                                                                   */
 /*********************************************************************/
 /*                                                                   */
 /* DECLARATIONS                                                      */
 /*                                                                   */
 /*********************************************************************/
 DCL PLIRETV    BUILTIN ;
 DCL FLMLNK     ENTRY EXTERNAL OPTIONS(ASM,INTER,RETCODE) ;
 DCL ISPLINK    ENTRY EXTERNAL OPTIONS(ASM,INTER,RETCODE) ;

 /*********************************************************************/
 /* PARAMETERS USED IN THIS PROGRAM                                   */
 /*********************************************************************/
 DCL PARMS              CHAR(80) VARYING;

 DCL 1 PARM,
       2 PARM1          CHAR(8)  INIT('') ,
       2 DELM1          CHAR(1)  INIT('') ,
       2 PARM2          CHAR(8)  INIT('') ,
       2 DELM2          CHAR(1)  INIT('') ,
       2 PARMX          CHAR(62) INIT('') ;

 /*********************************************************************/
 /* VARIABLES USED BY SCLM SERVICES                                   */
 /*********************************************************************/
 DCL SERVICE            CHAR(8)  INIT(' ') ;
 DCL APPL_ID            CHAR(8)  INIT(' ') ;
 DCL SCLM_ID            CHAR(8)  INIT(' ') ;
 DCL PRJ_DEF            CHAR(8)  INIT(' ') ;
 DCL PROJECT            CHAR(8)  INIT(' ') ;
 DCL MSG_LINE           CHAR(80) INIT(' ') ;

 DCL Y                  CHAR(24) INIT('Y                       '),
     C                  CHAR(24) INIT('C                       '),
     N                  CHAR(24) INIT('N                       ');

 DCL SLMLIB             CHAR(8),
     SLMLIB2            CHAR(8),
```

```
    SLMLIB3             CHAR(8),
    SLMLIB4             CHAR(8),
    ALL_HIER            CHAR(24),
    IMAC                CHAR(8),
    PROF                CHAR(8),
    CONFIRM             CHAR(24),
    MIX                 CHAR(24),
    WS                  CHAR(24),
    PRESERVE            CHAR(24),
    AUTHCODE            CHAR(8),
    CHGCODE             CHAR(8),
    VOLSER              CHAR(8),
    SLMPROJ             CHAR(8),
    SLMALTP             CHAR(8),
    SLMTYP              CHAR(8),
    SLMMEM              CHAR(8),
    MSGLIST             CHAR(80);

DCL BLNK8               CHAR(8) INIT(' '),
    DDNAME              CHAR(8),
    DONE                BIT(1);

/*********************************************************************/
/*   MAIN PROGRAM LOGIC                                              */
/*********************************************************************/

PARM1      = 'PROJECT ';
PARM2      = 'ALT_PROJ';
PROJECT    = PARM1;
PRJ_DEF    = PARM2;
IF PRJ_DEF = ' ' THEN PRJ_DEF = PROJECT ;


CALL SCLSTRT ;
CALL SCLINIT ;
CALL SCLEDIT ;
CALL SCLFREE ;
CALL SCLEND ;

/*********************************************************************/
/* GENERATE AN APPLICATION ID FOR SCLM SESSION                       */
/*********************************************************************/
SCLSTRT: PROC ;
SERVICE    = 'START';
APPL_ID    = '';

CALL FLMLNK (SERVICE , APPL_ID) ;
RETCODE    = PLIRETV() ;

END SCLSTRT ;

/*********************************************************************/
/* INITIALIZE SCLM ID FOR SERVICES                                   */
/*********************************************************************/
SCLINIT: PROC ;
SERVICE    = 'INIT' ;
SCLM_ID    = '';

CALL FLMLNK (SERVICE , APPL_ID
                     , PROJECT
                     , PRJ_DEF
                     , SCLM_ID
                     , MSG_LINE) ;
RETCODE    = PLIRETV() ;

END SCLINIT ;

/*********************************************************************/
/* EDIT A MEMBER IN THE PROJECT HIERARCHY                            */
/*********************************************************************/
SCLEDIT: PROC ;
SLMLIB     = 'DEV1    ';
SLMLIB2    = '        ';
SLMLIB3    = '        ';
SLMLIB4    = '        ';
SLMTYP     = 'SOURCE  ';
SLMMEM     = 'MEMBER1 ';
SERVICE    = 'EDIT    ';
DDNAME     = 'EDIT    ';
ALL_HIER   = N;
IMAC       = '        ';
PROF       = '        ';
```

```
CONFIRM    = N;
MIX        = N;
WS         = N;
PRESERVE   = 'Y';
AUTHCODE   = ' ';
CHGCODE    = '    ';
VOLSER     = BLNK8;

CALL FLMLNK(SERVICE,SCLM_ID,SLMLIB,
            SLMLIB2,SLMLIB3,SLMLIB4,
            SLMTYP,SLMMEM,ALL_HIER,
            IMAC,PROF,CONFIRM,MIX,WS,
            PRESERVE,AUTHCODE,CHGCODE,
            VOLSER,DDNAME);

RETCODE    = PLIRETV() ;
IF RETCODE > 0 THEN
    CALL ISPLINK('BROWSE  ','SCLM.MSGS ');

END SCLEDIT ;

/*********************************************************************/
/* FREE SCLM ID                                                      */
/*********************************************************************/
SCLFREE: PROC ;
SERVICE    = 'FREE    ' ;

CALL FLMLNK (SERVICE, SCLM_ID
                    , MSG_LINE) ;
RETCODE    = PLIRETV() ;

END SCLFREE ;

/*********************************************************************/
/* END AN SCLM SERVICES SESSION                                      */
/*********************************************************************/
SCLEND: PROC ;
SERVICE    = 'END     ' ;

CALL FLMLNK (SERVICE, APPL_ID
                    , MSG_LINE ) ;
RETCODE    = PLIRETV() ;

END SCLEND ;

END;
```

Chapter 21. SCLM macros

SCLM supplies a set of macro instructions that you can use to define project definitions. This topic describes those macro instructions, explaining the format of each. The macros described here are contained in ISP.SISPMACS, which is delivered with the product.

The macros appear in alphabetical order. For each macro, the chapter provides the command format, a description of the parameters you use, and an example.

Table 31. Macros

Macro	Description
“FLMABEG macro” on page 493	Define project name of the project definition
“FLMAEND macro” on page 493	Last macro in the project definition
“FLMAGRP macro” on page 494	Define a set of authorization codes
“FLMALLOC macro” on page 494	Many uses
“FLMALTC macro” on page 510	Specify control information
“FLMATVER macro” on page 513	Enable audit and version utility
“FLMCNTRL macro” on page 516	Specify project specific control options
“FLMCPYLB macro” on page 538	Identify data set name to be allocated
“FLMGROUP macro” on page 540	Define groups in the project definition
“FLMINCLS macro” on page 542	Associate include sets with types in the project hierarchy
“FLMLANGL macro” on page 545	Define a language to SCLM
“FLMLRBLD macro” on page 548	Cause certain members to be rebuilt when promoted into particular groups.
“FLMPROJ macro” on page 549	Define a subproject to SCLM.
“FLMSYSLB macro” on page 551	Define system macro or include data sets
“FLMTCOND macro” on page 552	Run or skip build translators
“FLMTOPTS macro” on page 556	Vary options passed to a build translator
“FLMTRNSL macro” on page 557	Similar to JCL EXEC statements
“FLMTYPE macro” on page 562	Define types in the project definition

Notes on using the SCLM macros

Because SCLM macros are S/370 Assembler macros, all rules pertaining to Assembler macros apply to them. For more information about Assembler macros, see High Level Assembler and Toolkit Feature in IBM Knowledge Center (www.ibm.com/support/knowledgecenter/SSENW6).

These additional SCLM guidelines apply to the use of SCLM macros:

- Assembler does not support blanks in macro parameters. If a blank is used in a parameter that is delimited by parentheses, everything on the line after the blank will be ignored. If you use single quotes to delimit parameters, be careful when continuing to a new line. All blanks between the first single quote and the continuation character are considered part of the parameter. This can result in exceeding

© Copyright IBM Corp. 1990, 2019

the maximum parameter length. If you need more than 71 characters for a line of code, you must put a continuation character in column 72 and begin the remaining lines in column 16.

- The maximum length of a parameter is 255 characters. (This is the maximum length of an Assembler macro call operand.)
- If any commas are omitted from the parameter list for any of the macros, the project definition might assemble correctly, but SCLM might use different defaults than expected, resulting in errors. All parameters except the last must be followed by a comma.
- If an optional parameter is specified without a value or the parameter is not specified, the default value is used; for example, PARM2A= or not specifying PARM2A on the macro statement causes PARM2A to default to Y. If the parameter does not have a default value then no value (null) is specified for the parameter.
- SCLM handles invalid values for required and optional parameters differently. If you specify an invalid value for a required parameter, SCLM might issue an error message and the project definition assembly ends with a return code of 8. If you specify an invalid value for an optional parameter, SCLM issues a warning message, uses the default value for the parameter, and returns a return code of 4. Limited verification of the parameters is done during the assembly of the project definition. In many cases, the error does not occur until run time. An MNOTE is added to the assembly listing to indicate the invalid parameter specifications.
- SCLM performs validation of data set names when the data sets are opened. SCLM performs validation of VSAM versioning data set names when they are required for use by SCLM.
- The messages you receive when the project definition is assembled are issued from the SCLM macros and the assembler. SCLM messages are identified as MNOTEs. For more information, refer to "SCLM macro messages (MNOTEs)" in *z/OS ISPF Messages and Codes*. For explanations of other messages, see High Level Assembler and Toolkit Feature in IBM Knowledge Center (www.ibm.com/support/knowledgecenter/SSENW6).

  The SCLM MNOTEs appear in one of two places within the listing:

  – Directly after the SCLM macro statement that contains incorrect parameter values
  – Near the end of the listing where SCLM cross-checks the values of the various SCLM macro statements. Assembler error messages usually occur inline where the syntax error was made.

Using SCLM variables in SCLM macros

Some SCLM macros accept SCLM variables as input parameters. The variable names all begin with @@FLM. For example, @@FLMPRJ is the name of the project and @@FLMMBR is the name of the member. A typical parser translator will have an allocation that looks like this:

```
FLMALLOC  IOTYPE=A,DDNAME=SOURCE
 FLMCPYLB @@FLMDSN(@@FLMMBR)
```

Valid variables are indicated in the parameter descriptions for the particular macros to which they apply. If the description of a parameter does not list valid variables, no variables can be used for that parameter. For more information on any variable mentioned in the descriptions, see Chapter 23, “SCLM Variables and Metavariables,” on page 631.

The variables are substituted into the string exactly as they are specified. If the data set name is ISPF.DEV1.SOURCE and the member name is TEST, the variables in the example are resolved as ISPF.DEV1.SOURCE(TEST). SCLM inserts the parentheses without the project manager having to use any concatenation characters.

This can be very useful. However, note that any stray characters before or after the variable name will also be appended. In this example, if you accidentally typed @@FLMTYPE instead of @@FLMTYP the options string resolves to ISPF.BACKUP.SOURCEE instead of ISPF.BACKUP.SOURCE:

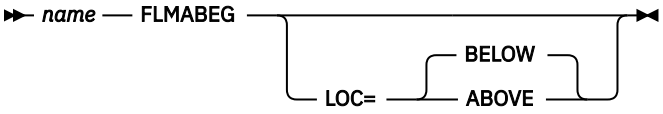
```
FLMTRNSL   CALLNAM='BACKUP  ',                                  C
      FUNCTN=VERIFY,                                            C
      COMPILE=DSBKUP,                                           C
      DSNAME=SCLM.CLIST,                                        C
      GOODRC=4,                                                 C
```

```
                PORDER=1,                                                C
                CALLMETH=TSOLNK,                                         C
                VERSION=1.0,                                             C
                OPTIONS='@@FLMPRJ.BACKUP.@@FLMTYPE'
```

FLMABEG macro

Use this macro to define the project name of the project definition. It is required for the project definition and must appear before the other SCLM macros in the project definition.

Macro format



Parameters

name
An 8-character project name. For alternate project definitions, use the "main" project definition name; this is the high-level qualifier of the project definition LOAD data set.

LOC=ABOVE|BELOW
This specifies residency mode of the resultant project definition module. If you specify BELOW, the module will be created with RMODE=24 (commonly called Below The Line). If you specify ABOVE, it will be created with RMODE=31 (Above The Line). If LOC is not specified, it defaults to BELOW.

Example

PROJ1 is the name of the project being specified by this project definition.

```
PROJ1 FLMABEG
```

FLMAEND macro

Use this macro as the last macro in the project definition. All SCLM macros you use to define the project definition must appear between the FLMABEG and FLMAEND macros. It is required and must be the last macro in the project definition.

Macro format

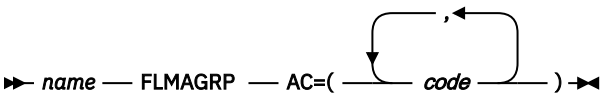
Parameters

This macro has no parameters.

FLMAGRP macro

Use this macro to define a set of authorization codes. You can then specify the authorization group name in the AC field on the FLMGROUP macro to assign the set of authorization codes to that group name.

Macro format



Parameters

name
An 8-character authorization group name containing no special characters or embedded blanks.

AC=(*code*)
A list of authorization codes and authorization groups that you can assign to the authorization group name. If *code#* is an authorization group, then you must have previously defined it with the FLMAGRP macro. Each authorization code or group can be up to 8 characters and cannot contain commas. The maximum number of characters allowed is 255, including commas and the delimiting parentheses.

Example

Authorization group SET1 contains the authorization codes R3M0, R3M1, and R3M2. Authorization group SET2 contains two authorization codes, R1M0 and R2M0, and one previously defined authorization group, SET1, for a total of five authorization codes (R1M0, R2M0, R3M0, R3M1, and R3M2).

```
SET1 FLMAGRP AC=(R3M0,R3M1,R3M2)
SET2 FLMAGRP AC=(R1M0,R2M0,SET1)
```

FLMALLOC macro

This macro provides these capabilities to SCLM:

- Allocate temporary or permanent data sets that are used by translators.

  FLMALLOC provides a limited equivalency to JCL DD or TSO ALLOCATE statements in your procedure libraries. The FLMALLOC parameters that provide this capability are BLKSIZE, CATLG, DDNAME, DIRBLKS, DISP, DSNTYPE, LRECL, RECFM, RECNUM, MEMBER, DINIT, MALLOC, and ALLCDEL. DINIT, MALLOC, and ALLCDEL indicate how the data set is to be dispositioned.

  When allocating permanent data sets, use IOTYPE=A or I. When allocating temporary data sets, use IOTYPE=O, P, S, or W.

  IOTYPEs A and I are used to associate a ddname with data sets that already exist.

  IOTYPE H is used to associate a ddname with a z/OS UNIX file. The file can be existing or new.

  IOTYPE=S is used for input data from an SCLM-controlled library. The member in this library is copied into a temporary data set for use by the translator.

IOTYPE=O is used for output data to be stored in an SCLM-controlled data set. A temporary sequential data set is allocated for use by the translator and the output produced by the translator is copied into the member in the SCLM-controlled data set.

IOTYPE=P is used for output data to be stored in an SCLM-controlled library. A temporary partitioned data set is allocated for use by the translator and members produced by the translator are copied into an SCLM-controlled data set.

In general, if the translator writes to a sequential data set, use IOTYPE=O; if the translator writes to a member of a partitioned data set, use IOTYPE=P.

IOTYPE=W is used to allocate temporary data sets for use by the translator. These data sets are discarded at the completion of a BUILD or PROMOTE.

SCLM creates temporary data sets for the translators to use rather than allocating directly to the hierarchy data sets. This protects the integrity of the project hierarchy data when the translator is producing output that will be stored in the hierarchy. The output is copied from the temporary data sets to the project hierarchy after all the translators have been invoked. If multiple translators are invoked, DDNAMEs for the temporary outputs must be unique for each translator or only the outputs from the last translator will be copied.

All of the allocations for FLMALLOC macros that follow the FLMTRNSL macro are performed just before the translator is invoked. The exception to this rule applies when MALLOC=Y; see the description of MALLOC for details. The ordering of FLMALLOC macros in relation to FLMTRNSL macros is similar to the ordering of DD statements in relation to EXEC statements in JCL.

Temporary data sets that were created by SCLM are deleted when all of the build translators have completed processing for the member being built.

- Control the contents of the ddname substitution list

  The ddname substitution list is passed as a parameter to a translator that has PORDER=2 or 3. If PORDER=0 or 1, SCLM does not generate a ddname substitution list. Not all translators accept ddname substitution lists. If a translator does accept a ddname substitution list, the ddnames in the list are used to override the default ddnames used by the translator.

  In addition to ddnames, sometimes the ddname substitution list specifies the name of a member to be created. This is true for the linkage editor used by SCLM. The linkage editor accepts the name of the member to be created as a parameter in the ddname substitution list. Valid FLMALLOC parameters for this capability are KEYREF and IOTYPE. Use IOTYPE=L to add a member name to its ddname substitution list.

  Ddname substitution lists are generated through the use of FLMALLOC statements. Each FLMALLOC statement adds a ddname to the list. The order in which the ddnames appear in the list is defined by the order of the FLMALLOC statements.

  For general information about ddname substitution lists, refer to "Invoking Utility Programs from an Application Program" in *z/OS DFSMSdfp Utilities*. See the manuals for the specific translator being invoked for details on the substitution list contents expected. For IBM supplied compilers, this information is located in the compiler's Programmers Guide manual under "Invoking Compiler from Application Programs" or "Dynamic Invocation of Compiler".

- Identify hierarchy data to be used or created by a translator

  FLMALLOC can be used to identify information in the hierarchy that is either the input to a translator or the destination of the output from a translator.

  The FLMALLOC parameters that are valid for this purpose are MALLOC, NOSAVRC, DFLTTYP, KEYREF, and LANG. The IOTYPE identifies whether the allocation is for input to or output from a translator. To identify the members of the hierarchy to use as input to a translator, use IOTYPE=S or A. To identify the temporary output data sets that SCLM should store in the hierarchy, use IOTYPE=O or P. If you want the information to be read from or saved in the project hierarchy, you must specify the KEYREF parameter, except for IOTYPE=S, which defaults to SINC.

- Identify the translator data sets to be copied to a listing data set

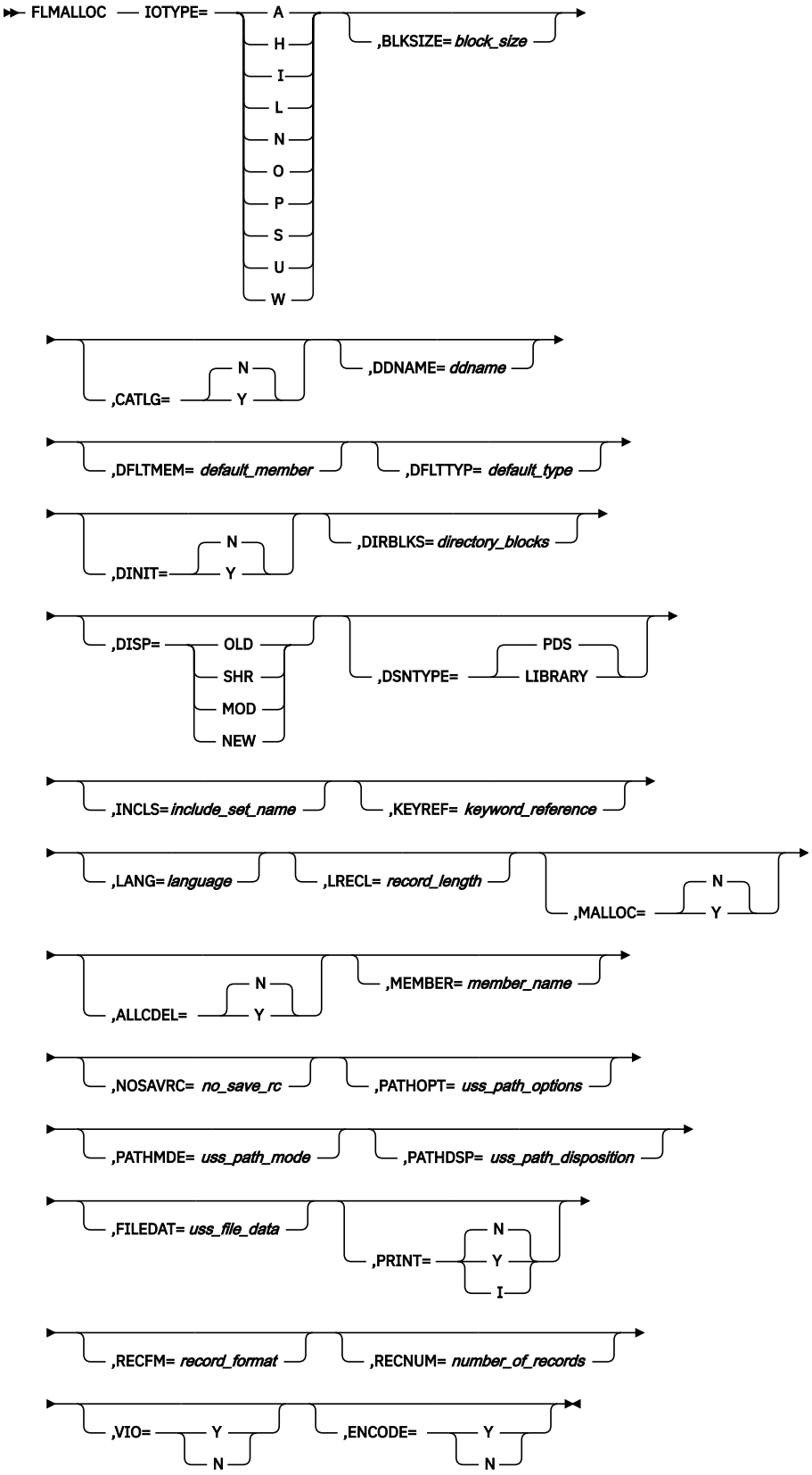
Use the PRINT parameter to copy the contents of a temporary data set to the listing data set. A listing data set is a sequential data set that contains any list information returned from a translator. Listings can be created by the SCLM build, promote, parse, migrate, or save services and by SCLM Edit. See the PRINT parameter description for more information.

- Use the output of one translator as input to another translator

  This capability is used when one translator creates information that is required by another translator. This is only possible when multiple translators are defined for a language definition for the same function (FUNCTN=) value.

  Use IOTYPE=U to indicate that the output from a previously called translator is to be used. See IOTYPE=U for more information.
- Allows SCLM outputs specified with IOTYPE=O or P to be encoded. As a part of this encoding process the member is compressed to save space.

Macro format


FLMALLOC
IOTYPE=
A
H
I
L
N
O
P
S
U
W
,BLKSIZE= block_size
,CATLG=
N
Y
,DDNAME= ddname
,DFLTMEM= default_member
,DFLTTYP= default_type
,DINIT=
N
Y
,DIRBLKS= directory_blocks
,DISP=
OLD
SHR
MOD
NEW
,DSNTYPE=
PDS
LIBRARY
,INCLS= include_set_name
,KEYREF= keyword_reference
,LANG= language
,LRECL= record_length
,MALLOC=
N
Y
,ALLCDEL=
N
Y
,MEMBER= member_name
,NOSAVRC= no_save_rc
,PATHOPT= uss_path_options
,PATHMDE= uss_path_mode
,PATHDSP= uss_path_disposition
,FILEDAT= uss_file_data
,PRINT=
N
Y
I
,RECFM= record_format
,RECNUM= number_of_records
,VIO=
Y
N
,ENCODE=
Y
N

Parameters

IOTYPE=A|H|I|L|N|O|P|S|U|W

Specifies the type of data sets to be allocated and how these data sets can be used. FLMALLOC has different capabilities based on the IOTYPE assigned to it. Therefore, through the use of IOTYPEs, FLMALLOC is like nine different macros.

Figure 209 on page 498 shows the language definition, FLM01ASM, which is delivered in the partitioned data set ISP.SISPSAMP. This sample is available as part of the ISPF product. Refer to this sample as necessary to understand the different IOTYPEs. Of course, not all IOTYPEs are used by any one language definition, but this will provide some aid in understanding most IOTYPEs.

```
*********************************************************************
*            HIGH LEVEL ASSEMBLER LANGUAGE DEFINITION FOR SCLM
*
* POINT THE FLMSYSLB MACRO(S) AT ALL 'STATIC' COPY DATA SETS
* ADD FLMCPYLB MACROS FOR EACH FLMSYSLB, TO THE 'SYSLIB' FLMALLOC MACRO
* CUSTOMIZE THE 'OPTIONS' AND 'GOODRC' FIELDS TO YOUR STANDARDS.
* ADD THE 'DSNAME' FIELD IF THE TRANSLATOR IS IN A PRIVATE LIBRARY.
* WHEN A NEW TRANSLATOR VERSION REQUIRES TOTAL RECOMPILATION FOR THIS
* LANGUAGE, THE 'VERSION' FIELD ON FLMLANGL SHOULD BE CHANGED.
*********************************************************************
ASM      FLMSYSLB SYS1.MACLIB
*
         FLMLANGL    LANG=ASM,VERSION=ASMV1.0
*
* PARSER TRANSLATOR
*
          FLMTRNSL   CALLNAM='SCLM ASM PARSE',                        C
               FUNCTN=PARSE,                                          C
               COMPILE=FLMLPGEN,                                      C
               PORDER=1,                                              C
               OPTIONS=(SOURCEDD=SOURCE,                              C
               STATINFO=@@FLMSTP,                                     C
               LISTINFO=@@FLMLIS,                                     C
               LISTSIZE=@@FLMSIZ,                                     C
               LANG=A)               *** THIS IS ASSEMBLER ONLY ***
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
* BUILD TRANSLATOR(S)
*
*
*         --ASSEMBLER INTERFACE--
          FLMTRNSL   CALLNAM='ASSEMBLER',                             C
               FUNCTN=BUILD,                                          C
               COMPILE=ASMA90,                                        C
               VERSION=1.0,                                           C
               GOODRC=0,                                              C
               PORDER=1,                                              C
               OPTIONS=(XREF(SHORT),LINECOUNT(75),OBJECT,RENT)
*
* DDNAME ALLOCATIONS
*
 FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=5000
 FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,RECNUM=17500
 FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,RECNUM=7500,DFLTTYP=OBJ
 FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
  FLMCPYLB NULLFILE
 FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
  FLMCPYLB NULLFILE
 FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
* ADD ONE FLMCPYLB FOR EACH FLMSYSLB
  FLMCPYLB SYS1.MACLIB
 FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,KEYREF=LIST,PRINT=Y,             C
               DFLTTYP=LIST,RECNUM=20000
*
* 5694-A01 COPYRIGHT IBM CORP 1980, 2007
```

Figure 209. Sample language definition for Assembler

For the purpose of explanation, assume that any source modules built by sample architecture definitions given in these descriptions have been saved with the preceding language definition.

IOTYPE=A
Allocate a permanent data set or set of permanent data sets for either input or output. You need the FLMCPYLB macro to identify the data sets. There is an MVS limitation to the number of data sets that can be allocated to the ddname; the maximum is 123 data sets. The data sets allocated using this IOTYPE can be either partitioned or sequential. The default disposition is SHR. The DISP parameter can be used to override the default when a single data set is to be allocated. For example, you might use the override when you want to allocate a data set to be used for output with a disposition of OLD. If more than one data set is allocated, the DISP parameter must be SHR.

For example, in Figure 209 on page 498, IOTYPE=A is used to allocate the SOURCE DDNAME. This identifies the source data set that will be used by the 'SCLM ASM PARSE' translator. If you use SCLM Edit to save member FLM01MD1 in type SOURCE and group DEV1 of project PROJ1, the FLMCPYLB statement identifies 'PROJ1.DEV1.SOURCE(FLM01MD1)' as the member to allocate as input to the parser.

IOTYPE=H
Allocate a z/OS UNIX file for either input or output. Use the FLMCPYLB macro to define the pathname. Additional requirements may be supplied using the PATHOPT, PATHMDE, PATHDSP, and FILEDAT parameters.

IOTYPE=I
Allocate libraries in the hierarchy for an include set. The INCLS parameter indicates the name of an include set as specified on an FLMINCLS macro. If no INCLS parameter is specified, the default include set is used.

A value should be specified for the KEYREF parameter or it will default to SINC and a warning message will be issued. The data sets allocated depend on the value of the KEYREF parameters:

- For KEYREF=SREF, the hierarchy for the SREF type is allocated.
- For KEYREF=CREF, the hierarchy for the CREF type is allocated.
- For KEYREF=SINC, the INCLS parameter indicates that the types allocated are listed in the FLMINCLS macro for the include set. The FLMSYSLB data sets are allocated if ALCSYSLB=Y is specified on the FLMLANGL macro for the language, followed by any data sets specified on FLMCPYLB macros.

  SCLM allocates all of the data sets for the types associated with the include set within the current view of the hierarchy. The starting group for the hierarchical view is defined by the group used as input to the function, rather than the group where the referenced member was found. The hierarchies for each type are allocated in the order specified on the FLMINCLS macro.

This allocation is typically used to resolve include dependencies when performing a compilation. FLMCPYLBs that follow this allocation should not reference SCLM-controlled data sets.

At least one data set must exist in the hierarchy for the types referenced.

For example, in Figure 209 on page 498, IOTYPE=I with KEYREF=SINC is used to allocate the SYSLIB DDNAME. Because no INCLS parameter is specified for the IOTYPE=I, the default include set is used. In addition, because no FLMINCLS macro is specified for the default include set in this language definition, an FLMINCLS macro is generated with TYPES=(@@FLMTYP,@@FLMETP). If the example project hierarchy has been set up according to the steps identified in “Project manager scenario” on page 39, member FLM01CMD for 'PROJ1.RELEASE.ARCHDEF' contains these statements:

```
*
*
* Object Module 1
* OBJ FLM01MD1 OBJ
 LIST FLM01MD1 LIST
 SINC FLM01MD1 SOURCE
 PARM NOXREF,LC(75)
```

If a build was performed on this member at the DEV1 group, the FLMALLOC macro would indicate that a hierarchy should be allocated starting at the DEV1 group for the type indicated by the SINC card. In

this case, 'PROJ1.DEV1.SOURCE', 'PROJ1.TEST.SOURCE', 'PROJ1.RELEASE.SOURCE', and 'SYS1.MACLIB' would be allocated.

If you were to look at the project definition for PROJ1, you would see this macro that defines the SOURCE type:

```
             SOURCE  FLMTYPE
```

Notice that there is no extend type defined. If, however, this type had been defined as follows:

```
             SOURCE  FLMTYPE EXTEND=SOURCE2
```

then building this member at the DEV1 group would have resulted in these data sets being allocated in order: 'PROJ1.DEV1.SOURCE', 'PROJ1.TEST.SOURCE', 'PROJ1.RELEASE.SOURCE', 'PROJ1.DEV1.SOURCE2', 'PROJ1.TEST.SOURCE2', 'PROJ1.RELEASE.SOURCE2', and 'SYS1.MACLIB'.

If the extended type SOURCE2 was defined as shown in the preceding macro, and a build was performed at the TEST group, these data sets would be allocated, in order: 'PROJ1.TEST.SOURCE', 'PROJ1.RELEASE.SOURCE', 'PROJ1.TEST.SOURCE2', 'PROJ1.RELEASE.SOURCE2', and 'SYS1.MACLIB'.

In this example, the default values have been used for the include set. If the FLMALLOC macro for IOTYPE=I had been written as follows, the include set of SYSLIB would have been used:

```
 FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC,INCLS=SYSLIB
* ADD ONE FLMCPYLB FOR EACH FLMSYSLB
  FLMCPYLB SYS1.MACLIB
```

In the previous example, the default values have been used for the include set and the FLMSYSLB data sets were not allocated. If the FLMLANGL macro had ALCSYSLB=Y and the FLMALLOC macro for IOTYPE=I had been written as follows, the include set of SYSLIB would have been used:

```
FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC,INCLS=SYSLIB
* COPYLIB ALLOCATION OF FLMSYSLB DATA SETS IS DONE AUTOMATICALLY
```

The FLMSYSLB macro would need to specify the include set using the INCLS parameter.

```
ASM    FLMSYSLB SYS1.MACLIB,INCLS=SYSLIB
```

An FLMINCLS macro is required in the language definition to indicate the types to be included in the allocation. This FLMINCLS macro first searches the MACROS type followed by the type and extended type.

```
* INDICATE THE TYPES TO SEARCH FOR THE SYSLIB INCLUDE-SET
SYSLIB    FLMINCLS TYPES=(MACROS,@@FLMTYP,@@FLMETP)
```

Using the preceding FLMINCLS macro, SCLM allocates data sets in this order for the SYSLIB ddname when building at group DEV1:

1. 'PROJ1.DEV1.MACROS'
2. 'PROJ1.TEST.MACROS'
3. 'PROJ1.RELEASE.MACROS'
4. 'PROJ1.DEV1.SOURCE'
5. 'PROJ1.TEST.SOURCE'
6. 'PROJ1.RELEASE.SOURCE'
7. 'PROJ1.DEV1.SOURCE2'
8. 'PROJ1.TEST.SOURCE2'
9. 'PROJ1.RELEASE.SOURCE2'
10. 'SYS1.MACLIB'

IOTYPE=L

Pass a member name in the ddname substitution list. See the PORDER parameter in “FLMTRNSL macro” on page 557 for more information. The KEYREF parameter identifies the member name and type. This IOTYPE is commonly used to identify the load module name for the S/370 linkage editor.

Note: IOTYPE=L is only valid when the PORDER parameter in the FLMTRNSL macro is set to 2 or 3.

For example, Figure 209 on page 498 is not a linkage editor language definition; therefore, it does not contain an example of IOTYPE=L. However, FLM01370 in ISP.SISPSAMP, part of the sample project definition, contains an example of IOTYPE=L. If the example project hierarchy has been set up according to the steps identified in “Project manager scenario” on page 39, member FLM01LD1 for 'PROJ1.RELEASE.ARCHDEF' contains these statements:

```
*
*   Load Module LMOD1
*
LOAD  FLM01LD1 LOAD
LMAP  FLM01LD1 LMAP
INCL  FLM01CMD ARCHDEF
PARM  MAP,NCAL,
PARM  LET
```

If a build was performed for this architecture definition, the FLMALLOC macro with IOTYPE=L and KEYREF=LOAD would pass "FLM01LD1" to the Linkage Editor.

IOTYPE=N

Skip over a field during ddname substitution. This IOTYPE is valid only for PORDER=2 or 3. SCLM adds 8 bytes of hexadecimal zeros to the ddname substitution list.

For example, this IOTYPE is not used by Figure 209 on page 498, but if a translator accepted a ddname substitution list, using this IOTYPE on the FLMALLOC macro would result in 8 hexadecimal zeros being placed in the ddname substitution list.

IOTYPE=O

Allocate a sequential temporary data set that will contain output from a translator that is to be saved in the project hierarchy. A KEYREF parameter must be used to identify the output member name and type. Valid KEYREF values are OBJ, COMP, LIST, LOAD, LMAP, and OUTx.

Note: If the outputs of a translator are empty files then SCLM will copy the translator outputs into the hierarchy as empty members and create accounting records for these members.

For example, in Figure 209 on page 498, IOTYPE=O is used to allocate the SYSPRINT DDNAME. This is a temporary data set into which the translator will write the Assembler listing. See the example for IOTYPE=I for an illustration of what is contained in architecture definition member FLM01CMD. If this member were built at DEV1, the build listing would be copied into the hierarchy into the member and type specified by the LIST card, FLM01MD1 LIST.

This example is building an architecture definition, so DFLTTYP will be ignored or overridden by the LIST card. If only the source were being built, the listing would go into the type specified by DFLTTYP.

IOTYPE=P

Allocate a temporary partitioned data set that will contain output from a translator that is to be saved in the project hierarchy. The dsntype parameter is used to indicate whether the temporary data set should be allocated as PDS or PDSE.

If the output from a translator is to be saved in the project hierarchy, then a KEYREF parameter must be used to identify the target member into which the translator output will be copied. Specify any output KEYREF value, such as KEYREF=LOAD or OUTx. If the output from a translator is not to be saved in the project hierarchy, do not specify a KEYREF parameter; this is essentially like using IOTYPE=W except that a partitioned data set is allocated for use by the translator instead of a sequential data set.

If the build is to occur on a workstation, use IOTYPE=P and DFLTMEM=* to take advantage of workstation build caching. IOTYPE=P preserves the workstation file's date and time information as it is copied to the host. If the output is needed as input for another build step, the date and time at the

host member is compared the date and time of the corresponding workstation file. If they match, the file is considered to be the same, and the file is not transferred.

IOTYPE=P would be used when a translator requires a partitioned data set. If a translator accepted a sequential data set, IOTYPE=O would be used.

SCLM determines the names of the members to be copied into the project hierarchy from the architecture definition being built or from the DFLTMEM parameter on the FLMALLOC macro. If an architecture definition member is being built, the name specified in that member is used. If a source member is being built directly or as a result of an INCLD architecture statement, the DFLTMEM parameter on the FLMALLOC macro is used.

If an asterisk is specified for the output member name in the architecture definition, or no DFLTMEM parameter is specified, then all members in the temporary data set are copied into the project hierarchy. Otherwise, only the member that matches the name on the architecture statement or DFLTMEM parameter is copied into the project hierarchy.

For example, Figure 209 on page 498 does not contain an example of IOTYPE=P. However, if the Assembler, ASMA90, had required a partitioned data set for the object module instead of a sequential data set, then the FLMALLOC for the SYSLIN DDNAME would have used IOTYPE=P instead of IOTYPE=O.

IOTYPE=S

Allocate a temporary sequential data set and create the input stream for the translator by concatenating the contents of all the members that are SINCed as well as any text specified via CMD cards. Concatenation will occur in the order specified by the architecture definition. Use the KEYREF parameter to identify the members from the project hierarchy that will be used to create the input stream.

When these criteria are met, SCLM allocates the PDS member directly from the SCLM-controlled library, rather than copying the member first to a sequential data set. The criteria are:

- There is only one input
- The input is from a SINC statement
- The KEYREF on the FLMALLOC statement is SINC
- you are not doing input list processing.

Any user-defined translators must take into account that the DDNAME allocated might be either a sequential data set or a PDS member.

For example, in Figure 209 on page 498, IOTYPE=S is used to allocate the data set that will contain the input stream for the translator SYSIN. See the example for IOTYPE=I for an illustration of what is contained in architecture definition member FLM01CMD. If this member were built at DEV1, the SYSIN data set would contain a copy of member FLM01MD1, type SOURCE. If more than one SINC card had been specified, then the source referenced by subsequent SINC cards would have been appended to the end of SYSIN in the order specified in the architecture definition.

IOTYPE=U

Any preallocated ddname that matches the DDNAME parameter value will be used. There will be no new ddname allocation. This is typically used for referring back to a preallocated ddname from a previous FLMALLOC following a previous FLMTRNSL in the same language definition. In this situation the DDNAME parameter values need to be the same.

ddname substitution lists are useful in situations in which more than one translator is defined for a language and one translator needs to use the output from a previous translator. This latter translator would have an FLMALLOC statement with IOTYPE=U and the same DDNAME parameter value as the previous FLMALLOC for a previous FLMTRNSL in the same language definition. In order to use ddname substitution lists the translator must be programmed to handle the ddname substitution list and the FLMTRNSL must have a PORDER value of 2 or 3 to construct and pass the list to the translator.

Translators that are programmed to use ddname substitution lists include some compilers, linkage editors, and utilities. These translators will use the DDNAME parameter value for a data set. If the

DDNAME parameter is not specified the system will generate a ddname for use in the ddname substitution list.

For PORDER values of 0 or 1 SCLM does nothing. There are no additional file allocations. At execution time the translator will use whatever data set has been allocated to the ddname specified by the translator program.

For example, Figure 209 on page 498 does not use IOTYPE=U. The sample language definition for the assembler language only calls one build translator. However, if this language definition had called a preprocessor and had PORDER=2 or 3, as shown in Figure 210 on page 503, the assembler compiler, ASMA90, would want to use the output from the preprocessor, IFPRE0. It would not be necessary for ASMA90 to create a data set that would contain the input stream because this has been prepared by IFPRE0.

```
*
* BUILD TRANSLATOR(S)
*
*
*    --CREATE THE INPUT STREAM FOR THE ASSEMBLER COMPILER--
*
*         --ASSEMBLER PREPROCESSOR--
          FLMTRNSL   CALLNAM='ASM PREPROCESSOR',                      C
               FUNCTN=BUILD,                                          C
               COMPILE=IFPRE0,                                        C
               PORDER=3,                                              C
               OPTIONS=(GROUP=@@FLMGRP,                               C
               TYPE=@@FLMTYP,                                         C
               MEMBER=@@FLMMBR)
*
* DDNAME ALLOCATIONS
*
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,                      C
               RECNUM=9000,DDNAME=SYSIN
* --CALL THE ASSEMBLER COMPILER TO PROCESS INPUT--
*
*         --ASSEMBLER INTERFACE--
          FLMTRNSL   CALLNAM='ASSEMBLER',                            C
               FUNCTN=BUILD,                                         C
               COMPILE=ASMA90,                                       C
               VERSION=1.0,                                          C
               GOODRC=0,                                             C
               PORDER=3,                                             C
               OPTIONS=(XREF(SHORT),LINECOUNT(75),OBJECT,RENT)
*
* DDNAME ALLOCATIONS
*
 FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,RECNUM=7500,DFLTTYP=OBJ
 FLMALLOC  IOTYPE=N
 FLMALLOC  IOTYPE=N
 FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
 * ADD ONE  FLMCPYLB FOR EACH FLMSYSLB
  FLMCPYLB SYS1.MACLIB
 FLMALLOC  IOTYPE=U,DDNAME=SYSIN
 FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,KEYREF=LIST,PRINT=Y,             C
  DFLTTYP =LIST,RECNUM=20000
 FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
  FLMCPYLB NULLFILE
 FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,RECNUM=15000
 FLMALLOC  IOTYPE=N
 FLMALLOC  IOTYPE=N
 FLMALLOC  IOTYPE=N
 FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
  FLMCPYLB NULLFILE
 *
```

Figure 210. Sample language definition that calls a preprocessor

IOTYPE=W

Allocate a temporary sequential data set for translator use. SCLM uses the RECFM, LRECL, and RECNUM parameters for allocation of this data set. If they are not specified, SCLM uses the defaults.

For example, Figure 209 on page 498 uses IOTYPE=W to allocate SYSUT1. When the Assembler, ASMA90, is invoked, a sequential data set is created and allocated to DDNAME SYSUT1. This data set

is used internally by the assembler and it is not necessary to store it in the project hierarchy. This language definition does not print the contents of this data set to the build listing data set because the PRINT keyword was not specified and defaults to N.

When all build translators for this language have completed processing, SYSUT1 will be deleted. In the preceding example, when the assembler has completed and returned control to build, build will deallocate the data set associated with SYSUT1.

The position of the FLMALLOC macros is very important because SCLM can pass ddnames directly to the translator; see the PORDER field description. SCLM passes ddnames to the translator in the order of the FLMALLOC macros.

SCLM deallocates temporary data sets after all translators for a particular member and a particular function (for example, FUNCTN=BUILD, COPY, and PURGE specified in the FLMTRNSL macro) for a particular language have completed processing.

To use the output from one translator step as input to another translator step, add (or modify) an FLMALLOC macro for the second translator step with IOTYPE=U and DDNAME=*ddname* allocated for the first translator step. Be careful when adding FLMALLOC macros for the second translator step. Depending on the PORDER that is specified in the FLMTRNSL macro, it may be necessary to put the new FLMALLOC macro in a particular position in the list of FLMALLOC macros. See the documentation for the particular compiler or translator you are calling to determine whether it accepts ddname substitution lists and, if so, what order it expects the parameters to be passed.

Table 32 on page 504 indicates the valid IOTYPEs for each function. Note that all IOTYPEs are valid for a build, and that IOTYPEs A, U, and W are valid for all functions.

Table 32. Valid IOTYPEs for each function

IOTYPE	Build	Copy	Parse	Purge	Verify
A	X	X	X	X	X
I	X				
L	X				
N	X				
O	X				
P	X				
S	X				
U	X	X	X	X	X
W	X	X	X	X	X

,BLKSIZE=*block_size*

Block size of the data set. This parameter is valid for IOTYPE=W, O, P, and S. If this parameter is not specified or is specified as 0 (zero), then the block size used is the largest integral multiple of the LRECL values that is less than or equal to 3120. It is recommended that this value match the block size of the target data set for IOTYPE=P and RECFM=U. This parameter is ignored for IOTYPE=A, I, L, N, and U.

The IBM linkage editor requires that the DCBS option parameter be passed in order for the SYSLMOD block size to be used in creating load modules. If the DCBS option is not specified, the linkage editor creates load modules using the maximum record size for the device type. Use the OPTIONS= parameter on the FLMTRNSL macro to pass the DCBS option. Failure to do so can result in message FLM44507.

,CATLG=N|Y
Indicates whether a data set is to be cataloged. Valid for IOTYPE=W, O, P, and S. SCLM temporarily allocates cataloged data sets with a predefined high-level qualifier, the TSO-prefix. The data set is deleted after all translators complete their functions. The default is N.

,DDNAME=*ddname*
The ddname to be used for this allocation. If you do not specify a ddname for the allocation, SCLM generates one for you. If the PORDER parameter in FLMTRNSL has the value of 0 or 1, a nonblank value is required for DDNAME.

A special case occurs when MALLOC=Y is specified. Because MALLOC=Y implies more than one allocation, you must allow SCLM to generate ddnames for these allocations. If PORDER=2 or 3, SCLM generates a ddname if the parameter is omitted. This parameter is not used for IOTYPE=L or IOTYPE=N.

Do not reuse the same ddname in multiple-step language definitions unless you intend to pass data from one step to the next using IOTYPE=U. If the same DDNAME is used for multiple translators, only the outputs from the last translator will be copied to the hierarchy.

,DINIT=N|Y
Indicates whether SCLM should create a member in a temporary data set allocated with IOTYPE=P. DINIT is ignored for all IOTYPEs except P. If DINIT=Y, SCLM initializes the member with a single record containing the string "DUMMY FILE" beginning in column 1. The member created will have the same name as the build map that is created if the translator is successful. If the MEMBER parameter is specified, its value will be used to determine the name of the member to initialize. If the MEMBER parameter is not specified, the member initialized will be the member to be saved in the hierarchy. If the member will not be saved in the hierarchy, the member initialized will have the same name as the source or architecture definition controlling the build.

,DIRBLKS=*directory_blocks*
The number of directory blocks allocated to the data set if the data set is partitioned (IOTYPE=P). For IOTYPE=P, the default is 1 and for all other IOTYPEs, the default is 0. SCLM will ignore nonzero values for all IOTYPES except IOTYPE=P.

,DISP= OLD|SHR|MOD|NEW
Optional parameter used to identify the disposition for the allocation on a DD card in JCL. Valid values are OLD, SHR, MOD, NEW. If not specified, the disposition defaults to an appropriate value for the IOTYPE parameter, as described in Table 33 on page 505.

Table 33. Valid DISP values for IOTYPE values

IOTYPE	Default	Valid Values	Condition
A	SHR	SHR,MOD,OLD	With a single copylib
A	SHR	SHR	With multiple copylibs
A	SHR	SHR,OLD	With MALLOC=Y
H	n/a	n/a	
I	SHR	SHR	
L	n/a	n/a	
N	n/a	n/a	
O	OLD	NEW,MOD	
O	SHR	SHR,OLD	With MALLOC=Y
P	NEW	NEW	
S	MOD	MOD	
U	n/a	n/a	

Table 33. Valid DISP values for IOTYPE values (continued)

IOTYPE	Default	Valid Values	Condition
W	NEW	NEW,MOD	

The DISP parameter applies to the temporary data set created by SCLM instead of the controlled members in the hierarchy for IOTYPE O and P. See the *z/OS TSO/E Command Reference* for more information.

This parameter is ignored for IOTYPE L, N, and U.

,DFLTMEM=*default_member*
Indicates the name for the output member. Use IOTYPE=O or P to allocate data sets that will be used for translator outputs. If this parameter is not used, the output member name for IOTYPE=O will be the same as the source member; for IOTYPE=P all output members will be copied using the translator-generated member names. SCLM ignores this field during a build if you use an architecture definition member to build the source member. If you are using an architecture definition member, define the translator outputs with an output keyword such as OBJ, OUTx, or LOAD. DFLTMEM is ignored unless:

1. KEYREF is specified with a valid output keyword.
2. DFLTTYP is specified.
3. IOTYPE is either O or P.

The name of the translator output can be based on the name of the source input by using an asterisk as a special match character. The asterisk is replaced by the name of the source member. If the substitution of the source name would result in a name longer than 8 characters, the source name is truncated to produce an 8-character name. For example, if the DFLTMEM parameter is *FM, a source member of EX00G would cause the output to be stored in name EX00GFM.

,DFLTTYP=*default_type*
Indicates the name of the SCLM type for translator outputs. Use IOTYPE=O or P to allocate data sets that will be used for translator outputs. The output member name is the same as the source member. SCLM ignores this field during a build if you use an architecture member to build the source member. If you are using an architecture member, define translator outputs with an output keyword such as OBJ, OUTx, or LOAD. DFLTTYP is ignored if no KEYREF is specified.

The type for the translator output can be based on the type of the source input by using an asterisk as a special match character. The asterisk is replaced by the type of the source member. If the substitution of the source type would result in a name longer than 8 characters, the source type is truncated to produce an 8-character result. If the DFLTTYP parameter is *LST, a source type of SRC1 would cause the output to be stored in type SRC1LST. The type specified on this parameter, or the type generated if an asterisk is used, must be defined to the project definition with the FLMTYPE macro. No verification of this parameter is performed when the project definition is generated.

,DSNTYPE=PDS|LIBRARY
Determines whether a temporary partitioned data set (IOTYPE=P) is allocated as a PDS or PDSE. Use DSNTYPE=LIBRARY to have the data set allocated as a PDSE. This parameter is only valid for IOTYPE=P. The default value is PDS.

,INCLS=FLMINCLS_name
Refers to an FLMINCLS macro in the language definition that lists the types to be allocated. If the FLMLANGL macro for the language has ALCSYSLB=Y, the FLMSYSLB data sets for the include set will be allocated after the data sets from the project. This parameter is only valid for IOTYPE=I. If no INCLS= parameter is specified for IOTYPE=I, the default include set is used to determine the types for allocation.

,KEYREF=*keyword_reference*
Refers to a keyword in the build map or architecture definition. The member name and type (as denoted in the build map or architecture definition) associated with the keyword are used by other parameters in this macro:

- If IOTYPE=L, *keyword_reference* identifies the member name the macro passes in the ddname substitution list for the translator.
- If IOTYPE=S, *keyword_reference* identifies the input members for the translator. For LEC architecture members, the contents of the temporary data set will depend on the KEYREF specified. If KEYREF is specified as INCL, an include statement in a format used by the S/370 linkage editor will be generated for each object member or load module referenced. If KEYREF is specified as SINC, the contents of each object member referenced or generated by SCLM will be copied into the temporary data set. This does not include any object modules referenced by the SCLM LINK command. LINK will cause an "INCLUDE" link-editor command to be included in the temporary data set. S/370 linkage editor include statements are generated for each load module specified as input. This is true when KEYREF=SINC or KEYREF=INCL. Although it will take longer to process KEYREF=SINC, this can be used to handle object members having large block sizes or containing linkage edit control statements.
- If IOTYPE=I, *keyword_reference* determines the type name of the hierarchy to allocate. The keywords that can be used with IOTYPE=I are SINC, SREF, and CREF.
- If IOTYPE=O or P, *keyword_reference* identifies the location in the hierarchy for build to copy the output created by the translator if the translator is successful. The keywords that can be used with IOTYPE=O or P are COMP, LIST, LMAP, LOAD, OBJ, and, OUTx.

,LANG=*language*
Allows a build output to be assigned a different language than the build input. If this parameter is not specified, then build outputs are assigned the same language as inputs.

This parameter is not necessary when you can create in a single build all the build outputs you want.

Use this parameter when you want the build output of one language definition to be verified, built, copied, or purged in another language definition.

,LRECL=*record_length*
Logical record length of the data set (numeric). It is valid for IOTYPE=W, O, P, and S. The default is 80. It is recommended that this value match the LRECL of the target data sets for IOTYPE=O or P.

,MALLOC=N|Y
Use MALLOC=Y when the translator generates a sequential output data set that has a specific data set name and cannot be allocated to a ddname before the translator is invoked. This condition might occur if the translator performs its own allocations and always creates a data set with a specific name. Input list translators are required to generate output data sets that can be captured with this type of allocation. For input list processing, one allocation is performed for each member processed on the input list.

When you specify the FLMALLOC macro with MALLOC=Y, you must also specify an FLMCPYLB macro that identifies the name of the data set to be allocated.

An FLMALLOC with MALLOC=Y is ignored for all iotypes except O and A. If MALLOC=Y and IOTYPE is not an A and not an O, an error message is produced. The KEYREF parameter must be specified on the FLMALLOC for the allocation to occur. If MALLOC=Y is specified, the ddname parameter must be blank.

,ALLCDEL=N|Y
Indicates that all data sets referenced by this FLMALLOC macro should be deleted when SCLM has finished processing them. For example, specify ALLCDEL=Y to indicate that the output listings from the Input List translator should be deleted after they are copied into the hierarchy. The ALLCDEL parameter is ignored unless MALLOC=Y is specified.

,MEMBER=*member_name*
Causes a ddname to be allocated to a member of a temporary partitioned data set created by SCLM. This parameter is valid only for IOTYPE=P.

The member name can be evaluated dynamically by specifying @@FLMONM or @@FLMMBR as the parameter value. If a KEYREF OUTx parameter is specified and the architecture definition has a matching OUTx statement, then SCLM uses the output member name in the architecture definition. If

no OUTx architecture statement is specified, then SCLM uses the name of the member being built. This can be the name of an architecture definition or the name of a build input.

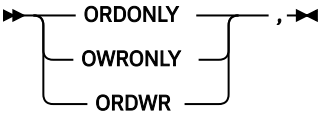
This parameter is not necessary for most translators. However, some translators must know the name of the output member.

,NOSAVRC=*no_save_rc*
A return code value set by a translator that indicates whether SCLM is to store a translator output in this data set. This parameter is valid for IOTYPE=O and P. SCLM provides this feature to handle translators that, by design, have missing or static outputs. If it is decided that these outputs need not be saved for some situations, then the translators can be written to recognize these situations and return an appropriate return code. Through the use of this return code and the NOSAVRC parameter, SCLM will be able to determine when the output should be saved in the hierarchy and when it should not. This helps avoid unnecessary rebuilds of some build components. This parameter, if specified, must have a nonzero positive value; if not specified, the default is zero.

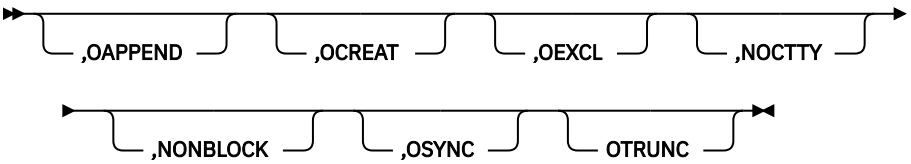
Note: An example is a translator that can differentiate "comment only" changes from code changes and determine which outputs are not affected. A listing is updated but not OBJECT code. SCLM can use this information to avoid unnecessary work.

,PATHOPT=*uss_path_options*
Specify a list of access and status options for a z/OS UNIX file allocation. Up to seven option keywords, separated by commas, can be specified. Only one keyword from the access group can be specified.

Access keywords are:



Status keywords are:



See the PATHOPTS parameter in *z/OS MVS JCL Reference* for more details.

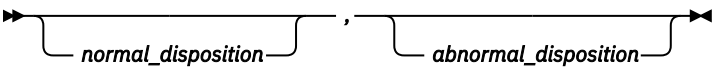
,PATHMDE=*uss_path_mode*
Specify file access attributes when creating a z/OS UNIX file. Up to 14 keywords, separated by commas, can be specified.

Keywords =

```
SIRUSR SIWUSR SIXUSR SIRWXU SIRGRP SIWGRP SIXGRP
SIRWXG SIROTH SIWOTH SIXOTH SIRWXC SISUID SISGID
```

See the PATHMODE parameter in *z/OS MVS JCL Reference* for more details.

,PATHDSP=*uss_path_disposition*
Specify file disposition when allocating a z/OS UNIX file. The syntax is:



where either disposition can be KEEP or DELETE.

See the PATHDISP parameter in *z/OS MVS JCL Reference* for more details.

,FILEDAT=*uss_file_data*
Specify the organisation of a z/OS UNIX file. Either TEXT or BINARY can be specified. BINARY indicates a byte-stream data with no record delimiters. TEXT indicates that the data contains records delimited by the EBCDIC newline character (X'15').

See the FILEDATA parameter in *z/OS MVS JCL Reference* for more details.

,PRINT=N|Y|I
Indicates whether the contents of a sequential data set are to be copied to the SCLM listings data set (userid.BUILD.LISTxx). The contents will only be copied in the case of an error when the *error listings only* field is selected on the build panel. This parameter is only valid for data sets allocated with IOTYPE=W, S, or O. The valid values are:

N
indicates the contents of the temporary sequential data set are *not* to be copied to the listings data set. This is the default setting.

Y
indicates that the contents of the temporary sequential data set are to be copied to the listings data set.

I
indicates that the contents of the temporary sequential data set are to be copied to the listings data set. SCLM will open and close the temporary data set before invoking the translator.

Data sets allocated with PRINT=Y must be opened by the translator. Otherwise, an ABEND can occur when SCLM attempts to copy the contents to the build listings data set. For data sets that will not be opened by the translator, use PRINT=I. As PRINT=I adds an open and close, build performance can be slightly degraded.

,RECFM=*record_format*
Record format of the data set. It is valid for IOTYPE=W, O, P, and S. Valid values are F, FA, FM, FB, FBA, FBM, V, VA, VM, VB, VBA, VBM, and U; the default is FB (fixed blocks). It is recommended that this value match the RECFM of the target data sets for IOTYPE=O or P.

,RECNUM=*number_of_records*
Number of records to be allocated (numeric). It is valid for IOTYPE=W, O, P, and S. The default is 500.

This parameter is used in the calculation of the primary and secondary space allocations required for the temporary data set. Space allocations are in blocks and the number of blocks is determined by the number of records using this formula:

```
(((number_of_records / ((3120 / record_length) + 1)) + 1) / 16) + 1
```

,VIO=Y|N
Overrides the selection for use of VIO. Y causes the data set to always be allocated using VIO; N causes the allocation to never use VIO. The default is to determine use of VIO by comparing the RECNUM specification to the value for MAXVIO on the FLMCNTRL macro.

Note: The Automatic Class Selection (ACS) routines defined for a DFSMS installation can override the selection requested by SCLM. Contact your site's system programmer for information about how these will interact on your system.

,ENCODE=Y|N
Specifies whether a member is encoded when it is saved. The parameter ENCODE=Y can only be specified if IOTYPE=O or P has been specified on the FLMALLOC macro. SCLM, when copying the temporary data set and member to the SCLM controlled library, encodes the member when it is saved.

Defining a software component using the FLMALLOC macro

You can specify a software component either with an architecture member or with the FLMALLOC macros you specified in the language definition. For example, the language definition for member xxxxxxxx in type SOURCE contains these FLMALLOC macros:

```
FLMALLOC IOTYPE=S,KEYREF=SINC
FLMALLOC IOTYPE=O,KEYREF=LIST,DFLTTYP=LISTING
FLMALLOC IOTYPE=O,KEYREF=OBJ,DFLTTYP=OBJECT
```

Building the member is the same as building this architecture definition:

```
SINC xxxxxxxx SOURCE
LIST xxxxxxxx LISTING
OBJ  xxxxxxxx OBJECT
```

Always use the SINC keyword (on the KEYREF= parameter of the FLMALLOC macro) to identify the input member. If you need multiple SINC keywords, you must use an architecture member to specify the software component. Options to override the translator options (using the PARM and PARMx keywords) also require that you use an architecture member. You can also use the fields DFLTCRF and DFLTSRF on the FLMLANGL macro to identify the types to use in resolving source dependencies.

Example 1

Two data sets are allocated: one to contain the input stream (IOTYPE=S), the other to contain the output from the translator (IOTYPE=O). The input stream is the member you specify on the SINC statement of an architecture member. The output is copied to the member specified with the LIST statement of an architecture member. The output is also copied to the listing data set for the SCLM function.

```
 FLMALLOC IOTYPE=S,KEYREF=SINC,RECNUM=5000,LRECL=80,RECFM=FB

 FLMALLOC IOTYPE=O,KEYREF=LIST,RECNUM=5000,LRECL=133,RECFM=VBA,  X
     PRINT=Y
```

Example 2

The hierarchy for the type specified on the SINC statement of an architecture member is allocated. Two additional data sets are allocated after the hierarchy by the FLMCPYLB macro.

```
  FLMALLOC IOTYPE=I,KEYREF=SINC
  FLMCPYLB SYS1.LINKLIB
  FLMCPYLB SYS1.MACLIB
```

Example 3

The temporary partitioned data set (IOTYPE=P) that will contain the translator output to be saved into the project hierarchy will be allocated as a PDSE.

```
FLMALLOC  IOTYPE=P,KEYREF=LOAD,RECFM=U,LRECL=0,                X
   BLKSIZE=6144,RECNUM=5000,DIRBLKS=200,DDNAME=SYSLMOD,        X
   DSNTYPE=LIBRARY
```

FLMALTC macro

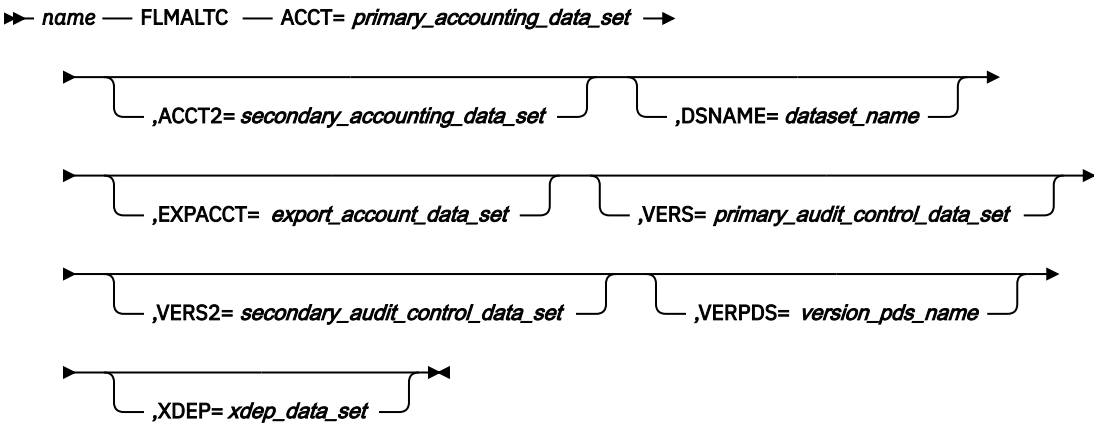
With this macro, you can specify control information that is different from that specified by FLMCNTRL. You can specify different VSAM databases or flexible data set naming conventions to associate with a group.

When the ALTC parameter of the FLMGROUP macro matches the name of the FLMALTC macro, only the control information for the VSAM databases and data set naming conventions defined in the FLMALTC macro are used for that group.

The FLMALTC macro values override the ACCT, ACCT2, DSNAME, EXPACCT, VERS, VERS2, and VERPDS values from the FLMCNTRL macro. The FLMALTC macro does *not* use these values from the FLMCNTRL macro so you must specify all the parameters you want on the FLMALTC macro statement. Any values not available to the FLMALTC macro are taken from the FLMCNTRL macro.

Any number of FLMGROUP macros can reference a single FLMALTC macro. SCLM issues a warning if an FLMALTC macro is defined that is not referenced by any FLMGROUP macro.

Macro format



Parameters

name
A unique 8-character name used to identify the control information defined by the FLMALTC macro. The name must be used in conjunction with the ALTC parameter of an FLMGROUP macro to indicate which set of information should be used for that group.

ACCT=*primary_accounting_data_set*
The name of the primary accounting data set to be used by any group referencing this FLMALTC macro. The data set you specify must be the name of the VSAM cluster you want to use. No SCLM variables can be used for this parameter.

,ACCT2=*secondary_accounting_data_set*
The name of the secondary accounting data set to be used by any group referencing this FLMALTC macro. Allocate this secondary VSAM data set following the same criteria as the primary accounting data set. Choose a unique name for this data set. It should reside on a different volume than the primary one. If a severe problem occurs with the primary data set (for example, a head crash on that disk), you can use this backup data set to restore the primary data set. The default is no secondary accounting data set.

Because additional accounting updates take place if you use this option, the updates will degrade performance. No SCLM variables can be used for this parameter.

,DSNAME=*dataset_name*
This parameter lets you specify the data set naming conventions for the partitioned data sets controlled by SCLM. The naming convention is specified as a pattern that can include a subset of the SCLM variables.

The only SCLM variables that can be used in the DSNAME parameter of FLMALTC are:

- @@FLMPRJ
- @@FLMGRP
- @@FLMTYP

The value specified in this parameter is used to resolve the SCLM variable @@FLMDSN. If this parameter is not specified, the data set name pattern defaults to

@@FLMPRJ.@@FLMGRP.@@FLMTYP. You can enter up to 44 characters for this parameter, including the SCLM variables and the periods.

If a data set name is specified, it must include the SCLM variable @@FLMTYP. It is also recommended that the variable @@FLMGRP be used in the data set name pattern. This helps prevent data from one group overwriting data in another group.

Attention: SCLM does not enforce or guarantee the uniqueness of partitioned data set names.

The variables can appear in any location within the DSNAME parameter. Any user-specified qualifiers can also be used. The preceding SCLM variables will be substituted with values that range from 1 to 8 characters. When determining the length of the final data set name, assume that the SCLM variables will contain values that are the maximum (8) number of characters.

Examples of data set name lengths are:

- APPL1.@@FLMGRP.@@FLMTYP is 5 + 1 + 8 + 1 + 8 = 23
- @@FLMPRJ.@@FLMGRP.@@FLMTYP.COMMON is 8 + 1 + 8 + 1 + 8 + 1 + 6 = 33

The data set name must meet all of the requirements specified by the MVS data set naming conventions. If the data set name is too long or it does not meet MVS data set naming conventions, errors occur during SCLM functions (for example, build or promote).

,EXPACCT=*export_account_data_set*
The name of the export accounting data set used by any group referencing this FLMALTC macro. The data set you specify must be the name of the VSAM cluster you want to use and must have a different name from any ACCT or ACCT2 parameter specified in FLMCNTRL or any FLMALTC macro. These variables can be used in specifying the name of the export accounting data set name:

- @@FLMPRJ
- @@FLMGRP
- @@FLMUID

,VERS=*primary_audit_control_data_set*
The name of the primary audit control data set to be used by any group referencing this FLMALTC macro. If you do not specify a VERS value, audit and versioning operations are not performed for the group. If you specify the VERS keyword and omit the primary_audit_control_data_set name, SCLM does not verify the name, and errors occur later during processing. If you do not specify a name, the value is blank.

,VERS2=*secondary_audit_control_data_set*
The name of the secondary audit control data set to be used by any group in the project referencing this FLMALTC macro. If you specify the VERS2 keyword and omit the secondary_audit_control_data_set name, SCLM does not verify the name, and errors occur later during processing. If you do not specify a name, the value is blank.

Because additional audit record updates occur if this option is used, be aware that overall performance will degrade. Do not specify VERS2 unless you have specified VERS. If you do, an error will occur when the project definition is assembled.

,VERPDS=*version_pds_name*
The name of the partitioned data set to contain the version data. These variables can be used when specifying the name of the partitioned data set: @@FLMPRJ, @@FLMGRP, @@FLMTYP, and @@FLMDSN. For example:

- VERPDS=@@FLMPRJ.@@FLMGRP.@@FLMTYP.VERSION
- VERPDS=@@FLMDSN.VERSION
- VERPDS=@@FLMPRJ.VERSION.@@FLMGRP

This parameter is optional. If you do not specify a value, the value @@FLMDSN.VERSION is assigned to the parameter (even if versioning is not active). See the description of the DSNAME parameter for more information about the value of @@FLMDSN.

If @@FLMDSN is used, it must be specified in the first 8 characters of the VERPDS= statement. For example, VERPDS=@@FLMDSN.VERSN12 is valid, but VERPDS=@@FLMPRJ.@@FLMDSN.VERSN12 is invalid. The VERPDS parameter on the FLMALTC macro can be used to override the version data partitioned data set for a specific group or set of groups.

You can have only one VERPDS data set per group and type at a time. However, you can respecify the VERPDS data set name to control the size of the version data sets. If the VERS=primary audit control data set name remains the same, a pointer to the VERPDS that holds a particular version allows you to retrieve and delete versions of members, even if you have changed the name of the VERPDS data set.

The FLMATVER macro must be used to enable versioning for particular groups. If you specify a value of 2 or more for the VERCOUNT parameter on the FLMCNTRL macro, you must specify a separate VERPDS for each combination of group and type that you intend to version.

Note: Failure to specify a separate VERPDS for each combination of group and type can cause retrieval problems.

XDEP=xdep_data_set

The name of the cross-dependency file to be used to by any group referencing this FLMALTC macro. The name you specify must be the name of the VSAM cluster you want to use. No SCLM variables can be used for this parameter.

If one FLMALTC macro supplies an XDEP data set name, then all FLMALTC macros must supply an XDEP data set name if an XDEP data set name is not specified on the FLMCNTRL macro.

Example

```
PROJXYZ   FLMABEG

          FLMCNTRL ACCT=PROJXYZ.ACCT.DATABASE

RELCNTL   FLMALTC  ACCT=PROJ2.ACCT.DATABASE,                          C
                DSNAME=RELEASE.PROJ2.@@FLMGRP.@@FLMTYP

DEVCNTL   FLMALTC  ACCT=PROJDEV.ACCT.DATABASE,                         C
                DSNAME=SWDEV.@@FLMPRJ.@@FLMGRP.@@FLMTYP

REL       FLMGROUP KEY=Y,ALTC=RELCNTL
INT       FLMGROUP KEY=Y,PROMOTE=REL
DEV       FLMGROUP KEY=Y,PROMOTE=INT,ALTC=DEVCNTL
```

The DEVCNTL FLMALTC macro defines an alternate accounting database and data set name to be used by the DEV group that references this macro. The PDS data sets associated with the DEV group have the naming convention

'SWDEV.PROJXYZ.DEV.type'.

The RELCNTL FLMALTC macro defines an accounting database and data set name to be used by the REL group that references this macro. The naming convention used for the PDS data sets associated with the REL group is

'RELEASE.PROJ2.REL.type'.

FLMATVER macro

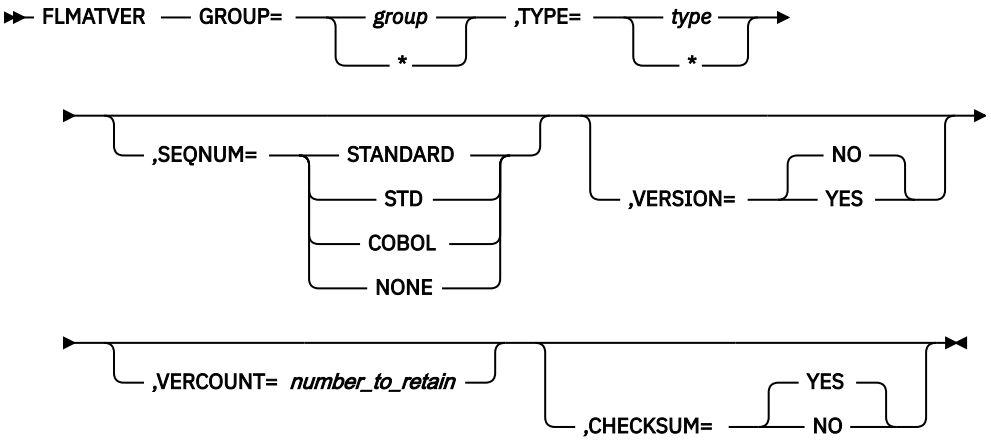
Use this macro to enable the audit and version utility and to define the group and the type of members in that group to record audit and version information for.

You must specify the name of the VSAM data sets to contain the audit information and the name of the partitioned data sets to contain the versions using the FLMCNTRL and FLMALTC macros. You can define multiple versioning partitioned data sets for a project.

Using the group and type defined in the FLMATVER macro, SCLM records information in the VSAM data set each time a member's accounting information is created, updated, or deleted within that SCLM group. This information is a record that contains the member's accounting information, the type of operation, the user ID of the user who performed the operation, and the date and time the operation occurred.

You can use the FLMATVER macro to store a version of a member. The member is stored when the particular SCLM operation (such as SAVE) has completed successfully. The version contains the information to recreate the member as it previously existed. You can disable the versioning function while maintaining the audit capabilities. Version information is captured each time an editable member or an output that is **not record format U** is created or updated, but not when it is deleted. Sequence number differences can be ignored by coding the SEQNUM parameter, otherwise, they are treated as data.

Macro format



Parameters

GROUP|*
The name of the group for which the audit data, version data, or both, is to be maintained. The group must be defined in the project. Use an asterisk (*) to indicate all groups.

,TYPE|*
The name of the type for which the audit data, version data, or both, is to be maintained. The type must be defined in the project. Use an asterisk (*) to indicate all types.

Audit information can be captured for editable or noneditable types. Version information can be captured for editable types and non-editable types that are not record format U. This means that you can maintain version information for types such as "source" and "object", but not for load modules or other data that has record format U. Therefore, if you have a project with record format U data, such as load modules, do *not* specify TYPE=* and VERSION=YES. If you attempt to version data that is record format U, an error message is issued during SCLM processing.

,SEQNUM=STANDARD|STD|COBOL|NONE
If you specify STANDARD, STD, or COBOL, SCLM ignores sequence number differences when creating a version of a member.

STANDARD or STD means ignore differences in the last eight columns of the data for fixed formats, and the first eight columns of the data for variable formats. In both cases the ignored columns are presumed to be standard sequence numbers.

COBOL means ignore differences in the first six columns of the data, which are presumed to be COBOL sequence numbers.

Omitting this parameter, or specifying NONE, indicates that all columns are to be treated as data.

Note: When changing the value of the SEQNUM specification for a project, also change the VERPDS specification on the FLMCNTRL or FLMALTC macros for the affected groups. Failure to do so may

cause checksum verification errors when attempting to recover versions created with the previous specification (see the CHECKSUM keyword).

,VERSION=YES|NO
If you specify YES, both the versioning and auditing processes are active. If you specify NO, versioning is *not* active; however, the audit process is active. If not specified, VERSION will default to NO. Version data can be captured for any editable or non-editable (output) members that are **not record format U**.

Note: You cannot have versioning without auditing.

,VERCOUNT=number_to_retain
The number of versions to keep in the version partitioned data set for the group or type specified. If you specify a value of zero (0), then all versions associated with a member are kept.

If you specify a value of two (2) or more, each time a member is changed the latest copy of the member is stored and the earliest copy is deleted, so that the number of versions remains constant. Any audit records that are associated with versions that have been deleted are retained, but no longer indicate that a version of the member exists. If you do specify a value of two or more, allocate a separate VERPDS for each combination of group and type that has versioning enabled.

Note: Failure to allocate a separate VERPDS for each combination of group and type can cause retrieval problems. Use the FLMALTC macro, or use the @@FLMGRP variable in the VERPDS name.

If a VERCOUNT value is not specified on the FLMATVER macro or if a value of one (1) is specified, then the value specified using the VERCOUNT parameter on the FLMCNTRL macro is used. If a VERCOUNT value is not specified on either macro, then all versions associated with a member are kept.

,CHECKSUM=YES|NO
If you specify YES or omit this parameter, checksum verification of versions on retrieval is in effect.

In the case of message FLM39220 Return Code 34, which indicates a damaged version or a version created before SEQNUM support was available in SCLM, you may take this action to override the checksum verification failure:

- Insert the CHECKSUM=NO parameter.
- Reassemble the project definition.
- Retry retrieval of the version.

Attention: The validity of the retrieved version is not assured. *This procedure is recommended for emergency use only.*

Example

The statements shown here illustrate how to capture versions of members as well as auditing information.

The first of these statements saves versions of members with TYPE=COBOL and GROUP=PROD, ignoring differences in columns 1-6 where COBOL sequence numbers are expected. The second statement saves versions of your COPYBOOK members, including the non-editable members that might have been generated as a result of building a BMS member. The third statement tells SCLM to keep only the latest two versions of your object modules. This overrides any VERCOUNT specified on the FLMCNTRL macro for the project.

```
FLMATVER GROUP=PROD,TYPE=COBOL,VERSION=YES,SEQNUM=COBOL
FLMATVER GROUP=PROD,TYPE=COPYBOOK,VERSION=YES
FLMATVER GROUP=PROD,TYPE=OBJ,VERSION=YES,VERCOUNT=2
```

Note: If sequence number differences are to be ignored, full length source lines are saved in the delta file for all lines with non-sequence number differences, but lines that differ only in the sequence number are not saved in the delta file. So, when the version is retrieved, the original sequence numbers for unchanged lines are lost. Instead, the sequence numbers for the most current version are retained.

This statement saves only the auditing information for members with TYPE=PASCAL and GROUP=PROD:

```
FLMATVER GROUP=PROD,TYPE=PASCAL,VERSION=NO
```

Note: You can omit the VERSION=NO parameter as it is the default. If omitted, versions of the member will not be saved.

The order of the FLMATVER macros is important to keep in mind. Versioning is enabled/disabled in the order specified, so after turning versioning off for GROUP=* or TYPE=*, any later FLMATVER macros that specify a particular GROUP or TYPE will be ignored.

In this example, no version will be saved for PROJ.AAA.SOURCE.

```
    FLMATVER GROUP=*,TYPE=*,VERSION=NO
    FLMATVER GROUP=AAA,TYPE=SOURCE,VERSION=YES
```

However, if the two statements are reversed, a version of PROJ.AAA.SOURCE will be saved.

```
    FLMATVER GROUP=AAA,TYPE=SOURCE,VERSION=YES
    FLMATVER GROUP=*,TYPE=*,VERSION=NO
```

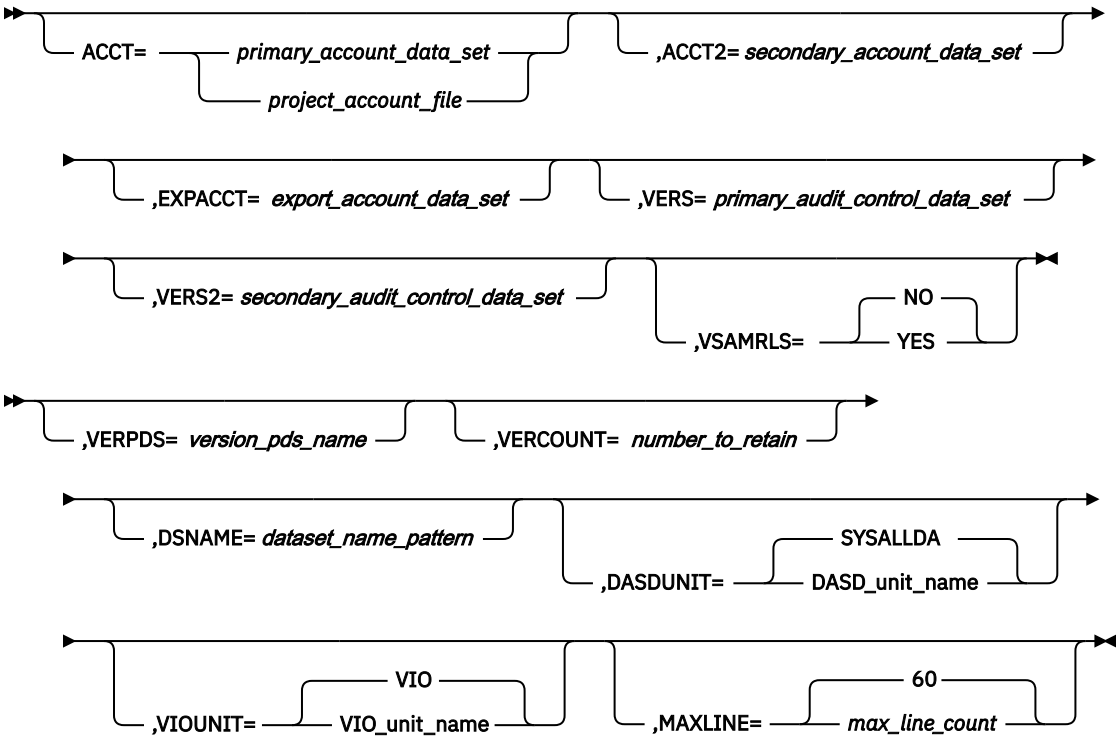
FLMCNTRL macro

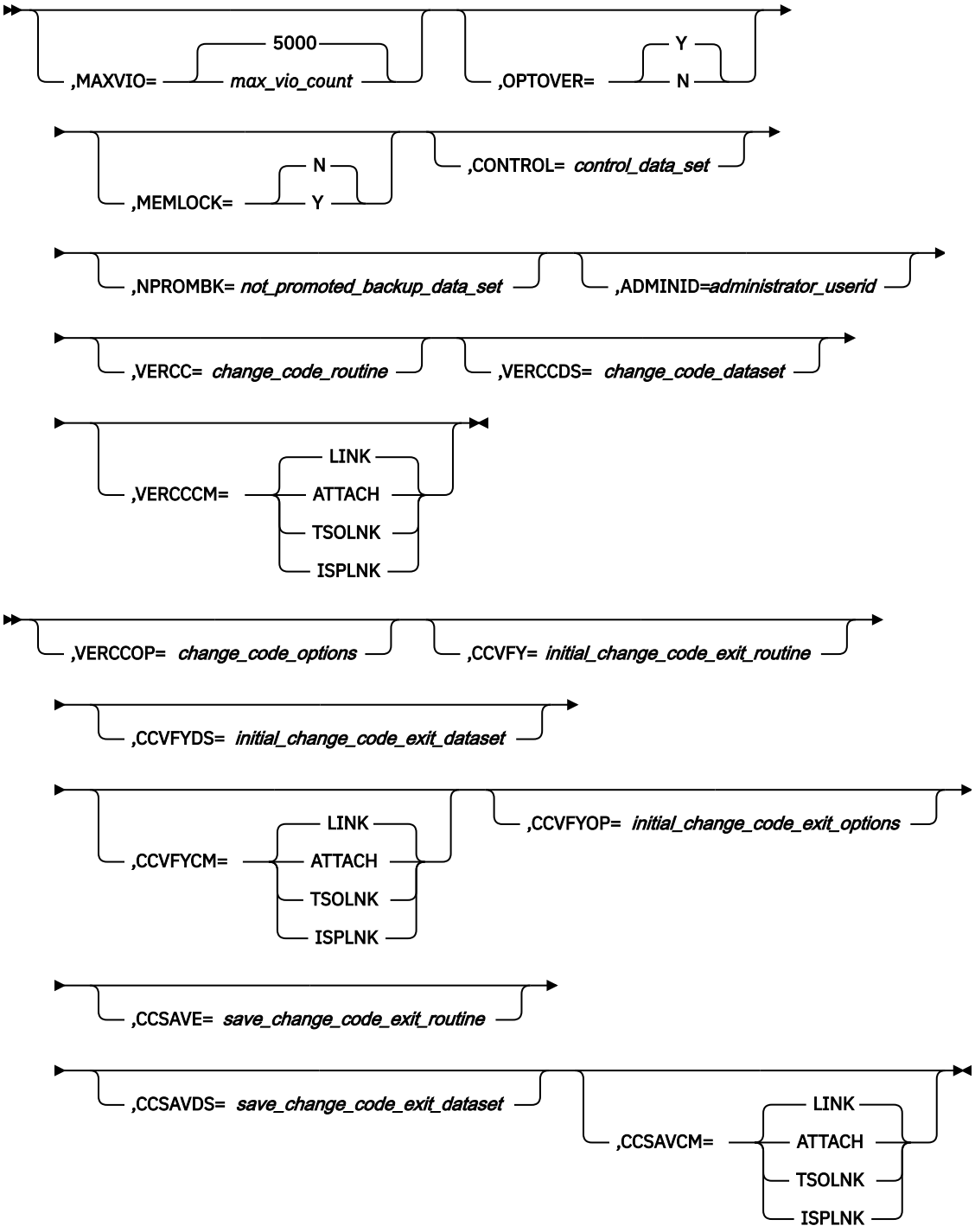
Use this macro to specify project-specific control options. This macro can appear only once in any project definition.

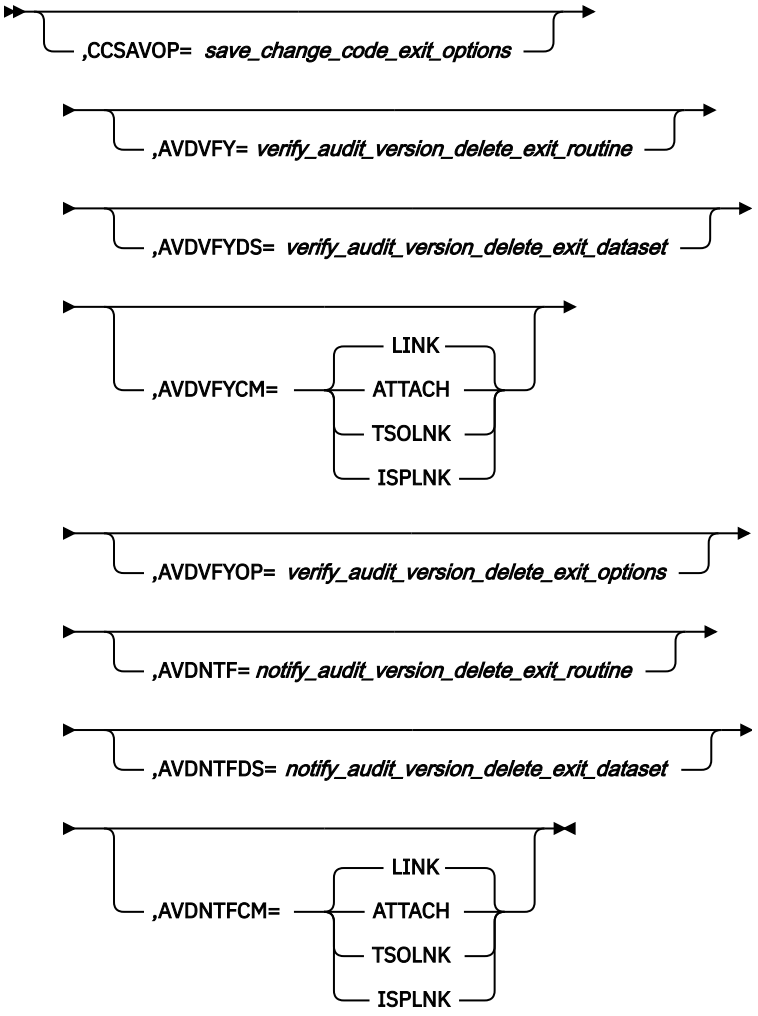
If FLMCNTRL is not specified, it defaults to this definition for any group that does not have internal data sets explicitly defined through the FLMALTC macro:

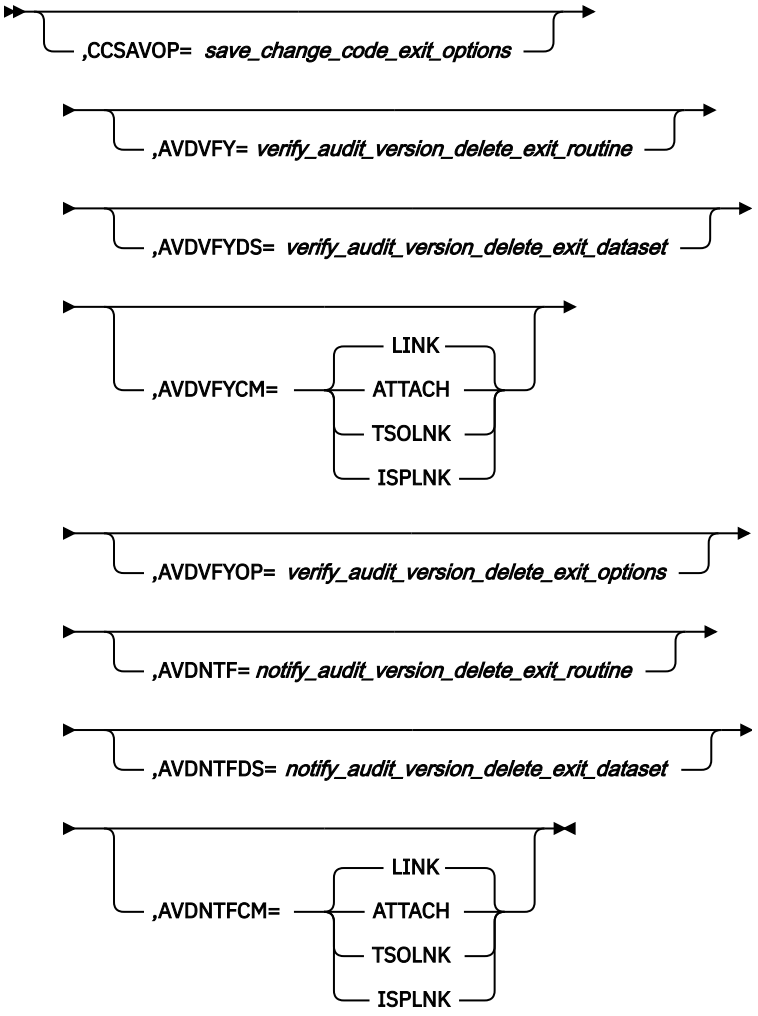
```
    FLMCNTRL ACCT=project.ACCOUNT.FILE
```

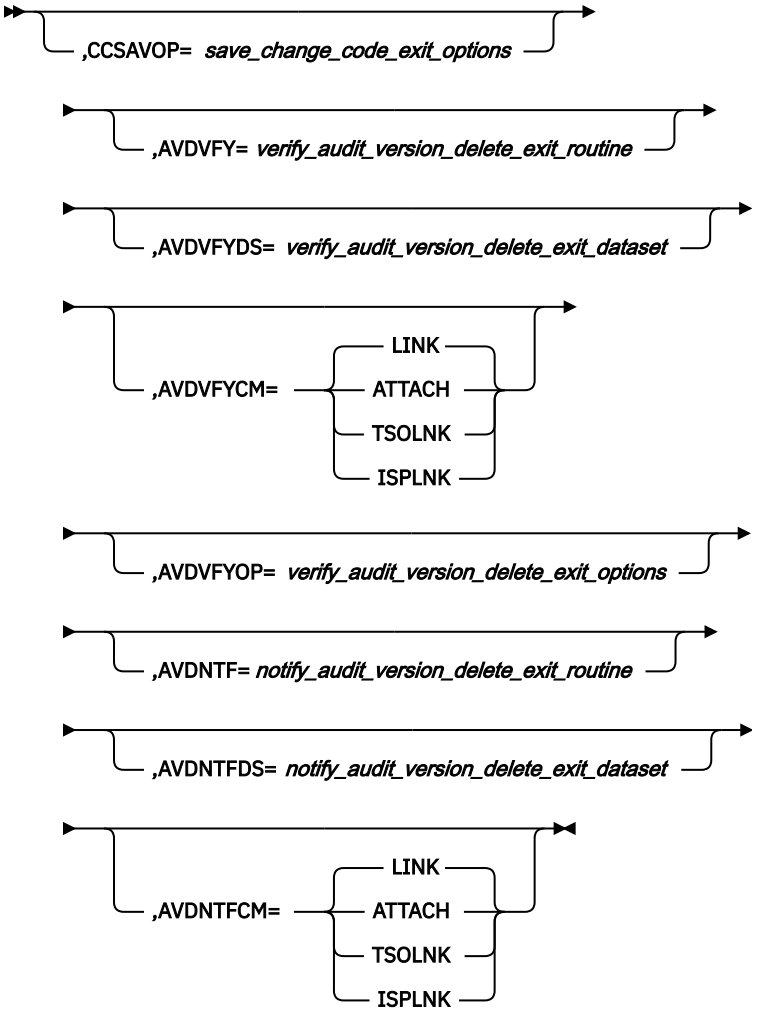
Macro format

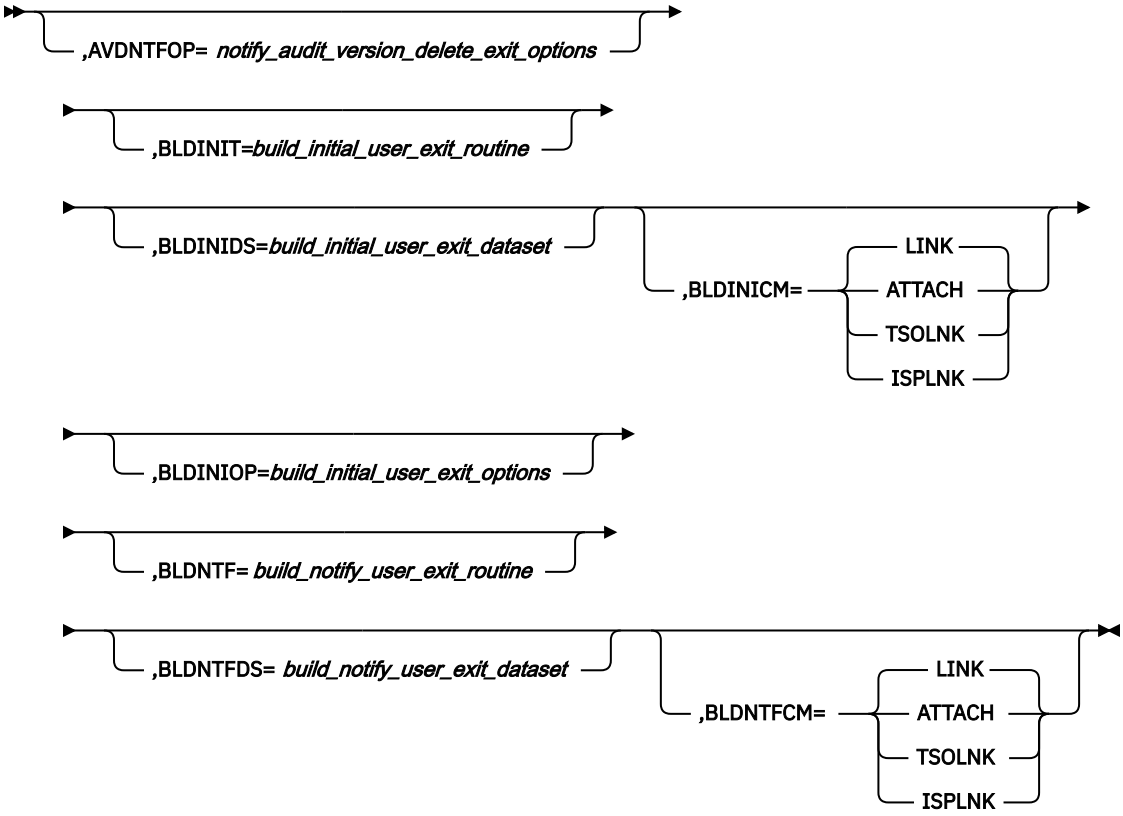


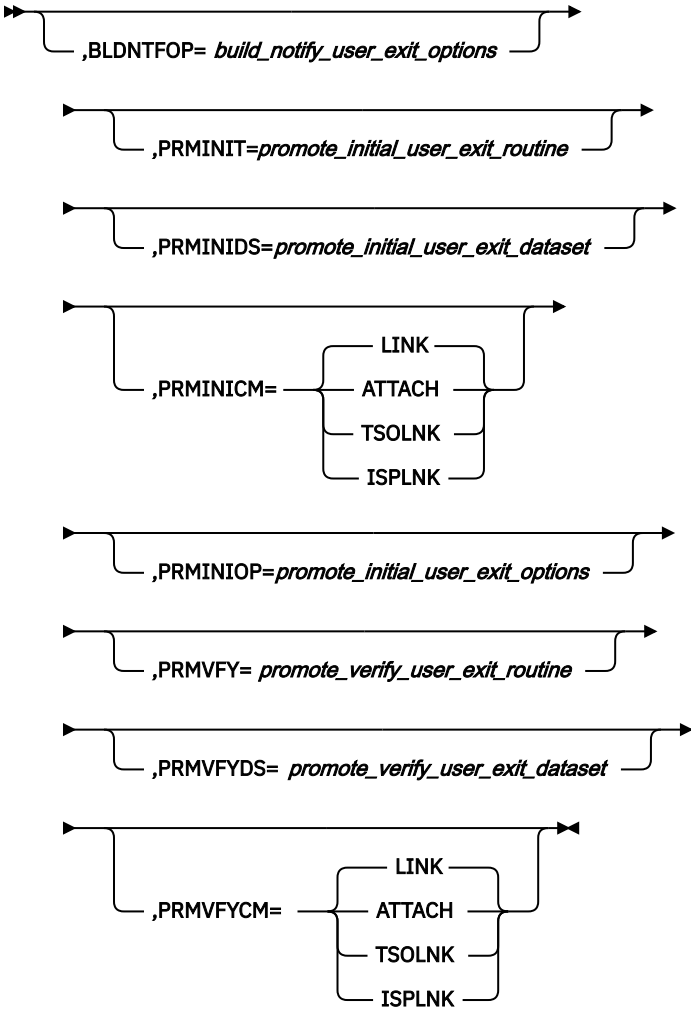

,MAXVIO=
5000
max_vio_count
,OPTOVER=
Y
N
,MEMLOCK=
N
Y
,CONTROL= control_data_set
,NPROMBK= not_promoted_backup_data_set
,ADMINID=administrator_userid
,VERCC= change_code_routine
,VERCCDS= change_code_dataset
,VERCCCM=
LINK
ATTACH
TSOLNK
ISPLNK
,VERCCOP= change_code_options
,CCVFY= initial_change_code_exit_routine
,CCVFYDS= initial_change_code_exit_dataset
,CCVFYCM=
LINK
ATTACH
TSOLNK
ISPLNK
,CCVFYOP= initial_change_code_exit_options
,CCSAVE= save_change_code_exit_routine
,CCSAVDS= save_change_code_exit_dataset
,CCSAVCM=
LINK
ATTACH
TSOLNK
ISPLNK

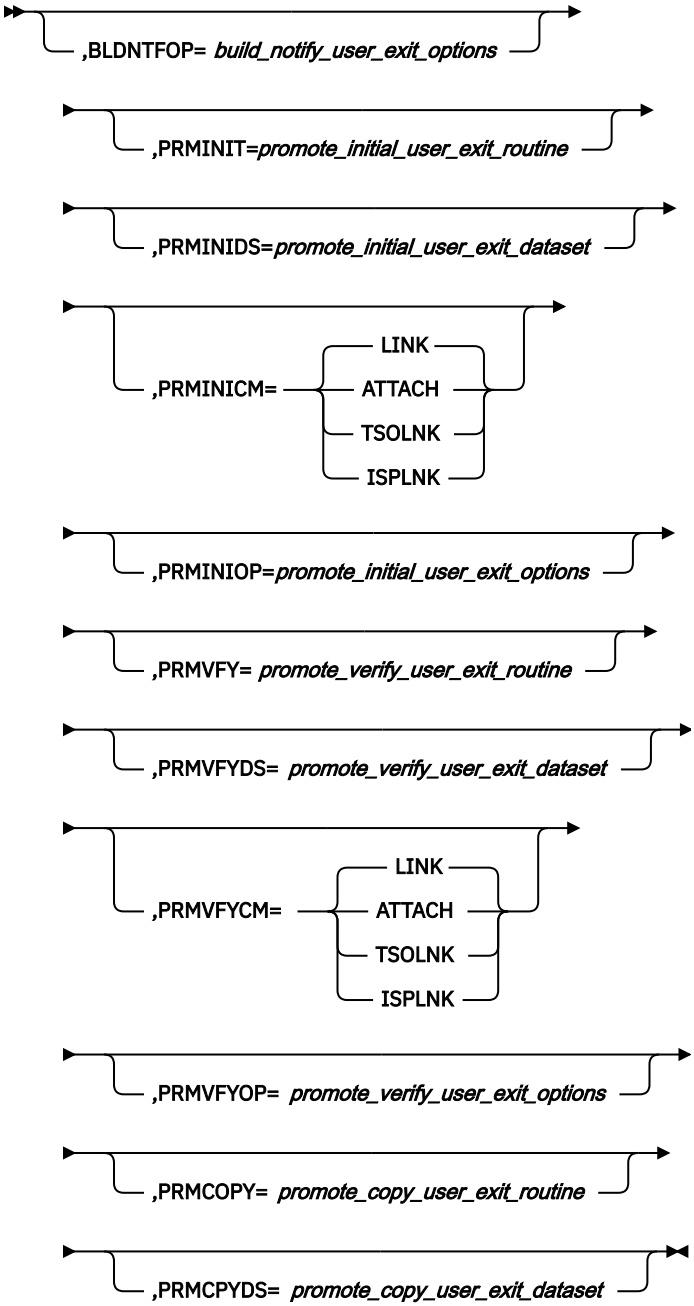

,CCSAVOP= save_change_code_exit_options
,AVDVFY= verify_audit_version_delete_exit_routine
,AVDVFYDS= verify_audit_version_delete_exit_dataset
,AVDVFYCM=
LINK
ATTACH
TSOLNK
ISPLNK
,AVDVFYOP= verify_audit_version_delete_exit_options
,AVDNTF= notify_audit_version_delete_exit_routine
,AVDNTFDS= notify_audit_version_delete_exit_dataset
,AVDNTFCM=
LINK
ATTACH
TSOLNK
ISPLNK

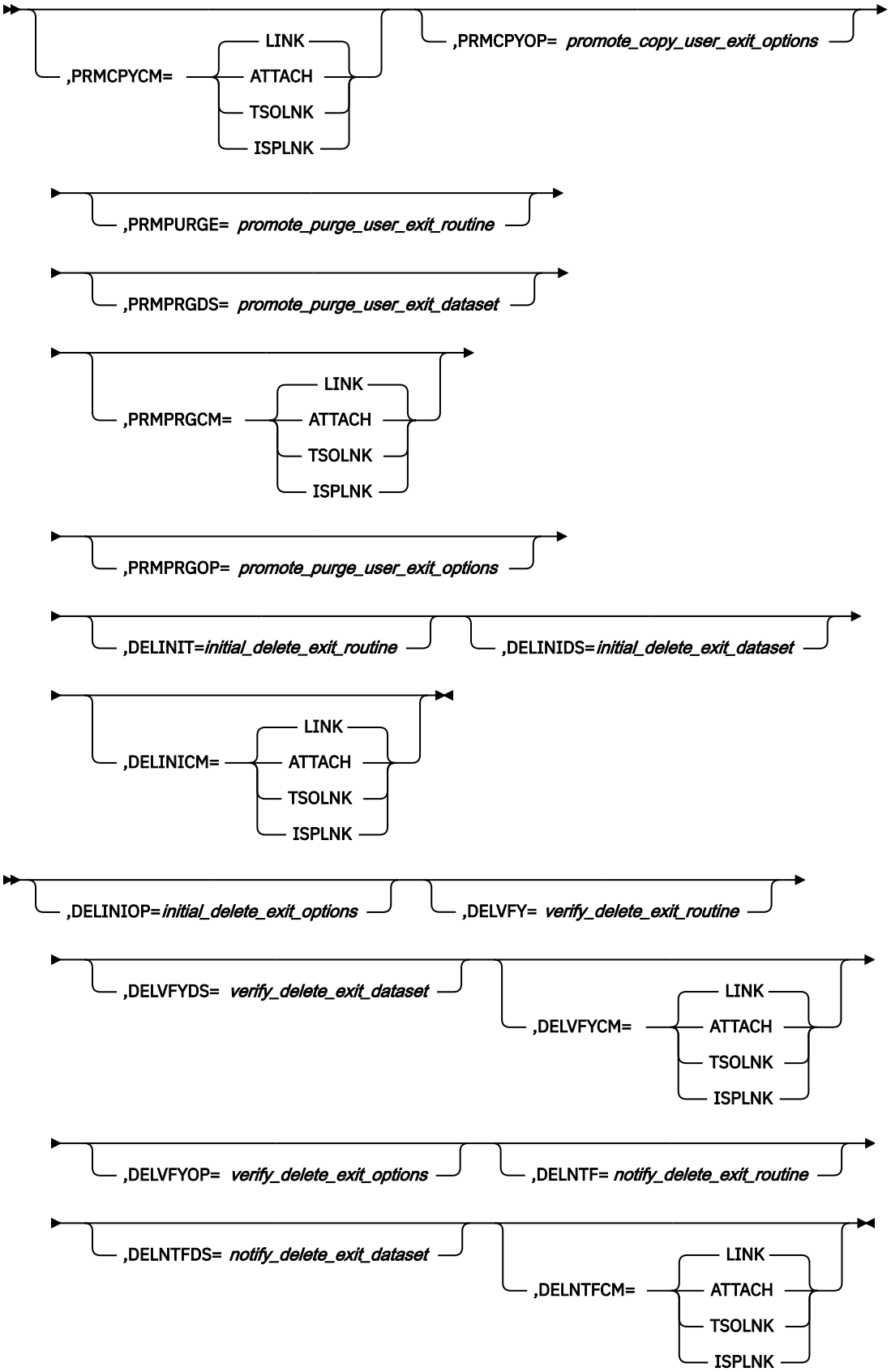

,CCSAVOP= save_change_code_exit_options
,AVDVFY= verify_audit_version_delete_exit_routine
,AVDVFYDS= verify_audit_version_delete_exit_dataset
,AVDVFYCM=
LINK
ATTACH
TSOLNK
ISPLNK
,AVDVFYOP= verify_audit_version_delete_exit_options
,AVDNTF= notify_audit_version_delete_exit_routine
,AVDNTFDS= notify_audit_version_delete_exit_dataset
,AVDNTFCM=
LINK
ATTACH
TSOLNK
ISPLNK

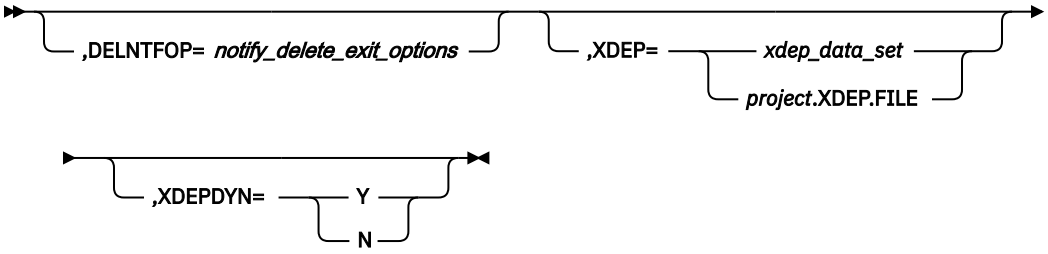

,CCSAVOP= save_change_code_exit_options
,AVDVFY= verify_audit_version_delete_exit_routine
,AVDVFYDS= verify_audit_version_delete_exit_dataset
,AVDVFYCM=
LINK
ATTACH
TSOLNK
ISPLNK
,AVDVFYOP= verify_audit_version_delete_exit_options
,AVDNTF= notify_audit_version_delete_exit_routine
,AVDNTFDS= notify_audit_version_delete_exit_dataset
,AVDNTFCM=
LINK
ATTACH
TSOLNK
ISPLNK


,AVDNTFOP= notify_audit_version_delete_exit_options
,BLDINIT=build_initial_user_exit_routine
,BLDINIDS=build_initial_user_exit_dataset
,BLDINICM=
LINK
ATTACH
TSOLNK
ISPLNK
,BLDINIOP=build_initial_user_exit_options
,BLDNTF= build_notify_user_exit_routine
,BLDNTFDS= build_notify_user_exit_dataset
,BLDNTFCM=
LINK
ATTACH
TSOLNK
ISPLNK


,BLDNTFOP= build_notify_user_exit_options
,PRMINIT=promote_initial_user_exit_routine
,PRMINIDS=promote_initial_user_exit_dataset
,PRMINICM=
LINK
ATTACH
TSOLNK
ISPLNK
,PRMINIOP=promote_initial_user_exit_options
,PRMVFY= promote_verify_user_exit_routine
,PRMVFYDS= promote_verify_user_exit_dataset
,PRMVFYCM=
LINK
ATTACH
TSOLNK
ISPLNK


,BLDNTFOP= build_notify_user_exit_options
,PRMINIT=promote_initial_user_exit_routine
,PRMINIDS=promote_initial_user_exit_dataset
,PRMINICM=
LINK
ATTACH
TSOLNK
ISPLNK
,PRMINIOP=promote_initial_user_exit_options
,PRMVFY= promote_verify_user_exit_routine
,PRMVFYDS= promote_verify_user_exit_dataset
,PRMVFYCM=
LINK
ATTACH
TSOLNK
ISPLNK
,PRMVFYOP= promote_verify_user_exit_options
,PRMCOPY= promote_copy_user_exit_routine
,PRMCPYDS= promote_copy_user_exit_dataset


,PRMCPYCM=
LINK
ATTACH
TSOLNK
ISPLNK
,PRMCPYOP= promote_copy_user_exit_options
,PRMPURGE= promote_purge_user_exit_routine
,PRMPRGDS= promote_purge_user_exit_dataset
,PRMPRGCM=
LINK
ATTACH
TSOLNK
ISPLNK
,PRMPRGOP= promote_purge_user_exit_options
,DELINIT=initial_delete_exit_routine
,DELINIDS=initial_delete_exit_dataset
,DELINICM=
LINK
ATTACH
TSOLNK
ISPLNK
,DELINIOP=initial_delete_exit_options
,DELVFY= verify_delete_exit_routine
,DELVFYDS= verify_delete_exit_dataset
,DELVFYCM=
LINK
ATTACH
TSOLNK
ISPLNK
,DELVFYOP= verify_delete_exit_options
,DELNTF= notify_delete_exit_routine
,DELNTFDS= notify_delete_exit_dataset
,DELNTFCM=
LINK
ATTACH
TSOLNK
ISPLNK



Parameters

ACCT=*primary_account_data_set*|project.ACCOUNT.FILE
The name of the primary accounting data set for the project. The data set you specify must be the name of the VSAM cluster you want to use. The default accounting data set name is *project*.ACCOUNT.FILE, where *project* is the project name specified on the FLMABEG macro. The ACCT parameter on the FLMALTC macro can be used to override the primary accounting data set for a specific group or set of groups. No SCLM variables can be used for this parameter.

,ACCT2=*secondary_account_data_set*
The name of a secondary accounting data set for the project. Allocate this secondary VSAM data set following the same criteria as the primary accounting data set. Choose a unique name for this data set. It should reside on a different volume than the primary one. If a severe problem occurs with the primary data set (for example, a head crash on that disk), you can use this backup data set to restore the primary data set. The default is no secondary accounting data set. The ACCT2 parameter on the FLMALTC macro can be used to override the secondary VSAM accounting data set for a specific group or set of groups.

Because additional accounting updates take place if you use this option, be aware that the updates will degrade overall performance. No SCLM variables can be used for this parameter.

,EXPACCT=*export_account_data_set*
The name of the export accounting data set used for exporting or importing project accounting information. The data set you specify must be the name of the VSAM cluster you want to use and must have a different name from any ACCT or ACCT2 parameter specified in FLMCNTRL or any FLMALTC macro. The default is no export accounting data set. The EXPACCT parameter on the FLMALTC macro can be used to override the export accounting data set for a specific group or set of groups. These variables can be used in specifying the name of the export accounting data set name:

- @@FLMPRJ
- @@FLMGRP
- @@FLMUID

,VERS=*primary_audit_control_data_set*
The name of the primary audit control data set for the project. This parameter is required to perform audit and versioning for groups that do not reference an FLMALTC macro with VERS specified. If you specify the VERS keyword and omit the primary_audit_control_data_set name, errors occur later during processing. The default is no audit control data set.

,VERS2=*secondary_audit_control_data_set*
The name of the secondary audit control data set for the project. If you specify the VERS2 keyword and omit the secondary_audit_control_data_set name, errors occur later during processing. The default is no secondary audit control data set. The VERS2 parameter on the FLMALTC macro can be used to override the secondary audit control data set for a specific group or set of groups.

Because additional audit record updates occur if this option is used, be aware that overall performance will degrade. Do not specify VERS2 unless you have specified VERS. If you do, an error will occur when the project definition is assembled.

,VSAMRLS=NO | YES
Indicates whether SCLM should allow the VSAM data sets to be shared across systems when the level of DFSMS installed is 1.3 or later. The default is NO.

SCLM uses VSAM Record Level Sharing (RLS) to allow the sharing of the VSAM data sets. To maintain the integrity of the VSAM data sets in a shared environment, the VSAM data sets must be allocated for RLS and all hardware and software to support RLS must be in place for the system. (See the DFSMS documentation for hardware and software requirements.)

The VSAM data sets cannot be shared under any other condition. Accessing any of the VSAM data sets from multiple systems when VSAM RLS is not available can result in the corruption of data, system errors, or other integrity problems. To avoid these problems, the project manager must allocate the VSAM data sets so that they cannot be accessed from multiple systems.

,VERPDS=*version_pds_name*
The name of the partitioned data set to contain the version data. These variables can be used when specifying the name of the partitioned data set: @@FLMPRJ, @@FLMGRP, and @@FLMTYP, or @@FLMDSN. For example:

- VERPDS=@@FLMPRJ.@@FLMGRP.@@FLMTYP.VERSION
- VERPDS=@@FLMDSN.VERSION
- VERPDS=@@FLMPRJ.VERSIO.@@FLMGRP

This parameter is optional. If you do not specify a value, the value @@FLMDSN.VERSION is assigned to the parameter (even if versioning is not active.) See the description of the DSNAME parameter for more information about the value of @@FLMDSN.

If @@FLMDSN is used, it must be specified in the first 8 characters of the VERPDS= statement. For example, VERPDS=@@FLMDSN.VERSN12 is valid, but VERPDS=@@FLMPRJ.@@FLMDSN.VERSN12 is not valid. The VERPDS parameter on the FLMALTC macro can be used to override the version data partitioned data set for a specific group or set of groups.

You can have only one VERPDS data set per group and type at a time. However, you can respecify the VERPDS data set name to control the size of the version data sets. If the VERS=primary audit control data set name remains the same, a pointer to the VERPDS that holds a particular version allows you to retrieve and delete versions of members, even if you have changed the name of the VERPDS data set.

The FLMATVER macro must be used to enable versioning for particular groups. If you specify a value of 2 or more for the VERCOUNT parameter on the FLMCNTRL macro, you must specify a separate VERPDS for each combination of group and type that you intend to version.

Note: Failure to specify a separate VERPDS for each combination of group and type can cause retrieval problems.

,VERCOUNT=*number_to_retain*
The number of versions to keep in the version partitioned data set. If you specify a value of 0 (the default), all versions associated with a member will be kept. If you specify a value of 2 or more, each time a member is saved or promoted, the latest copy of the version is stored and the earliest copy is disposed. Any audit records that were associated with the version are retained but will no longer indicate that a version of the member exists. If you do specify a value of 2 or more, allocate a separate VERPDS for each combination of group and type that has versioning enabled.

Note: Failure to allocate a separate VERPDS for each combination of group and type can cause retrieval problems. Use the FLMALTC macro, or use the @@FLMGRP variable in the VERPDS name.

If you specify a value of 1, an error will occur when the project definition is assembled. The only version maintained in this case would be a full source copy of the member that exists in the project hierarchy.

,DSNAME=*dataset_name*
This parameter lets you specify the data set naming conventions for the project partitioned data sets controlled by SCLM. The naming convention is specified as a pattern that can include a subset of the SCLM variables.

The only SCLM variables that can be used in the DSNAME parameter of FLMCNTRL are:

- @@FLMPRJ
- @@FLMGRP
- @@FLMTYP

The value specified in this parameter is used to resolve the SCLM variable @@FLMDSN. If this parameter is not specified, the data set name pattern defaults to @@FLMPRJ.@@FLMGRP.@@FLMTYP. You can enter up to 44 characters for this parameter, including the SCLM variables and the periods.

If a data set name is specified, it must include the SCLM variable @@FLMTYP. It is also recommended that the variable @@FLMGRP be used in the data set name pattern. This helps prevent data from one group overwriting data in another group.

Attention: SCLM does not enforce or verify the uniqueness of partitioned data set names.

The DSNAME parameter on the FLMALTC macro can be used to override the data set naming conventions for a specific group or set of groups.

The variables can appear in any location within the DSNAME parameter. Any user-specified qualifiers can also be used. The preceding SCLM variables can contain values up to 8 characters.

Examples of data set name lengths are:

- APPL4.@@FLMGRP.@@FLMTYP is 5 + 1 + 8 + 1 + 8 = 23
- REL30.COMMON2A.@@FLMGRP.@@FLMTYP is 5 + 1 + 8 + 1 + 8 + 1 + 8 = 32

The resulting data set name must meet all of the requirements specified by the MVS data set naming conventions. If the data set name is too long or it does not meet MVS data set naming conventions, then errors occur during SCLM functions (for example, Build or Promote).

,DASDUNIT=*dasd_unit_name*|SYSALLDA
The name of the unit where DASD data sets will reside. The maximum DASD unit name length is 8 characters. The default is SYSALLDA.

,VIOUNIT=*VIO_unit_name*|VIO
The name of the unit where a temporary VIO data set will reside. The maximum VIO unit name length is 8 characters. The default is VIO. For more information on MAXVIO, see MAXVIO ",MAXVIO=max_vio_count|5000" on page 527.

,MAXLINE=*max_line_count*|60
An integer value indicating the maximum number of lines per page for all SCLM reports. The minimum value you can specify is 35, and the default is 60.

,MAXVIO=*max_vio_count*|5000
An integer value indicating the maximum number of records permitted for VIO allocation. The default is 5000. The maximum value is 2147483647.

,OPTOVER=N|Y
Indicates whether translator option overrides are allowed or disallowed. If OPTOVER=Y, developers can override the translator options by specifying the keyword PARMx in the architecture member followed by the new options. The default is Y. See the PARMx parameter for more information.

,MEMLOCK=N|Y
Specifies whether member level locking is in enabled. If MEMLOCK=Y, the member can't be updated by another SCLM user. The default is N.

,CONTROL=*control_data_set*
Specifies the VSAM data set where the SCLM administrator information and control information is stored. If MEMLOCK=Y, or a value for NOPROMBK is specified, this parameter must be specified

,NPROMBK=*not_promoted_backup_data_set*
The name of the partitioned data set to contain a backup of unpromoted members. This parameter is optional. If you do not specify a NOPROMBK value, promoted members are not backed up. No SCLM variables can be used for this parameter.

If the NPROMBK parameter is coded, the CONTROL parameter on FLMCNTRL must be specified.

,ADMINID=*administrator_userid*
Specifies the user ID of the default SCLM administrator.

,VERCC=*change_code_routine*
The member name of the change code verification routine. Specify the data set containing the member in the VERCCDS parameter. If you do not specify the VERCC parameter, then SCLM does not invoke the exit routine.

,VERCCDS=*change_code_dataset*
The name of the data set containing the translator load module, REXX exec, or CLIST specified by the VERCC parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

,VERCCCM=LINK|ATTACH|TSOLNK|ISPLNK
Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the VERCC parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the VERCCOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the VERCCOP parameter.

The default is LINK.

,VERCCOP=*change_code_options*
Option list to be passed to the VERCC user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a nonblank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, VERCCCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *z/OS ISPF Services Guide*.

,CCVFY=*verify_change_code_exit_routine*
The name of the verify change code exit routine. If you do not specify the CCVFY parameter, SCLM does not invoke the exit routine.

,CCVFYDS=*verify_change_code_exit_dataset*
The name of the data set containing the translator load module, REXX exec or CLIST specified by the CCVFY parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters.

,CCVFYCM=LINK|ATTACH|TSOLNK|ISPLNK
Indicates whether the exit routine is to be linked, attached, invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF

variables or services; in that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the CCVFY parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the CCVFYOP parameter. The name of the load module, CLIST, REXX exec or other command is also specified as part of the CCVFYOP parameter.

The default is LINK.

,CCVFYOP=*verify_change_code_exit_options*
Option list to be passed to the CCVFY user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a nonblank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, CCVFYCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For a description of the ISPF SELECT service, refer to the *z/OS ISPF Services Guide*.

,CCSAVE=*save_change_code_exit_routine*
The name of the save change code exit routine. If you do not specify the CCSAVE parameter, SCLM does not invoke the exit routine.

,CCSAVDS=*save_change_code_exit_dataset*
The name of the data set containing the translator load module, REXX exec or CLIST specified by the CCSAVE parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters.

,CCSAVCM=LINK|ATTACH|TSOLNK|ISPLNK
Indicates whether the exit routine is to be linked, attached, invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services; in that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the CCSAVE parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the CCSAVOP parameter. The name of the load module, CLIST, REXX exec or other command is also specified as part of the CCSAVOP parameter.

The default is LINK.

,CCSAVOP=*save_change_code_exit_options*
Option list to be passed to the CCSAVE user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a nonblank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, CCSAVCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For a description of the ISPF SELECT service, refer to the *z/OS ISPF Services Guide*.

,AVDVFY=*verify_audit_version_delete_exit_routine*
The name of the audit version delete verification exit routine. If you do not specify the DELVFY parameter, SCLM does not invoke the exit routine.

,AVDVFYDS=*verify_audit_version_delete_exit_dataset*
The name of the data set containing the translator load module, REXX exec, or CLIST specified by the AVDVFY parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

,AVDVFYCM=LINK|ATTACH|TSOLNK|ISPLNK
Indicates whether the exit routine is to be linked, attached, invoked by the TSO service facility routine, or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the AVDVFY parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the AVDVFYOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the AVDVFYOP parameter.

The default is LINK.

,AVDVFYOP=*verify_audit_version_delete_exit_options*
Option list to be passed to the AVDVFY user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a nonblank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, AVDVFYCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *z/OS ISPF Services Guide*.

,AVDNTF=*notify_audit_version_delete_exit_routine*
The name of the audit version delete notification exit routine. If you do not specify the AVDNTF parameter, SCLM does not invoke the exit routine.

,AVDNTFDS=*notify_audit_version_delete_exit_dataset*
The name of the data set containing the translator load module, REXX exec, or CLIST specified by the AVDNTF parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

,AVDNTFCM=LINK|ATTACH|TSOLNK|ISPLNK
Indicates whether the exit routine is to be linked, attached, invoked by the TSO service facility routine, or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the AVDVFY parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the AVDNTFOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the AVDNTFOP parameter.

The default is LINK.

,AVDNTFOP=*verify_audit_version_delete_exit_options*
Option list to be passed to the AVDNTF user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a nonblank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, AVDNTFCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *z/OS ISPF Services Guide*.

,BLDINIT=*build_initial_user_exit_routine*
The member name of the Build Initial user exit routine. SCLM invokes the routine at the beginning of the build process during initialization. Specify the data set containing the member using the BLDINIDS parameter. If you do not specify the BLDINIT parameter, then SCLM does not invoke the exit routine.

,BLDINIDS=*build_initial_user_exit_dataset*
The name of the data set containing the translator load module, REXX exec, or CLIST specified by the BLDINIT parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

,BLDINICM=LINK|ATTACH|TSOLNK|ISPLNK
Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the BLDINIT parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the BLDINIOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the BLDINIOP parameter.

The default is LINK.

,BLDINIOP=*build_initial_user_exit_options*
Option list to be passed to the BLDINIT user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a nonblank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, BLDINICM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *z/OS ISPF Services Guide*.

,BLDNTF=*build_notify_user_exit_routine*
The member name of the build notification user exit routine. SCLM invokes the routine at the end of the build process after the build has taken place. Specify the data set containing the member using the BLDNTFDS parameter. If you do not specify the BLDNTF parameter, then SCLM does not invoke the exit routine.

Note: The original format for this user exit, using the BLDEXT1 parameter, is still supported by SCLM. However, parameters used with this exit routine must be either *ALL* in the old format or *ALL* in the new format. Specifying the user exit routine in both the old and new formats, or mixing old and new format parameters for the same exit causes errors when the project definition is assembled.

,BLDNTFDS=*build_notify_user_exit_dataset*
The name of the data set containing the translator load module, REXX exec, or CLIST specified by the BLDNTF parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

,BLDNTFCM=LINK|ATTACH|TSOLNK|ISPLNK
Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the BLDNTF parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the BLDNTFOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the BLDNTFOP parameter.

The default is LINK.

,BLDNTFOP=*build_notify_user_exit_options*
Option list to be passed to the BLDNTF user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a nonblank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, BLDNTFCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *z/OS ISPF Services Guide*.

,PRMINIT=*promote_initial_user_exit_routine*
The member name of the initial promote user exit routine. SCLM invokes this routine at the beginning of the promote process during initialization. Specify the data set containing the member in the PRMINIDS parameter. If you do not specify the PRMINIT parameter, then SCLM does not invoke the exit routine.

,PRMINIDS=*promote_initial_user_exit_dataset*
The name of the data set containing the translator load module, REXX exec, or CLIST specified by the PRMINIT parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

,PRMINICM=LINK|ATTACH|TSOLNK|ISPLNK
Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out_of_space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the PRMINIT parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the PRMINIOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the PRMINIOP parameter.

The default is LINK.

,PRMINIOP=*promote_initial_user_exit_options*
Option list to be passed to the PRMINIT user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a nonblank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, PRMINICM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *z/OS ISPF Services Guide*.

,PRMVFY=*promote_verify_user_exit_routine*
The member name of the promote verification user exit routine. SCLM invokes this routine at the end of the verification phase of the promote process. Specify the data set containing the member in the PRMVFYDS parameter. If you do not specify the PRMVFY parameter, then SCLM does not invoke the exit routine.

Note: The original format for this user exit, using the PRMEXT1 parameter, is still supported by SCLM. However, parameters used with this exit routine must be either *ALL* in the old format or *ALL* in the new format. Specifying the user exit routine in both the old and new formats, or mixing old and new format parameters for the same exit causes errors when the project definition is assembled.

,PRMVFYDS=*promote_verify_user_exit_dataset*
The name of the data set containing the translator load module, REXX exec, or CLIST specified by the PRMVFY parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

,PRMVFYCM=LINK|ATTACH|TSOLNK|ISPLNK
Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the PRMVFY parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the PRMVFYOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the PRMVFYOP parameter.

The default is LINK.

,PRMVFYOP=*promote_verify_user_exit_options*
Option list to be passed to the PRMVFY user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single

quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a nonblank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, PRMVFYCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *z/OS ISPF Services Guide*.

,PRMCOPY=*promote_copy_user_exit_routine*
The member name of the Promote Copy user exit routine. SCLM invokes this routine at the end of the copy phase of the promote process. Specify the data set containing the member in the PRMCPYDS parameter. If you do not specify the PRMCOPY parameter, then SCLM does not invoke the exit routine.

Note: The original format for this user exit, using the PRMEXT2 parameter, is still supported by SCLM. However, parameters used with this exit routine must be either *ALL* in the old format or *ALL* in the new format. Specifying the user exit routine in both the old and new formats, or mixing old and new format parameters for the same exit causes errors when the project definition is assembled.

,PRMCPYDS=*promote_copy_user_exit_dataset*
The name of the data set containing the translator load module, REXX exec, or CLIST specified by the PRMCOPY parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

,PRMCPYCM=LINK|ATTACH|TSOLNK|ISPLNK
Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the PRMCOPY parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the PRMCPYOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the PRMCPYOP parameter.

The default is LINK.

,PRMCPYOP=*promote_copy_user_exit_options*
Option list to be passed to the PRMCOPY user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a nonblank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, PRMCPYCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *z/OS ISPF Services Guide*.

,PRMPURGE=*promote_purge_user_exit_routine*
The member name of the promote purge user exit routine. SCLM invokes this routine at the end of the copy phase of the promote process. Specify the data set containing the member in the PRMPRGDS parameter. If you do not specify the PRMPURGE parameter, then SCLM does not invoke the exit routine.

Note: The original format for this user exit, using the PRMEXT3 parameter, is still supported by SCLM. However, parameters used with this exit routine must be either *ALL* in the old format or *ALL* in the new format. Specifying the user exit routine in both the old and new formats, or mixing old and new format parameters for the same exit causes errors when the project definition is assembled.

,PRMPRGDS=*promote_purge_user_exit_dataset*
The name of the data set containing the translator load module, REXX exec, or CLIST specified by the PRMPURGE parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

,PRMPRGCM=LINK|ATTACH|TSOLNK|ISPLNK
Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the PRMPURGE parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the PRMPRGOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the PRMPRGOP parameter.

The default is LINK.

,PRMPRGOP=*promote_purge_user_exit_options*
Option list to be passed to the PRMPURGE user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a nonblank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, PRMPRGCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *z/OS ISPF Services Guide*.

,DELINIT=*initial_delete_exit_routine*
The name of the initial delete exit routine. If you do not specify the DELINIT parameter, then SCLM does not invoke the exit routine. This routine is only invoked for the DELGROUP service or Delete from Group dialog (ISPF Option 10.3.9).

,DELINIDS=*initial_delete_exit_dataset*
The name of the data set containing the translator load module, REXX exec, or CLIST specified by the DELINIT parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

,DELINICM=LINK|ATTACH|TSOLNK|ISPLNK
Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out_of_space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the DELINIT parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the DELINIOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the DELINIOP parameter.

The default is LINK.

,DELINIOP=*initial_delete_exit_options*
Option list to be passed to the DELINIT user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a nonblank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, DELINICM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *z/OS ISPF Services Guide*.

,DELVFY=*verify_delete_exit_routine*
The name of the delete verification exit routine. If you do not specify the DELVFY parameter, then SCLM does not invoke the exit routine. This exit routine is invoked for Library Utility Delete (ISPF Option 10.3.1) or the Delete service.

,DELVFYDS=*verify_delete_exit_dataset*
The name of the data set containing the translator load module, REXX exec, or CLIST specified by the DELVFY parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

,DELVFYCM=LINK|ATTACH|TSOLNK|ISPLNK
Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the DELVFY parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the DELVFYOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the DELVFYOP parameter.

The default is LINK.

,DELVFYOP=*verify_delete_exit_options*
Option list to be passed to the DELVFY user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a nonblank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, DELVFYCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *z/OS ISPF Services Guide*.

,DELNTF=*notify_delete_exit_routine*
The name of the delete notification exit routine. If you do not specify the DELNTF parameter, then SCLM does not invoke the exit routine. This exit is invoked for Library Utility Delete (ISPF Option 10.3.1), the DELGROUP service or Delete from Group dialog (ISPF Option 10.3.9) or Delete service.

,DELNTFDS=*notify_delete_exit_dataset*
The name of the data set containing the translator load module, REXX exec, or CLIST specified by the DELNTF parameter. The data set name is not required when the translator resides in one of the system concatenation libraries. The data set name can be up to 44 characters long.

,DELNTFCM=LINK|ATTACH|TSOLNK|ISPLNK
Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services. In that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the DELNTF parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the DELNTFOP parameter. The name of the load module, CLIST, REXX exec, or other command is also specified as part of the DELNTFOP parameter.

The default is LINK.

,DELNTFOP=*notify_delete_exit_options*
Option list to be passed to the DELNTF user exit routine. You can specify a maximum of 255 characters for the options, including delimiters. Enclose the option string in parentheses or single quotes. The options string precedes the list of parameters passed to the exit routine by SCLM. SCLM removes any trailing blanks and does not add a delimiter between the option string and the SCLM parameters. End the options string with a nonblank delimiter so that the options and parameters can be identified by the exit routine.

When the call method for the exit routine, DELNTFCM, is ISPLNK, the options string must contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the service. For more information about the ISPF SELECT service, refer to the *z/OS ISPF Services Guide*.

XDEP=xdep_data_set|project.XDEP.FILE
The name of the cross-dependency file for the project. The name you specify must be the name of the VSAM cluster you want to use. The default cross-dependency file name is *project*.XDEP.FILE, where *project* is the name specified on the FLMABEG macro. The XDEP parameter on the FLMALTC macro can be used to override the cross-dependency file name for a specific group or set of groups. No SCLM variables can be used for this parameter.

Note: The XDEP data set name supplied on the FLMCNTRL macro is used if the XDEP parameter is not specified on any FLMALTC macros. If one FLMALTC macro supplies an XDEP data set name, then all FLMALTC macros must supply an XDEP data set name if no XDEP value is specified on the FLMCNTRL macro.

XDEPDYN=Y|N
Indicates whether the Cross-dependency data set is to kept in sync with the accounting data set. If Y is specified, then when each accounting data set record is updated the changes are analysed and one or more cross-dependency records may be updated. Maintaining the Cross-dependency database dynamically requires additional I/O operations. Specify N if the extra I/O activity creates performance problems. If N is specified, then it is necessary to run the XDEPUPDT service to synchronise the cross-dependency data from the accounting data set.

Example

This information has been specified: the accounting data set, allocation of data sets using VIO if the FLMALLOC macro RECNUM parameter is not greater than 10000 and a verify change code exit routine that is called using ISPLNK.

```
      FLMCNTRL ACCT=PROJ1.ACCOUNT.FILE,                          X
            CCVFY=SELECT,                                        X
            CCVFYCM=ISPLNK,                                      X
            CCVFYOP='CMD(SAYPARM ,',                             X
            MAXVIO=10000
```

This information has been specified: the accounting data set, the versioning VSAM database, the versioning PDS, the maximum number of versions to be kept for any member, allocation of data sets using VIO if the FLMALLOC macro RECNUM parameter is not greater than 999,999, and the unit for any VIO data sets is VIO.

```
      FLMCNTRL ACCT=PROD1.ACCOUNT.FILE,                          X
            VERS=PROD1.VERSION.FILE,                             X
            VERPDS=PROD1.VERSION.PDS,                            X
            VERCOUNT=5,                                          X
            MAXVIO=999999,                                       X
            VIOUNIT=VIO
```

Here is an example using member level locking:

This information has been specified: the accounting data set, the member level locking control data set, member level locking is turned on, and the administrator ID for member level locking is SHELLY.

```
      FLMCNTRL ACCT=PROD1.ACCOUNT.FILE,                          X
            CONTROL=PROD1.CONTROL.FILE,                          X
            MEMLOCK=Y,                                           X
            ADMINID=SHELLY
```

FLMCPYLB macro

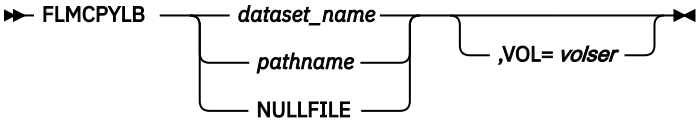
The FLMCPYLB macro identifies the name of a data set to be allocated by an occurrence of the FLMALLOC macro. FLMCPYLB macros that do not immediately follow either an FLMALLOC macro or another FLMCPYLB macro are ignored.

FLMCPYLB macros for build translators support only a limited set of the variables that are discussed in Chapter 23, “SCLM Variables and Metavariables,” on page 631. For Build translators, the FLMCPYLB macro must be associated with an FLMALLOC macro that has its IOTYPE set to A, H, or I. When the FLMCPYLB macro meets these conditions, these variables are supported:

- @@FLMDBQ
- @@FLMSRF
- @@FLMPRJ
- @@FLMALT
- @@FLMUID
- @@FLMGRP. For build translators: The value for group in @@FLMGRP will be the group where the member referenced on the first SINC statement is found if the architecture definition being built is a CC or Generic architecture definition. If the architecture definition is an HL or LEC architecture definition, the value for @@FLMGRB will be the group where the build is taking place.
- @@FLMMBR. The @@FLMMBR variable is replaced with the name of the member being built. For a CC or a generic architecture definition, it is the name of the architecture definition.
- @@FLMTYP. The @@FLMTYP variable is replaced with the name of the type of the member being built. For a CC or generic architecture definition, it is the type of the architecture definition.

- @@FLMDSN

Macro format



Parameters

***dataset_name*|*pathname*|NULLFILE**

Use the FLMCPYLB macro to allocate a data set or z/OS UNIX file to a ddname. Place the FLMCPYLB after an FLMALLOC macro with IOTYPE=A, H, or I. See the IOTYPE parameter on the FLMALLOC macro for more information. For all other IOTYPEs, SCLM ignores the data sets, unless MALLOC=Y. When MALLOC=Y, the IOTYPE can be either O or A.

If you specify more than one FLMCPYLB, SCLM concatenates the data sets in the order they are specified. When you use them with IOTYPE=I, SCLM allocates the data sets after the type hierarchy libraries. SCLM can concatenate up to 123 data sets. Thus, when you use IOTYPE=I, ensure that the number of groups in the hierarchy (primary groups), plus the number of FLMCPYLB macros you specify does not exceed 123. If you concatenate more than 123 data sets, the project definition assembles without errors but using it produces unpredictable results. Concatenation is not supported for IOTYPE=H.

You can specify partitioned data sets, sequential data sets, members of fully qualified partitioned data sets, or z/OS UNIX path names. Specify NULLFILE to allocate a dummy data set.

FLMCPYLB data set names can contain these SCLM variables

- @@FLMDBQ
- @@FLMSRF
- @@FLMPRJ
- @@FLMALT
- @@FLMUID
- @@FLMGRP
- @@FLMGRB
- @@FLMMBR
- @@FLMTYP
- @@FLMDSN

The specified data set name or z/OS UNIX path name, or the resulting data set name when SCLM variables are used, must meet all of the requirements of MVS data set names. For MVS data set names, the project definition allows up to 54 characters, including periods and parentheses, to support a data set with member name specification. z/OS UNIX path names must start with a forward slash (/). They can be up to 255 characters. A data set name or path name that contains SCLM variables and is longer than this will cause errors when the project definition is assembled, even if the substituted value meets all MVS naming conventions. A data set name that is allowed by the project definition, but does not meet MVS naming convention restrictions (for example, a data set name without the member specified that is more than 44 characters long), causes errors to occur during SCLM functions such as Build.

,VOL=volser

Specifies the serial number of an eligible direct access volume on which the data set is located. This allows reference to a data set that is either uncataloged or that is located on a different volume than the catalog specifies. The default action, if not specified, is to use the volume in the data set's catalog entry.

Note:

1. If an SMS-managed volume is specified, the system will override this specification with the volume in the catalog entry.
2. The VOL keyword cannot be specified with a z/OS UNIX path name or the NULLFILE keyword.
3. The VOL keyword cannot be specified with MALLOC=Y on the FLMALLOC macro.

Example

The three data sets specified by the FLMCPYLB macro are allocated to the DDNAME ISPLOAD.

```
FLMALLOC IOTYPE=A,DDNAME=ISPLOAD
  FLMCPYLB PROJ1.INTERNAL.LOAD
  FLMCPYLB SYS2.ISPF.LOAD
  FLMCPYLB SYS1.LINKLIB
```

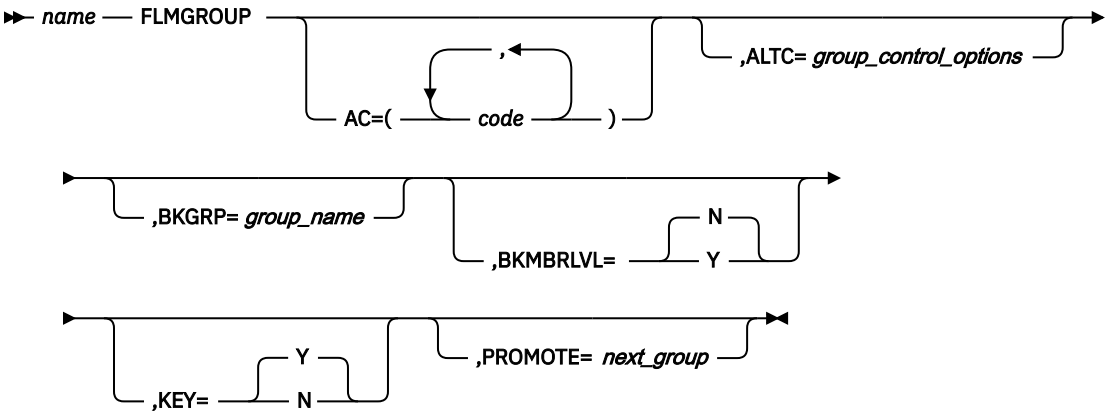
The number of concatenated data sets and the names of the data sets are verified at run time.

If some includes are coming from system libraries instead of from the hierarchy, FLMCPYLB macros might be needed to allow the compiler or other build processors to find those includes. The FLMCPYLB macros are needed if FLMSYSLB macros are used for the language and the language definition macro (FLMLANGL) has ALCSYSLB=N. In this case, an FLMCPYLB macro must be specified for each FLMSYSLB macro.

FLMGROUP macro

Use this macro to define each group in the project definition. This macro is required and can be used multiple times.

Macro format



Parameters

name
An 8-character group name.

AC=(*code*)
A list of authorization codes and authorization groups that defines the authorization codes for the given group. If any item in the list is an authorization group, you must have previously defined it with the FLMAGRP macro.

The first authorization code you specify is the default authorization code used when a member is introduced to SCLM in this group. Each authorization code can be up to 8 characters and cannot

contain commas. The maximum number of characters allowed for the authorization code list is 255, including commas and the delimiting parentheses.

If you omit this parameter, you cannot edit any members in this group. In addition, no editable members can be promoted into or out of this group.

,ALTC=*group_control_options*
Specifies an alternate set of control options to be used for this group. The name must match the name of an FLMALTC macro in the project definition. The data sets defined on the referenced FLMALTC macro are used to store the information for this group instead of the data sets specified on the FLMCNTRL macro. If this parameter is not specified, the group uses the data sets specified on the FLMCNTRL macro.

,BKGRP=*group_name*
Specifies the group to which this FLMGROUP is backed up.

,BKMBRLVL=N|Y
Defines whether member-level restore is activated.

,KEY=N|Y
Defines whether the group is a key group or a non-key group. The default is Y. The KEY parameter does not apply to groups specified with the EXLIBID parameter.

,PROMOTE=*next group*
Defines the next higher group within the hierarchy for this group. If you do not specify it, SCLM does not allow any promotions out of this group.

Example 1

Seven groups are defined for this project definition. The hierarchy consists of five layers. Groups DEV1 and DEV2 are defined as development groups because no groups promote to them. All groups except for the TEST group are defined as key groups. A list of authorization codes are assigned to each group. Group RELEASE is defined as the highest group in the hierarchy because it does not specify the PROMOTE parameter.

```
DEV1     FLMGROUP AC=(R6M0),KEY=Y,PROMOTE=STAGE1
DEV2     FLMGROUP AC=(R7M0),KEY=Y,PROMOTE=STAGE2
STAGE1   FLMGROUP AC=(R6M0,R7M0),KEY=Y,PROMOTE=INT
STAGE2   FLMGROUP AC=(R6M0,R7M0),KEY=Y,PROMOTE=INT
INT      FLMGROUP AC=(R6M0,R7M0),KEY=Y,PROMOTE=TEST
TEST     FLMGROUP AC=(R6M0,R7M0),KEY=N,PROMOTE=RELEASE
RELEASE  FLMGROUP AC=(R6M0),KEY=Y
```

Example 2

In this example:

- The ALTC parameter of the DEV group specifies that the control information defined by the FLMALTC macro DEVCNTL is used instead of the control information defined by the FLMCNTRL macro. The PDS data sets associated with this group have the naming convention SWDEV.PROJXYZ.DEV.type.
- The INT group uses the control information defined by the FLMCNTRL macro. The data set name used for SCLM-controlled PDS data sets defaults to @@FLMPRJ.@@FLMGRP.@@FLMTYP, resulting in a naming convention of PROJXYZ.INT.type for these data sets.
- The accounting database used by the REL group is PROJ2.ACCT.DATABASE as defined by the RELCNTL FLMALTC macro. The naming convention used for the PDS data sets is RELEASE.PROJ2.REL.type.

```
PROJXYZ  FLMABEG

         FLMCNTRL ACCT=PROJXYZ.ACCT.DATABASE

RELCNTL  FLMALTC  ACCT=PROJ2.ACCT.DATABASE,                          C
               DSNAME=RELEASE.PROJ2.@@FLMGRP.@@FLMTYP

DEVCNTL  FLMALTC  ACCT=PROJDEV.ACCT.DATABASE,                         C
               DSNAME=SWDEV.@@FLMPRJ.@@FLMGRP.@@FLMTYP
```

```
REL      FLMGROUP KEY=Y,ALTC=RELCNTL
INT      FLMGROUP KEY=Y,PROMOTE=REL
DEV      FLMGROUP KEY=Y,PROMOTE=INT,ALTC=DEVCNTL

         FLMAEND
```

FLMINCLS macro

Use this macro to associate include sets with types in the project hierarchy. This association is used to determine the location of include members within the project. Parsers may be written to associate an include set with each include found in a source member. This macro indicates to the build function where include sets can be found and which data sets to allocate for input to build translators. This macro is part of a language definition. The FLMINCLS macro must follow the FLMLANG macro for the language definition.

The FLMSYSLB macro is used to specify data sets outside the project that contain includes. The INCLS parameter value on the FLMSYSLB macro associates the name of an include set with the FLMSYSLB libraries. The search for an include member will first take place in the include set and then in the associated FLMSYSLB.

The default include set is associated with an FLMSYSLB that has no INCLS parameter. The default include set is specified by an FLMINCLS that has no name parameter. Only one default FLMINCLS may be specified for each language definition. SCLM will generate a default include set if one is not specified.

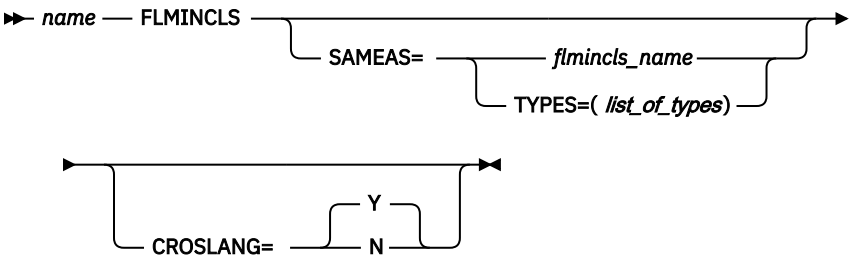
Use the INCLS parameter on an FLMALLOC macro of IOTYPE=I to associate an FLMINCLS macro with an FLMALLOC macro. If no name is specified on the FLMINCLS macro, the macro is for the default include set. The default include set is associated with FLMALLOC macros of IOTYPE=I where no INCLS parameter is specified.

If an FLMINCLS macro is specified for the default include set, at least one FLMALLOC macro must reference the default include set (by specifying IOTYPE=I and no INCLS parameter). If there is no FLMINCLS macro in the language, an FLMALLOC macro for the default include set is optional.

SCLM ensures that each language definition includes a default include set and a COMPOOL include set. If the language definition does not include macros to define these two include sets, these definitions are generated:

```
          FLMINCLS TYPES=(@@FLMTYP,@@FLMETP)
COMPOOL   FLMINCLS TYPES=(@@FLMCRF @@FLMECR)
```

Macro format



Parameters

name
The name of the include set that is being defined in this macro. An include set is associated with FLMSYSLB or FLMALLOC when the name matches the value of an INCLS parameter. In addition, the

name may be the name of an include set returned by a parser for the language that includes this FLMINCLS macro. Each include set name can only be used once per language definition.

To specify the default include set, leave this parameter blank.

SAMEAS=*flmincls_name*
The name of another FLMINCLS macro that contains the list of types to search. If you use this parameter, the include set defined by this macro has the same list of types as the include set listed on the SAMEAS parameter. You cannot reference the default include set by specifying SAMEAS= with a blank.

TYPES=*(list_of_types)*
A list of the types that contain the includes for the include set. Build searches these types in the order given on this parameter. The hierarchies for each type are concatenated for use by all FLMALLOC macros that reference this FLMINCLS macro.

Duplicate types are not removed from the list.

Two SCLM variables can be used on this parameter: @@FLMTYP and @@FLMETP. The value of @@FLMTYP is the type of the member on the first SINC statement in an architecture definition or the type of the member if a single member is being built. The value of @@FLMETP is the extended type of that member. (See the EXTEND parameter on the FLMTYPE macro).

The value that will be substituted into the @@FLMCRF variable is the DFLTCRF type. The value that will be substituted into the @@FLMECR variable for include set definitions is the extended type of the DFLTCRF type. If there is no extended type for the DFLTCRF type, the @@FLMECR variable will be ignored.

CROSLANG=Y|N
The CROSLANG parameter indicates whether SCLM processes the includes of an included member when the included member has a different language from the source member. Y indicates that includes are processed even if language boundaries are crossed. N indicates that only the includes of a member of the same language are processed. The value of the CROSLANG parameter is not affected by the SAMEAS parameter. The default for CROSLANG is Y.

Here is an example of how includes are processed given the two possible values for this parameter:

Member
Includes

SCRIPT1 language=SCRIPT
COBOL1 language=COBOL

COBOL1 language=COBOL
INCLUDE1 language=COBOL

INCLUDE1 language=COBOL
none

- If CROSLANG=Y when SCRIPT1 is built, the build processor checks the dates and times of COBOL1 and INCLUDE1 and puts them in the build map.
- If CROSLANG=N when SCRIPT1 is built, the build processor checks the dates and times of COBOL1 and puts them in the build map. The dates and times of INCLUDE1 are not processed.

Note: If both the SAMEAS and TYPES parameters are omitted for an FLMINCLS macro, no types are searched for that include set. This can be used when includes are only in data sets specified by FLMSYSLB macros or no includes of that type are allowed. Even if no parameters are specified on an FLMINCLS macro, it must be referenced by at least one FLMALLOC macro.

Example 1

The example shown here demonstrates how to define where the includes in the default include set are found. It indicates that the INCLUDE type is to be searched first, followed by the source type of the member being processed, and finally by the extended type of the source member if there is one. The

types listed on this macro are used for all IOTYPE=I FLMALLOC macros where no INCLS parameter is specified.

```
          FLMINCLS TYPES=(INCLUDE,@@FLMTYP,@@FLMETP)
```

Example 2

The example shown here demonstrates how to define where the includes in the MACRO and COPY include sets are found. It indicates that the MACRO type is the only type to be searched. Both the MACRO and COPY include sets are referenced by IOTYPE=I FLMALLOC macros. The IOTYPE=I FLMALLOC macros specify the allocation for the include hierarchies of the build translators. The MACRO FLMINCLS does not allow processing of includes across language boundaries. The COPY FLMINCLS processes the includes because the default value (Y) was not overridden for the CROSLANG parameter.

```
MACRO     FLMINCLS TYPES=(MACRO),CROSLANG=N
COPY      FLMINCLS SAMEAS=MACRO
```

Example 3

The example shown here demonstrates how to use different sequences of types for locating includes. This may be useful in situations in which includes in several different types have the same name.

```
         FLMLANGL    LANG=ABC,VERSION=1
*
* SEQUENCES OF TYPES FOR LOCATING INCLUDES
*
DBRM     FLMINCLS TYPES=(DBRMTYPE,@@FLMTYP,@@FLMETP)
SPECIAL  FLMINCLS TYPES=(COPYBOOK,SOURCE,MACRO,TOOLS)
*
* PARSER TRANSLATOR
*
          FLMTRNSL   CALLNAM='ABC PARSE',                            C
               FUNCTN=PARSE,                                         C
               COMPILE=FLMLPGEN,                                     C
               PORDER=1,                                             C
               GOODRC=0,                                             C
               OPTIONS=(SOURCEDD=SOURCE,                             C
               STATINFO=@@FLMSTP,                                    C
               LISTINFO=@@FLMLIS,                                    C
               LISTSIZE=@@FLMSIZ,                                    C
               LANG=T)
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMPRJ.@@FLMGRP.@@FLMTYP(@@FLMMBR)
*
* BUILD TRANSLATORS
*
          FLMTRNSL   CALLNAM='USE DEFAULT',                          C
               FUNCTN=BUILD,                                         C
               COMPILE=USEDFLT,                                      C
               VERSION=1.0,                                          C
               GOODRC=0,                                             C
               PORDER=1
*
* DDNAME ALLOCATIONS
* SYSLIB WILL USE DEFAULT OF THE TYPE FOR THE SINC MEMBER AND THE
* EXTENT AS DEFINED IN THE PROJECT DEFINITION
*
 FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
 FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000
 FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,RECNUM=5000,DFLTTYP=OBJ
*
*
          FLMTRNSL   CALLNAM='LOOK AT DBRM',                         C
               FUNCTN=BUILD,                                         C
               COMPILE=LOOKDBRM,                                     C
               VERSION=1.0,                                          C
               GOODRC=0,                                             C
               PORDER=1
*
* DDNAME ALLOCATIONS
* SYSLIB WILL USE DBRMTYPE FOLLOWED BY THE TYPE FOR THE SINC MEMBER AND
* THEN THE EXTENT OF THE SINC MEMBER TYPE AS DEFINED IN THE PROJECT
```

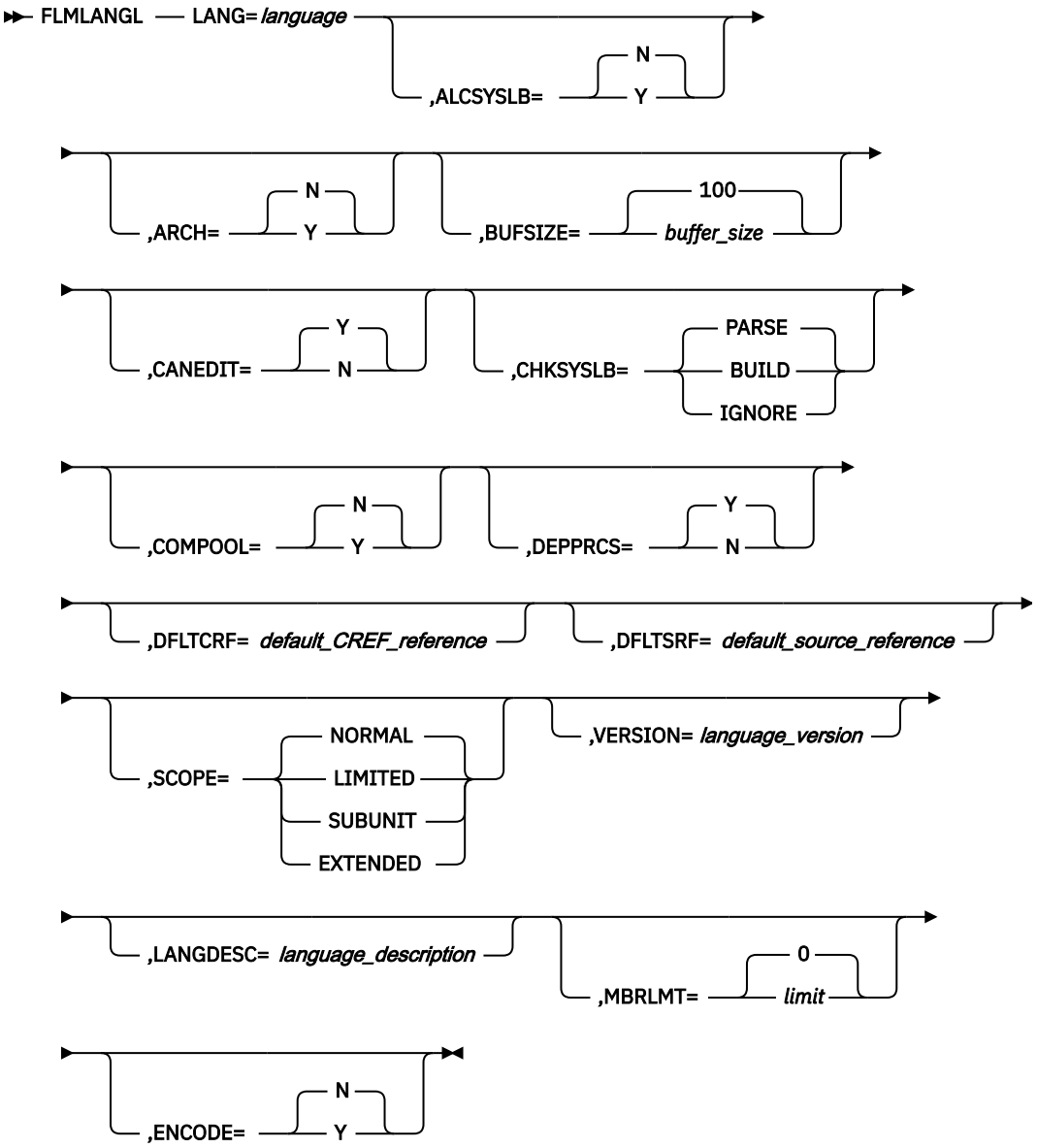
```
* DEFINITION
*
 FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC,INCLS=DBRM
 FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000
 FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,RECNUM=5000,DFLTTYP=OBJ
*
*
          FLMTRNSL   CALLNAM='USE SPECIAL',                           C
               FUNCTN=BUILD,                                          C
               COMPILE=IKJSPECL,                                      C
               VERSION=1.0,                                           C
               GOODRC=0,                                              C
               PORDER=1
*
* DDNAME ALLOCATIONS
* SYSLIB WILL USE COPYBOOK, SOURCE, MACRO, and TOOLS
*
 FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC,INCLS=SPECIAL
 FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000
 FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,RECNUM=5000,DFLTTYP=OBJ
```

FLMLANGL macro

Use this macro to define a language to SCLM. Specify the name of the language and processing characteristics using the keywords supported by this macro. Specify the translators to be invoked for this language by using the FLMTRNSL macro after FLMLANGL.

The order in which data sets of various types are to be allocated for finding includes may be specified by using one or more FLMINCLS macros. The FLMALLOC macros following each FLMTRNSL macro are associated with FLMINCLS macros by use of the INCLS parameter.

Macro format



Parameters

LANG=*language*
A user-specified pseudonym for a language. It can be up to 8 characters. It is stored with the accounting information of editable members. Specify this name when you first define a member to SCLM.

,ALCSYSLB=N|Y
Indicates whether data sets on FLMSYSLB macros are allocated automatically for IOTYPE=I allocations (see "FLMALLOC macro" on page 494). If ALCSYSLB=N, use FLMCPYLB macros for each FLMSYSLB data set on IOTYPE=I allocations. If ALCSYSLB=Y, FLMSYSLB data sets are allocated by build following the allocation of the data sets from the project.

FLMSYSLB data sets are concatenated to the IOTYPE=I allocations for FLMALLOC macros when the FLMALLOC and FLMSYSLB macros both specify the INCLS parameter with the same value. If no INCLS parameter is specified on the FLMSYSLB macro, the FLMSYSLB data sets are concatenated to the FLMALLOC macros with IOTYPE=I and no INCLS parameter.

,ARCH=N|Y
Indicates whether a member parsed in this language is an architecture member. The default is N.

,BUFSIZE=*buffer_size*|100
The number of $list_info records SCLM allocates for a parse, verify, build, copy, or purge translator. The translator returns dependency information in the allocated memory. The default size is 100. The buffer_size must be large enough to accommodate the maximum number of entries returned in $list_info by any translator including one entry for the END record which is always required in a $list_info buffer. SCLM requires one record for each include, change code, user data record, or external dependency the translator returns.

,CANEDIT=Y|N
Indicates whether the language can be assigned to editable members. You should specify language definitions for linkage editors with CANEDIT=N. The default is Y.

,CHKSYSLB=PARSE|BUILD|IGNORE
Indicates when SCLM will check the FLMSYSLB data sets to determine if an include is to be tracked. If CHKSYSLB=PARSE, FLMSYSLB data sets are checked at parse time. Any includes not found in the hierarchy that are in FLMSYSLB data sets are not recorded in the accounting record. If CHKSYSLB=BUILD, FLMSYSLB data sets are checked at build time. Any includes not found in the hierarchy that are in FLMSYSLB data sets are recorded in the accounting record but not in the build map. If CHKSYSLB=IGNORE, any includes not found in the hierarchy at build time are ignored. They are recorded in the accounting record, but are not recorded in the build map. The build translator must determine if includes are missing and generate a return code indicating that the member could not be built.

Use IGNORE for workstation builds when the syslib data sets do not reside in a location that SCLM can check.

IGNORE can also be used to improve performance when your system libraries are fairly stable. By specifying IGNORE, you bypass the overhead of checking all system libraries at either parse or build time. The performance improvement can be significant, particularly when a large number of system libraries is specified in the language definition. The trade-off is that you invoke a translator that will fail when an include is not found. If your system libraries are fairly stable, it might be better to invoke the translator when occasionally an include might be missing, than to search all of the system libraries each time a member is either parsed or built.

,COMPOOL=N|Y
Indicates whether a compool output is required. If COMPOOL=Y is specified, SCLM verifies that a compool output is generated and saved in the hierarchy. SCLM issues a warning message if there is no output identified by the COMP architecture definition keyword.

,DEPPRCS=Y|N
Indicates whether components depending on the member being built are rebuilt if some outputs from the translator were not saved for this member. The default is Y.

,DFLTCRF=*default_CREF_reference*
Identifies the type that is substituted into the @@FLMCRF variable for include-set definitions. The @@FLMCRF variable can be used in the list of types to search for includes. The CREF statement architecture statement can be used to override this value. If both the CREF statement and DFLTCREF parameter are omitted the @@FLMCRF variable is ignored.

The value that is substituted into the @@FLMECR variable for include-set definitions is the extended type of the DFLTCRF type. If there is no extended type for the DFLTCRF type, the @@FLMECR variable is ignored.

,DFLTSRF=*default_source_reference*
A type name that can be used to allocate a hierarchical view. This hierarchical view is typically used by the translator to resolve references to SCLM hierarchy members. This parameter has no effect unless an FLMALLOC macro with IOTYPE=I and KEYREF=SREF is used for the language. SCLM ignores this parameter during a build if a CC, Generic, or LEC architecture definition is used to build the source member.

,SCOPE=LIMITED|NORMAL|SUBUNIT|EXTENDED
Indicates the minimum scope allowed. SCLM compares this parameter with the mode specified as input to build and promote functions to allow or disallow processing. The input mode must be of equal or greater value than the language scope. Valid scope values, in ascending order, are LIMITED, NORMAL, SUBUNIT, and EXTENDED. The default is NORMAL.

,VERSION=*language_version*
The 8-character version name associated with this language. Altering this parameter causes all source members under this language to be rebuilt. If you do not specify it, SCLM sets this parameter to blank.

,LANGDESC=*language_description*
The 40-character description associated with this language. If you do not specify this parameter, SCLM sets it to blank.

,MBRLMT=*limit*|0
Indicates the maximum number of source members that can be present in any input list presented to a translator. SCLM does not exceed the specified MBRLMT value (*limit*). If the default MBRLMT=0 is accepted or specified, there is no limit on the number of source members.

,ENCODE=N|Y
Specifies whether a member with this language is encoded or not. If ENCODE=Y is specified, SCLM decodes the member when it is read and encodes it when it is saved, either in the edit session, or when using the LOCK/PARSE/STORE, MIGRATE, or SAVE SCLM services.

Example 1

The language definition for PASCAL is defined.

```
     FLMLANGL LANG=PASCAL,VERSION=1.0,ALCSYSLB=Y
```

FLMLRBLD macro

The FLMLRBLD macro causes members with a particular language to be rebuilt whenever they are promoted into particular groups. Rebuilding is often necessary when processing changes due to FLMTOPTS or FLMTCOND. The FLMLRBLD macro is only valid within a language definition; it must follow an FLMLANGL.

During the promotion of a member whose language requests a rebuild with the FLMLRBLD macro for that particular group, the build map is not copied during the promote. After the promote completes, the build function is invoked using the 'to group'. The build is conditional and is invoked against the same member that was promoted. Because the build maps will be missing for members having that language, those members, and any dependent members, will be rebuilt. All other members will have the build maps copied, and will not be rebuilt during the conditional build.

If the Promote Copy succeeds, then the build will take place.

Note:

1. There can be multiple FLMLRBLD macros for each language.
2. FLMLRBLD is supported against buildable types. The exception to this rule is Linkage Editor translators such as FLM@L370. Due to special processing that occurs within a Linkage Edit Control translator, FLMLRBLD is not supported, and is ignored.

Macro format

Parameters

GROUP=*group_list*

This parameter specifies the groups at which promoted members will be rebuilt. After a member with the language given on the previous FLMLANGL macro is promoted to one of the listed groups, the member is conditionally rebuilt.

The group list must be enclosed in parentheses or single quotes, with a comma and no spaces between the group names. The list of groups is not checked for validity when the project definition is assembled or during build. This allows alternate project definitions to function without requiring that all groups be defined in the alternate project definition.

If the GROUP= parameter is omitted, no groups are rebuilt on promotion.

Example

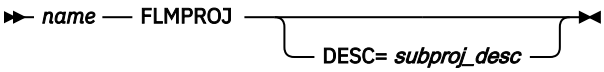
This example shows a part of a language definition of a language that changes translator options at group TEST. The FLMLRBLD macro specifies that members with language COMPLANG will automatically be rebuilt after a promotion.

```
FLMLANGL LANG=COMPLANG,VERSION=1.0,ALCSYSLB=Y
FLMLRBLD  GROUP=(PROD)
FLMTRNSL CALLNAM='Compile',                                X
FUNCTN=BUILD,                                              X
COMPILE=EXAMPLE,                                           X
OPTIONS='ANSI'

FLMTOPTS OPTIONS='ANSI,NODEBUG,OPTIMIZE',                  X
GROUP=(PROD),ACTION=REPLACE
```

FLMPROJ macro

Use this macro to define a subproject to an SCLM project/alternate. This should only be performed if you are going to use SCLM subproject security. For more information on subproject security, see Chapter 17, “SCLM security,” on page 337.

Macro format



Parameters

name

An 8-character subproject name.

DESC=*subproj_desc*

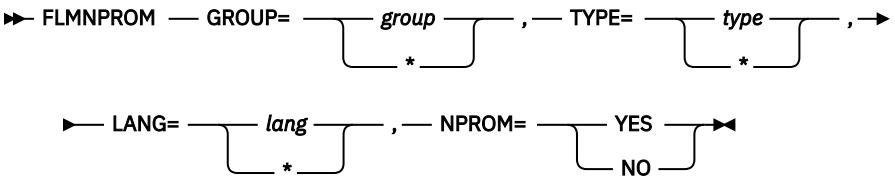
The 40-character description associated with the subproject. If you do not specify this parameter, SCLM sets it to blanks.

FLMNPROM macro

Use this macro to specify which SCLM editable elements may or may not be marked as non-promotable. FLMNPROM can be specified multiple times. If this macro is used, only members that match the groups, types, and languages associated with the NPROM=YES parameter can be marked as non-promotable using one of these methods:

- SCLM option 3.1 Library Utility, command N.
- SCLM option 3.11 Unit of Work, command N.
- The NOPROM service.

Macro format



Parameters

If NPROM=YES is specified, then GROUPS,TYPES, and LANGUAGES can be used to specify which members can be marked as NON-PROMOTABLE. If NPROM=NO is specified, then the GROUPS, TYPES, and LANGUAGES can be used to specify which members cannot be marked as non-promotable.

GROUP=group|*
If NPROM=YES is specified, the members of the group specified can be marked as non-promotable. If NPROM=NO is specified, the names of the members of the group specified may not be marked as non-promotable. Use an asterisk (*) to indicate all groups.

,TYPE=type|*
If NPROM=YES is specified, the members of the type specified can be marked as non-promotable. If NPROM=NO is specified the names of members of the type specified may not be marked as non-promotable. Use an asterisk (*) to indicate all types.

,LANG=lang|*
If NPROM=YES is specified, the language of these members can be marked as non-promotable. If NPROM=NO is specified, the names of the language of these members may not be marked as non-promotable. Use an asterisk (*) to indicate all languages.

NPROM=YES|NO
Specifies whether members in the specified groups and types, and with the specified languages, may or may not be marked as non-promotable. If you specify YES, the NOPROM service can be used to mark identified members as non-promotable. If you specify NO, the NOPROM service cannot be used to mark identified members as non-promotable.

Examples

This example shows how to specify that editable members with a language of COBCOPY in the SOURCE type, in any group, can be marked as non-promotable.

```
FLMNPROM GROUP=*,TYPE=SOURCE,LANG=COBCOPY,NPROM=YES
```

This example shows how to specify that all editable members with a language of COBCOPY in the SOURCE type, in all groups except EMERFIX can be marked as non-promotable.

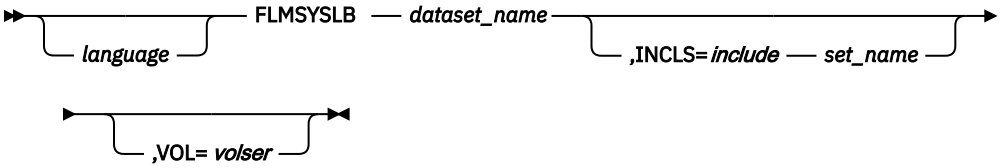
```
FLMNPROM GROUP=*,TYPE=SOURCE,LANG=COBCOPY,NPROM=YES
FLMNPROM GROUP=EMERFIX,TYPE=SOURCE,LANG=COBCOPY,NPROM=NO
```

FLMSYSLB macro

Use this macro to define a set of system macro or include data sets for an include set in a language. The data sets defined by FLMSYSLB contain members that are referenced by SCLM members. Whether or not these include dependencies are tracked is determined by the CHKSYSLB parameter of the language. These data sets also can be allocated for the build translator(s) by using the ALCSYSLB parameter of the language.

Different sequences of data sets may be specified by using the INCLS parameter. FLMSYSLB macros with the INCLS parameter will be used in conjunction with FLMALLOC macros that have IOTYPE=I and an INCLS parameter with the same value. The value of the INCLS parameter is the name of an FLMINCLS macro in the language definition.

Macro format



Parameters

language
An 8-character language name. The language must be the same name as the language specified in the LANG field on the FLMLANGL macro. To specify multiple data sets for a language, omit the language on all but the first data set.

dataset_name
The partitioned data set or member of a fully qualified partitioned data set containing macros or includes from outside the project. The data set name must meet all of the requirements specified by the MVS data set naming conventions. The project definition allows up to 54 characters, including periods and parentheses, to support the specification of a member name. If the data set name is too long (for example, more than 44 characters for a data set name without a member specified), or it does not meet the MVS data set naming conventions, then errors occur during SCLM functions (for example, Parse or Build). The data sets are searched and allocated in the order that they occur in the project definition.

,INCLS=*include_set_name*
This refers to the include-set name on an FLMINCLS macro. When searching for includes, SCLM first checks the types specified on the FLMINCLS macro, followed by the data set on this and other FLMSYSLB macros with the same include-set name. If no INCLS parameter is specified, this FLMSYSLB macro is used for the default include set. All of the FLMSYSLB statements for an include set must be specified together.

,VOL=*volser*
Specifies the serial number of an eligible direct access volume on which the data set is located. This allows reference to a data set that is either uncataloged or that is located on a different volume than the catalog specifies. The default action, if not specified, is to use the volume in the data set's catalog entry.

Note: If an SMS-managed volume is specified, the system will override this specification with the volume in the catalog entry.

Example

This example shows the FLMSYSLB macros that might be included in the project definition for a language that has includes in 3 different include sets.

```
DBAPPL        FLMSYSLB  SYS1.COMPILER.INCLUDES
              FLMSYSLB  SYS1.DATABASE.INCLUDES,INCLS=DATABASE
              FLMSYSLB  SYS1.TRANSACT.INCLUDES,INCLS=DATABASE
              FLMSYSLB  APPL.REUSE.INCLUDES,INCLS=REUSE
```

In this example the includes for members of language DBAPPL will first find their members in the project hierarchy. If the includes are not found in the project hierarchy the FLMSYSLB data sets will be searched. Only the FLMSYSLB data sets that are associated with the same include set as the include will be searched for the include.

For example, in the DBAPPL language definition:

1. FLMALLOC with IOTYPE=I and no INCLS parameter will be associated with SYS1.COMPILER.INCLUDES
2. FLMALLOC with IOTYPE=I and INCLS=DATABASE will be associated with SYS1.DATABASE.INCLUDES concatenated with SYS1.TRANSACT.INCLUDES
3. FLMALLOC with IOTYPE=I and INCLS=REUSE will be associated with APPL.REUSE.INCLUDES
4. FLMALLOC with IOTYPE=I and INCLS=XXXXX will not have an associated FLMSYSLB since there are no FLMSYSLB macros associated with language DBAPPL that have an INCLS parameter with the value XXXXX

Includes in data sets outside of the project definition can either be tracked in the accounting record of the member that includes them or not tracked at all.

To track external includes in the accounting record:

1. Specify CHKSYSLB=BUILD or IGNORE in the language definition.
2. Specify an FLMSYSLB for each data set containing included members.
3. Either specify ALCSYSLB=Y in the language definition, or specify each of the data sets from FLMSYSLB macros on FLMCPYLB macros for the appropriate ddnames.

When CHKSYSLB is BUILD, SCLM checks the FLMSYSLB data sets at build time. When CHKSYSLB is IGNORE, the build translator determines if includes are missing. In either case, any includes not found in the hierarchy are recorded in the accounting record when the member is parsed.

To not track external includes at all:

1. Specify CHKSYSLB=PARSE in the language definition (the default)
2. Specify an FLMSYSLB for each data set containing included members
3. Either specify ALCSYSLB=Y in the language definition, or specify each of the data sets from FLMSYSLB macros on FLMCPYLB macros for the appropriate ddnames.

When CHKSYSLB is PARSE, SCLM verifies at parse time that any includes that are not found in the project hierarchy can be found in the FLMSYSLB data sets. The includes found in the FLMSYSLB data sets are not recorded as includes in the accounting record.

FLMTCOND macro

The FLMTCOND macro provides a means of running or skipping BUILD translators based upon the group at which the BUILD takes place and return codes from previous BUILD translators in the same language definition. The use of FLMTCOND is similar to the use of the COND keyword parameter on a JCL EXEC statement. This similarity is restricted by the requirement that multiple uses of FLMTCOND in a language

definition require each corresponding FLMTRNSL macro to have identical output KEYREF and DFLTTYP keyword values on the FLMALLOC statements.

The FLMTCOND macro can be used to specify a group, combinations of return codes from previous BUILD translators in the same language definition, or both in order to:

- Run one of two BUILD translators
- Run or skip a BUILD translator only under certain conditions
- Run or skip several BUILD translators that have the same outputs

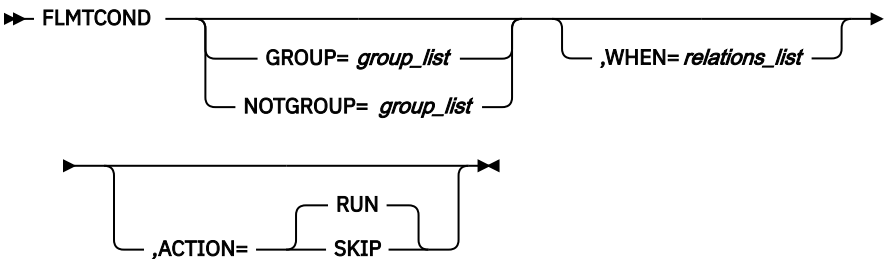
Note:

1. An FLMTCOND macro must follow an FLMTRNSL macro with FUNCTN=BUILD. Only one FLMTCOND macro can be specified for each FLMTRNSL.
2. The GROUP and NOTGROUP parameters are mutually exclusive. If neither GROUP nor NOTGROUP is specified, the relations list and action applies to all groups.
3. FLMTCOND can be used with the GROUP or NOTGROUP parameters to provide an IF-THEN-ELSE effect in which only one of two translators is used. The NOTGROUP keyword can provide flexibility in altering the hierarchy without similar alterations in the language definitions.
4. Use of the FLMTCOND statement does not cause a recompilation when a member is promoted to another group, it only specifies actions to be taken *if* a build is performed at the new group. To cause a rebuild to occur automatically, add an FLMLRBLD statement for the language.

The logic of the GROUP, NOTGROUP, WHEN, and ACTION parameters is shown in this illustration:

```
For specifying GROUP keyword:
  IF the BUILD group is in the group_list
       AND
  WHEN at least one relation is TRUE THEN
       DO ACTION
  ELSE
       DO OTHER ACTION

For specifying NOTGROUP keyword:
  IF the BUILD group is NOT in the group_list
       AND
  WHEN at least one relation is TRUE THEN
       DO ACTION
  ELSE
       DO OTHER ACTION

DEFAULTS:
  GROUP  = ALL GROUPS
  WHEN   = TRUE
  ACTION = RUN
```

Macro format



Parameters

GROUP=*group_list*

This parameter specifies the groups where the relations list and action are used.

The group list must be enclosed in parentheses or single quotes, with a comma and no spaces between the group names. The list of groups is not checked for validity when the project definition is

assembled or during build. This allows alternate project definitions to function without requiring that all groups be defined in the alternate project definition.

The other ACTION is taken for build groups not in the group_list.

GROUP=( ) or GROUP=' ' specifies an empty group list.

NOTGROUP=*group_list*
Use this parameter to specify groups to which you do not want the relations list and action applied.

NOTGROUP=( ) or NOTGROUP=' ' specifies an empty group list.

The format of the group_list is the same as for the GROUP parameter.

,WHEN=*relations_list*
This parameter specifies the conditions under which the translator is run. The default is TRUE.

The relations_list is *(s1,r1,v1)* or *((s1,r1,v1),...,(sn,rn,vn))* where:

- *si* is the translator label for a previous FLMTRNSL macro of a BUILD translator in the same language definition. An asterisk (*) can be used to match the previous FLMTRNSL (with or without a translator label) in the language definition for the BUILD translator that last executed. Using an asterisk allows you to refer back at run time to the BUILD translator that was last executed in the language definition at this point. There is no default.
- *ri* is a standard relation such as EQ, NE, LT, GT, LE, or GE. There is no default.
- *vi* is an unsigned integer with a maximum value of 999999999. This relation is compared with the return code from a previous Build translator identified by *si* in the language definition. There is no default.

At run time, SCLM stops examining the relations in a list as soon as a TRUE relation is found. Incorrect labels in relations that follow a TRUE relation do not result in an error message. The relations in a list can be viewed as Boolean values connected by a Boolean OR. Build translators that have not executed are ignored for *si* = *.

SCLM stops examining previous Build translators for a relation when the label *si* is located even if the Build translator did not execute. The relation is evaluated as FALSE for Build translators that did not execute.

,ACTION=RUN|SKIP
This parameter specifies the action to take at run time.

The decision to RUN or SKIP the translator depends upon the build group, the GROUP|NOTGROUP parameter, and the WHEN parameter.

All FLMTRNSL macros in a language definition that also use FLMTCOND with a WHEN keyword must use the same FLMALLOC KEYREF and DFLTTYP keyword values for all output allocations (KEYREF is OBJ, OUTx, COMP, LIST, LOAD or LMAP). Use of the same keywords results in an architecture definition that is correct for all possible return codes at run time.

When the default architecture definition is constructed, SCLM does not know what the return codes will be at run time. These assumptions are made:

- The WHEN keyword value contains a TRUE relation.
- The FLMTRNSL macros will not be executed.

Example

Code example
Result

not specified
Run the translator for all groups.

FLMTCOND
Run the translator for all groups. This is legal, but has the same effect as not specifying FLMTCOND.

FLMTCOND ACTION=SKIP
Skip the translator for all groups.

FLMTCOND WHEN=(STEP1,EQ,4)
Run the translator for all groups if the return code from the previous translator with label STEP1 equaled 4.

Skip the translator for all groups if the return code from the previous translator with label STEP1 did not equal 4.

FLMTCOND WHEN=(STEP1,EQ,4), ACTION=SKIP
Run the translator for all groups if the return code from the previous translator with label STEP1 did not equal 4.

Skip the translator for all groups if the return code from the previous translator with label STEP1 equaled 4.

FLMTCOND NOTGROUP=(FVT,SVT,PROD)
Run the translator if the group is not FVT, SVT, or PROD.

Skip the translator if the group is FVT, SVT, or PROD.

FLMTCOND NOTGROUP=(FVT,SVT,PROD), ACTION=SKIP
Run the translator if the group is FVT, SVT, or PROD.

Skip the translator if the group is not FVT, SVT, or PROD.

FLMTCOND NOTGROUP=(FVT,SVT,PROD), WHEN=(STEP1,EQ,4)
Run the translator if the group is not FVT, SVT, or PROD and the return code from the previous translator with label STEP1 equaled 4.

Skip the translator if the return code from the previous translator with label STEP1 did not equal 4.

Skip the translator if the group is FVT, SVT, or PROD.

FLMTCOND NOTGROUP=(FVT,SVT,PROD), WHEN=(STEP1,EQ,4), ACTION=SKIP
Run the translator if the return code from the previous translator with label STEP1 did not equal 4 or if the group is FVT, SVT, or PROD.

Skip the translator if the group is not FVT, SVT, or PROD and the return code from the previous translator with label STEP1 equaled 4.

FLMTCOND GROUP=(FVT,SVT,PROD)
Run the translator if the group is FVT, SVT, or PROD.

Skip the translator if the group is not FVT, SVT, or PROD.

FLMTCOND GROUP=(FVT,SVT,PROD), ACTION=SKIP
Run the translator if the group is not FVT, SVT, or PROD.

Skip the translator if the group is FVT, SVT, or PROD.

FLMTCOND GROUP=(FVT,SVT,PROD), WHEN=(STEP1,EQ,4)
Run the translator if the group is FVT, SVT, or PROD and the return code from the previous translator with label STEP1 equaled 4.

Skip the translator if the return code from the previous translator with label STEP1 did not equal 4.

Skip the translator if the group is not FVT, SVT, or PROD.

FLMTCOND GROUP=(FVT,SVT,PROD), WHEN=(STEP1,EQ,4), ACTION=SKIP
Run the translator if the return code from the previous translator with label STEP1 did not equal 4 or if the group is not FVT, SVT, or PROD.

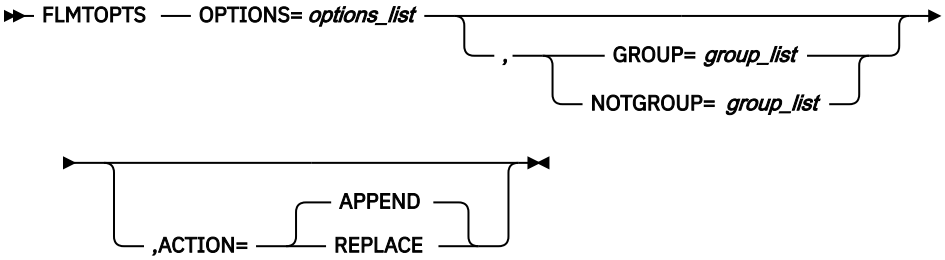
Skip the translator if the group is FVT, SVT, or PROD and the return code from the previous translator with label STEP1 equaled 4.

FLMTOPTS macro

The FLMTOPTS macro is used to vary the options passed to a build translator based on the group where the build is taking place. Options can be appended to the existing options or replace the options completely. FLMTOPTS macros must follow an FLMTRNSL macro with FUNCTN=BUILD. Multiple FLMTOPTS macros can be specified for each FLMTRNSL in which case the FLMTOPTS will be applied in the order they appear in the project definition. The GROUP and NOTGROUP parameters are mutually exclusive. If neither GROUP nor NOTGROUP is specified, the options list and action will apply to all groups.

Note: Use of the FLMTOPTS statement does not cause a recompile when a member is promoted to another group, it only specifies the options to be used **if** a build is performed at the new group. To cause a rebuild to occur automatically, add an FLMLRBLD statement for the language.

Macro format



Parameters

OPTIONS=options_list
This parameter specifies the options that are added to the end of the existing options or replace the existing options. See the OPTIONS parameter on the FLMTRNSL macro for more information about the content and format of options lists.

,GROUP=*group_list*
This parameter specifies the groups where the options list and action are to be applied. If the build group is found in the list then the translator options list will be updated.

The group list must be enclosed in parentheses or single quotes, with a comma and no spaces between the group names. The list of groups is not checked for validity when the project definition is assembled or during build. This allows alternate project definitions to function without requiring that all groups be defined in the alternate project definition.

,NOTGROUP=*group_list*
This parameter specifies the groups where the options list and action are **not** to be applied. If the build group is found in the list then the translator options list will not be updated. The format of the group list is the same as for the GROUP parameter.

,ACTION=APPEND|REPLACE
This parameter specifies how the translator options will be updated.

If APPEND (the default value) is specified the options list will be appended to the end of the existing options list for the translator. No commas or spaces will be added between the existing options list and the options list from the FLMTOPTS macro.

If REPLACE is specified the existing options list will be replaced with the options list from the FLMTOPTS macro.

Example

This example shows a part of a language definition that contains an FLMTRNSL followed by multiple FLMTOPTS macros. The options passed to the translator will vary by the group where the build is taking place. If options were specified in an architecture definition member, they would be added to the end of the options shown here.

Build Group
Options passed to translator

PROD
ANSI,NODEBUG,OPTIMIZE,LIST

TEST
ANSI,OPTIMIZE,LIST

INT
ANSI,LIST

PERFTEST
ANSI,OPTIMIZE,TIMER,LIST

others
ANSI,DEBUG,NOOPTIMIZE,LIST

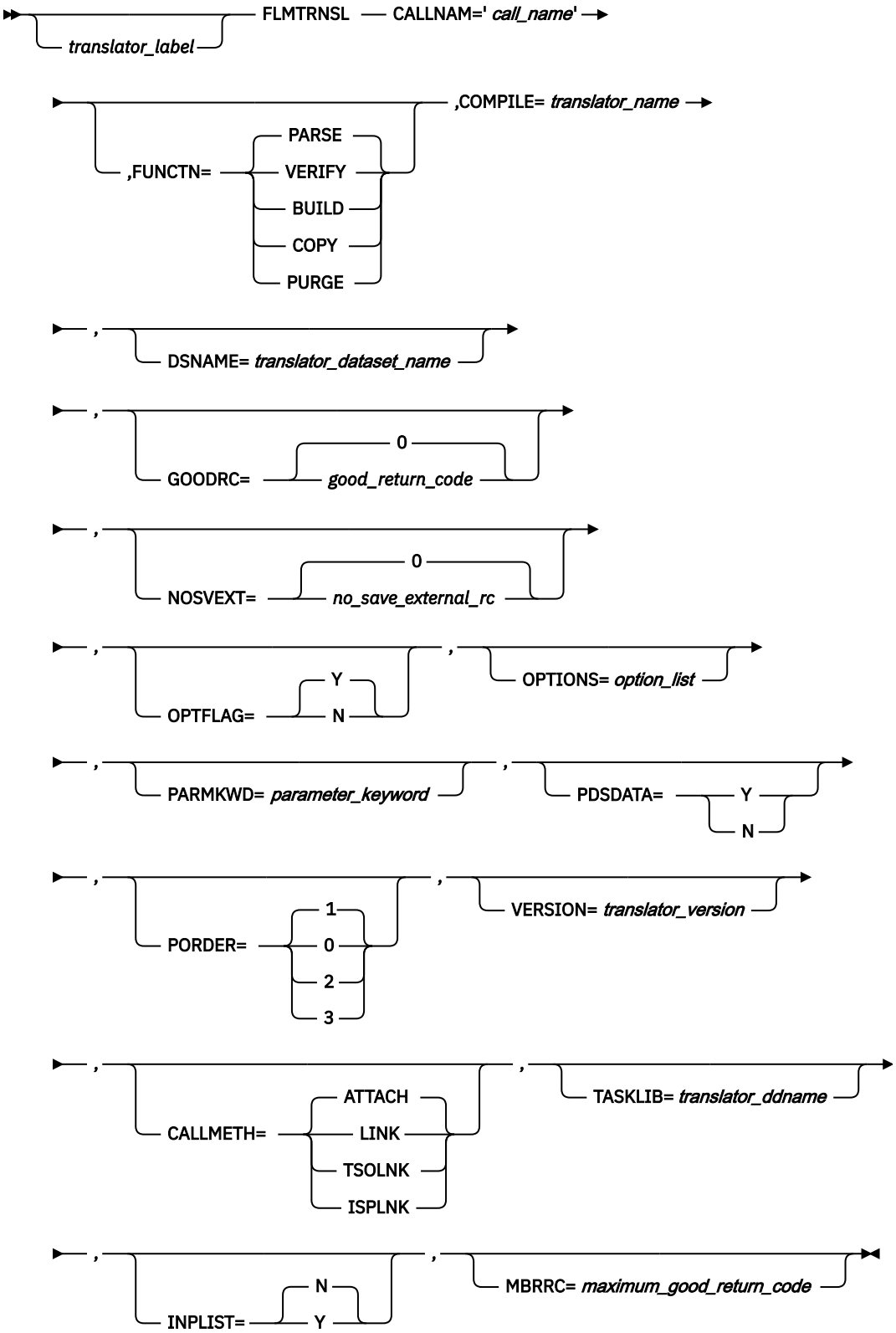
```
         FLMTRNSL CALLNAM='Compile',                                  X
               FUNCTN=BUILD,                                          X
               COMPILE=EXAMPLE,                                       X
               OPTIONS='ANSI'
*
         FLMTOPTS OPTIONS=',DEBUG,NOOPTIMIZE',                        X
               NOTGROUP=(PROD,TEST,INT),ACTION=APPEND
*
         FLMTOPTS OPTIONS='ANSI,OPTIMIZE,TIMER',                      X
               GROUP=(PERFTEST),ACTION=REPLACE
*
         FLMTOPTS OPTIONS='ANSI,NODEBUG,OPTIMIZE',                    X
               GROUP=(PROD),ACTION=REPLACE
*
         FLMTOPTS OPTIONS=',OPTIMIZE',                                X
               GROUP=(TEST)
*
         FLMTOPTS OPTIONS=',LIST'
```

FLMTRNSL macro

This macro serves a function similar to JCL EXECUTE (EXEC) statements in your procedure libraries. Several keyword parameters in this macro contain data identical to your procedures.

Use this macro once for each translator to be invoked for a language. Specify the translator load module name, translator load data set name, version of the translator, and translator options using this macro's parameters.

Macro format



Notes:

[1] Comma separators are only required up to and before the last optional parameter specified.

Parameters

translator_label
A 1- to 8-character string containing no blanks or commas. The translator label is used by the FLMTCOND macro to identify build translators in a language definition to examine their return codes at run time for conditional execution of build translators.

CALLNAM='*call_name*'
The name of the translator with a maximum of 16 characters. This name appears in SCLM messages along with translator return codes. If you want embedded blanks in the call name, surround the string with single quotes.

,FUNCTN=PARSE|VERIFY|BUILD|COPY|PURGE
Identifies the function that the translator performs. The default is PARSE.

- A parse translator gathers statistics and dependencies. Parse translators run during migration, when the member is saved in an edit session, or when the SAVE or PARSE service is called. A parse translator can also be used to define user data and change codes for the member.
- A verify translator can be used to perform validation in addition to default SCLM validation. The verify translator can be used to check the change codes or user data defined for members. Another example could be verification of data that is related to an SCLM-controlled member but is not under SCLM control itself. Verify translators run during build and promote verification.

  For builds, SCLM invokes a verify translator to verify inputs to build translators. For example, when an LEC architecture definition is being built, the source member is verified before compiling and the object member is verified before linking.

  For promotes, SCLM invokes a verify translator to verify build inputs and outputs. For example, when an LEC architecture definition is being promoted, the source, object, and load members are verified before the promote copy phase.
- A build translator can assemble, compile, link, or otherwise process a member so that the outputs have different formats than the inputs. For example, building a COBOL source program generates a listing and an object module.
- A copy translator is invoked when Promote copies an SCLM-controlled member to the next group in the hierarchy. Copy translators are invoked before Promote copies the SCLM-controlled member. If the copy translators for a member fail, Promote does not attempt to copy the controlled member. Copy translators can be used to copy data that is related to an SCLM-controlled member but is not under SCLM control itself.
- Purge translators can be used to purge data that is related to an SCLM-controlled member but is not under SCLM control itself. Purge translators are invoked whenever SCLM performs a delete operation on an SCLM-controlled member during build or promote.

,COMPILE=*translator_name*
The name of the program that is invoked by the translator. This would normally be a parser, compiler, assembler, or a user-supplied routine.

For CALLMETH of ATTACH, LINK, or TSOLNK, this is the name of a REXX exec or CLIST, or the entry point to a load module. For a CALLMETH of ISPLNK, this must have a value of SELECT.

,DSNAME=*translator_dataset_name*
The name of the data set containing the translator load module (COMPILE parameter) or REXX exec being invoked. The data set name parameter is not required with the translator load module that resides in the system concatenation. Use the TASKLIB parameter to specify additional libraries to be searched. SCLM looks in the data set specified by the DSNAME parameter first, followed by the data sets allocated to the TASKLIB ddname, if specified, and then follows the normal MVS search order. The data set name can be up to 44 characters.

Note: The DSNAME in the translate step is ignored when CALLMETH is ISPLNK. For REXX and CLIST, make sure that the required EXEC or CLIST is in the SYSEXEC or SYSPROC concatenation. For programs, make sure that the load module is allocated in ISPLLIB or STEPLIB.

,GOODRC=*good_return_code*|0
Definition of an acceptable return code from the translator that must be a positive integer or 0. If you get a return code value greater than *good_return_code* from a translator, the process has failed, and no accounting information is saved in the hierarchy. The default is 0. CALLMETH=TSOLNK will result in a return code equal to the translator return code for normal completion, the abend code from the translator, or a 40 in the event of an IKJEFTSR failure.

,NOSVEXT=*no_save_external_rc*|0
A return code value indicating whether any translator outputs targeted to an external data set were saved (valid for FUNCTN=BUILD). Use this parameter with the DEPPRCS parameter on the FLMLANGL macro. It allows or disallows dependency processing if you save some outputs produced by the translator.

The build processor determines that external outputs were not saved by the translator if *no_save_external_rc* is equal to a translator return code other than zero. The default is 0.

,OPTFLAG=N|Y
Indicates whether developers can override default translator options. The default is Y. This parameter has no effect if you specify OPTOVER=N on the FLMCNTRL macro.

,OPTIONS=*option_list*
A list of options passed to the program specified in the COMPILE parameter. For example, if COMPILE=FLMLPGEN, you can specify in the OPTIONS field any of the parameters that are supported by FLMLPGEN.

Delimit the list with single quotes or parentheses. The options can also contain variables to provide dynamic information to the COMPILE program. The maximum length is 255 characters, including delimiters. The @@FLMMBR and @@FLMTYP variables will be replaced with the name of the member and type of the last SINC, INCL, or INCLD statement in the architecture definition. If a source member is being built, it will be the name of the source member. See Chapter 23, “SCLM Variables and Metavariables,” on page 631. Also see the PARMKWD parameter for more information about options.

The IBM linkage editor requires that the DCBS option parameter be passed in order for the SYSLMOD block size to be used in creating load modules. If the DCBS option is not specified, the linkage editor creates load modules using the maximum record size for the device type. Use the OPTIONS= parameter on the FLMTRNSL macro to pass the DCBS option. Failure to do so can result in message FLM44507 RC4.

The CALLMETH of ISPLNK requires that the option string contain the keywords and parameters for the ISPF SELECT service. The options must be in the format expected by the ISPF SELECT service. For a description of the ISPF SELECT service, refer to the *z/OS ISPF Services Guide*.

,PARMKWD=*parameter_keyword*
The keyword (PARM0..PARM9) used in architecture members to specify additional options for this translator.

Note: The complete options list passed to the translator has a maximum length of 512 characters and has the format:

```
 string1
,string2
,string3
```

where

string1
is the options from the OPTIONS parameter on the FLMTRNSL macro.

string2
is the options from the PARM statements in the architecture definition.

string3
is the options from the PARMx statements in the architecture separated by commas.

Extraneous blanks are removed by SCLM.

,PDSDATA=Y|N
Specifies whether the input members for this translator reside in SCLM-controlled partitioned data sets. The value of this parameter must be Y for parse and build translators.

If this parameter is not specified, the default varies according to translator function type. The default for parse, build, and verify translators is Y; the default for copy and purge translators is N.

If multiple translators are defined for copy and purge functions, you must not specify Y for one translator and N for another.

Note: SCLM PROMOTE will only invoke Copy and Purge translators for SCLM-controlled partitioned data set members if PDSDATA is set to Y. Copy and Purge translators that operate on nonpartitioned data set controlled parts (such as CSP MSLs) must have PDSDATA set to N.

,PORDER=0|1|2|3
An integer indicating the parameter order to the translator. The translator parameter order must be an integer from 0 to 3. The default is 1. SCLM can pass two kinds of parameters to the translator: the option list and the ddname substitution list. The option list contains the translator options (OPTIONS parameter) concatenated with the options specified in the architecture member (see PARMKWD parameter). The ddname substitution list contains the ddnames specified for allocation. See the DDNAME parameter of “FLMALLOC macro” on page 494. This list defines the valid values for the translator parameter order:

0
No parameters passed

1
Pass option list

2
Pass ddname substitution list

3
Pass option list followed by ddname substitution list

Ddname substitution lists are a feature of many translators (such as precompilers, utilities, assemblers, and compilers). The correct format of a ddname substitution list is usually unique for each translator and can be located in the programming guide for the translator.

The ddname substitution list is a string of ddnames allocated for the translator. The ddnames appear on the substitution list in the order specified by the FLMALLOC macros in the language definition. See the topic, "Invoking Utility Programs from an Application Program" in *z/OS DFSMSdfp Utilities* for general information about ddname substitution lists. See the manuals for the specific translator being invoked for details on the substitution list contents expected. For IBM supplied compilers, this information is located in the compiler's Programmers Guide under "Invoking Compiler from Application Programs" or "Dynamic Invocation of Compiler". SCLM limits the size of the ddname substitution list to 512 characters or 64 ddnames.

,VERSION=*translator_version*
An 8-character representation of the translator version. This parameter is informational only. SCLM stores this parameter in the account record for each output member saved from the translators. If you do not specify this parameter, SCLM sets it to blank.

,CALLMETH=ATTACH|LINK|TSOLNK|ISPLNK
Indicates whether the translator is to be linked, attached, or invoked by the TSO service facility routine or called through ISPF services. Use ATTACH for load modules unless you need access to ISPF variables or services; in that case, use LINK. Using LINK can result in loops or out-of-space abends because storage is not freed between calls to the translators. The default is ATTACH.

TSOLNK is for translators written as REXX execs. TSOLNK results in the translator being invoked from IKJEFTSR (TSO service facility routine) with parameter 1 of x'00010001'. This parameter indicates that the TSO service facility should invoke the requested translator from an unauthorized environment and that the translator can be a TSO command, REXX exec, or CLIST.

ISPLNK is for translators that must have access to ISPF variables or services. The value specified on the COMPILE parameter is the ISPF service that is used to call the translator. The only supported value is SELECT. The keywords, including the command to run, are specified in the OPTIONS parameter. The name of the load module, CLIST, REXX exec or other command is specified in the OPTIONS parameter.

This example shows the CALLMETH, COMPILE, and OPTIONS parameters on an FLMTRNSL macro used to run the FLMLRC37 parser using ISPLNK:

```
         FLMTRNSL   CALLNAM='C PARSE',                                 C
               FUNCTN=PARSE,                                           C
               CALLMETH=ISPLNK,                                        C
               COMPILE=SELECT,                                         C
               PORDER=1,                                               C
               OPTIONS='PGM(FLMLRC37) PARM(STATINFO=@@FLMSTP,LISTINFO=@C
               @FLMLIS,LISTSIZE=@@FLMSIZ)'
```

,TASKLIB=*translator_ddname*

The ddname associated with one or more data sets that contain the translator load module. The data sets must be specified using an FLMALLOC macro with DDNAME=translator_ddname. When specified for a translator using a DDNAME substitution list, the TASKLIB allocation does not appear in the list passed to the translator.

TASKLIB is only valid for CALLMETH=ATTACH. The operating system searches for executable members in the specified DSNAME parameter, then in the TASKLIB concatenation, and then in the system concatenation.

,INPLIST=N|Y

Indicates that this translator supports Input List Processing. You must specify INPLIST=Y to use the IBM Ada/370 Compiler input list function.

,MBRRC=*maximum_good_return_code*

Use this parameter with the INPLIST parameter. MBRRC indicates the maximum value of the *good_return_code* for each member in the Input List. This parameter is similar in function to the GOODRC parameter for the translator. However, the GOODRC parameter applies to a single return code supplied by the translator. The MBRRC parameter indicates the maximum valid value for any member of the Input List.

Example

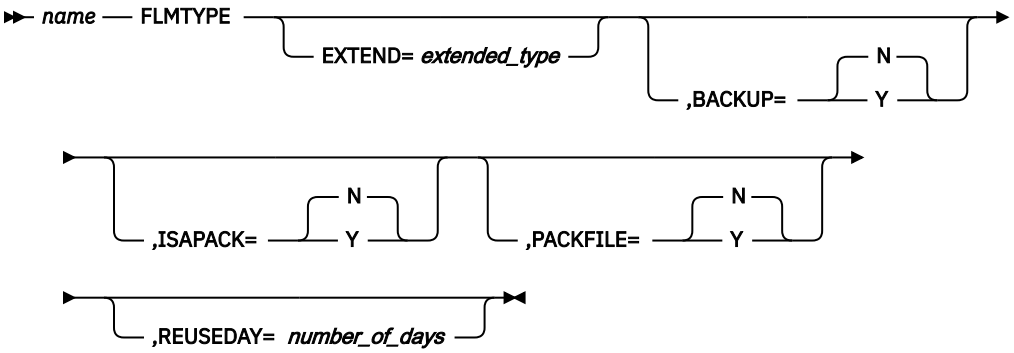
A translator for the Pascal compiler is defined. The compiler is member PASCALVS in data set SYS2.VSPASCAL.LOAD. The translator can only be invoked by the build processor (FUNCTN=BUILD). The build processor refers to the compiler by its call name, PASCAL COMPILER. Only the option list can be passed to the translator (PORDER=1). The default options for this translator are specified by the OPTIONS parameter. Build considers any translator return code greater than 0 as an error (GOODRC=0).

```
      FLMTRNSL CALLNAM='PASCAL COMPILER',                            X
            FUNCTN=BUILD,                                            X
            COMPILE=PASCALVS,                                        X
            DSNAME=SYS2.VSPASCAL.LOAD,                               X
            VERSION=1.0,                                             X
            GOODRC=0,                                                X
            PORDER=1,                                                X
            OPTIONS='NOXREF,CHECK,LINECOUNT(75),NOOPT'
```

FLMTYPE macro

Use this macro to define each type in the project definition. This macro is required and can be used multiple times in a project definition.

Macro format



Parameters

name
An 8-character type name.

EXTEND=*extended_type*
An 8-character name that can be used as an alternate type when resolving include dependencies.

The type specified for the EXTEND parameter is substituted into the @@FLMETP variable on FLMINCLS macros in language definitions. @@FLMETP is used to define the types that are searched to find included members.

,BACKUP=N|Y
Specifies that during the package promote process these types of files are to be backed up. If BACKUP=Y, you cannot also specify PACKFILE=Y.

,ISAPACK=N|Y
Specifies the high-level package file. This should be placed on the high-level architecture definition file that is used by the package backout facility to back up the required module. If ISAPACK=Y, you cannot also specify PACKFILE=Y.

Note: If a module is migrated using an ARCHDEF member that does not have this flag specified, SCLM will not back up any modules.

,PACKFILE=N|Y
Specifies that this is the file type where the package backout information is stored. This file has the attributes RECFM=FB and LRECL=130. If PACKFILE=Y, you cannot also specify BACKUP=Y or ISAPACK=Y.

,REUSEDAY=*nnnn*
Use this flag with the PACKFILE parameter. It specifies the number of days a package can be reused. If the package is older than this value the package and its details will be deleted before the promote.

Example

Six types are defined. Type SOURCE2 is an extension of type SOURCE. In SCLM, if a member exists in type SOURCE, its included dependencies can exist in either SOURCE or SOURCE2.

```
OBJ      FLMTYPE
LIST     FLMTYPE
LMAP     FLMTYPE
LOAD     FLMTYPE
SOURCE   FLMTYPE EXTEND=SOURCE2
SOURCE2  FLMTYPE
```

Chapter 22. SCLM translators

This chapter describes the translators provided with SCLM. The translators are used in language definitions that are included with SCLM. You can modify these language definitions for the specific needs of your site and environment. The supplied parsers might not recognize all syntax rules for a specific language, and you might have to modify statements to adhere to generic syntax.

- "FLMCSPDB DB2 Bind/Free translator" on page 566
- "FLMDTLC DTL Processor Build translator" on page 570
- "FLMLPCBL COBOL Parser" on page 571
- "FLMLPFRT FORTRAN Parser" on page 574
- "FLMLPGEN General Purpose Parser" on page 578
- "FLMLRASM REXX Assembler Parser" on page 583
- "FLMLRCBL REXX COBOL Parser" on page 587
- "FLMLRCIS MVS C/C++ parser with include set support" on page 590
- "FLMLRC2 C, C++, and Resource file parser for workstation source" on page 594
- "FLMLRC37 REXX C370 Parser" on page 597
- "FLMLRDTL REXX DTL Parser" on page 601
- "FLMLRIPF Script and OS/2 IPF Source Parser" on page 601
- "FLMLSS General Purpose Parser" on page 605
- "FLMLTWST Workstation Build translator" on page 608
- "FLMTBMAP Build Map Print - Build translator" on page 619
- FLMTCIDS, FLMTCVER, FLMTCLGT, FLMTCPC, FLMTCPP, FLMTCCPS CPS translators:

  SCLM CSP programs are shipped for historical reasons to allow SCLM customers to use the CSP within SCLM. However, since CSP is no longer supported by IBM, support for these SCLM CSP programs is not provided:

  FLMTCIDS
  Initialise dummy data sets

  FLMTCVER
  CSP verify translator

  FLMTCLGT
  CSP LIST(A) and generate translator

  FLMTCPC
  CSP Promote copy translator

  FLMTCPP
  CSP Promote purge translator

  FLMTCCPS
  CSP translator to copy from a PDS member into a sequential file
- "FLMTMJI Interface to JOVIAL Compiler" on page 621
- "FLMTMMI Interface to DFSUNUB0 (phase 2 of MFSUTL and MFSTEST)" on page 621
- "FLMTMSI Interface to SCRIPT/VS" on page 622
- "FLMTPRE" on page 623
- "FLMTPST" on page 625
- "FLMTXFER Workstation Transfer - Build translator" on page 627

There are five types of translators:

© Copyright IBM Corp. 1990, 2019

- A parse translator gathers statistics and dependencies. Parse translators run during migration, when the member is saved in an edit session, or when the SAVE or PARSE service is called. A parse translator can also be used to define user data and change codes for the member.
- A verify translator can be used to perform validation in addition to default SCLM validation. The verify translator can be used to check the change codes or user data defined for members. Another example could be verification of data that is related to an SCLM-controlled member but is not under SCLM control itself. Verify translators run during build and promote verification.

  For builds, SCLM invokes a verify translator to verify inputs to build translators. For example, when an LEC architecture definition is being built, the source member is verified before compiling and the object member is verified before linking.

  For promotes, SCLM invokes a verify translator to verify build inputs and outputs. For example, when an LEC architecture definition is being promoted, the source, object, and load members are verified before the promote copy phase.
- A build translator can assemble, compile, link, or otherwise process a member so that the outputs have different formats than the inputs. For example, building a COBOL source program generates a listing and an object module.
- A copy translator is invoked when Promote copies an SCLM-controlled member to the next group in the hierarchy. Copy translators are invoked before Promote copies the SCLM-controlled member. If the copy translators for a member fail, Promote does not attempt to copy the controlled member. Copy translators can be used to copy data that is related to an SCLM-controlled member but is not under SCLM control itself.
- Purge translators can be used to purge data that is related to an SCLM-controlled member but is not under SCLM control itself. Purge translators are invoked whenever SCLM performs a delete operation on an SCLM-controlled member during build or promote.

FLMCSPDB DB2 Bind/Free translator

Purpose

This is the DB2 Bind/Free translator to be used for binding and freeing DB2 plans. It is necessary to create a DB2 CLIST that will specify the DBRMs to be bound with the DB2 plan, as well as the name of the DB2 plan. An example of these can be found in Chapter 14, “SCLM support for workstation builds,” on page 307.

Parameters

The following guidelines apply when specifying parameters:

- The order of the parameters is not important.
- See the language definitions provided by SCLM for the actual usage of the parameters for the translator.

The following keyword parameters are expected as input for the translator:

ALTPROJ
The variable name for the alternate project name. This parameter is required, and must be set to @@FLMALT.

DBRMTYPE
The type name where the DBRMs are stored. This parameter is required.

Note: The FLMDBALC CLIST is run by the FLMCSPDB translator to allocate the DBRM type to the DBRMLIB ddname.

FUNCTN
The SCLM function invoking the translator: BUILD, COPY, or PURGE. This parameter is required.

GROUP
The variable name for the build group name. This parameter is required, and must be set to @@FLMGRP.

MEMBER
The variable name for the DBRM member name. This parameter is required, and must be set to @@FLMMBR.

OPTION
The operation to be performed with the DB2 Plan: BIND or FREE. This parameter is required.

PROJECT
The variable name for the project name. This parameter is required, and must be set to @@FLMPRJ.

SCLMINFO
The variable name for the SCLM internal pointer. This parameter is required, and must be set to @@FLMINF.

TOGROUP
The variable name for the group to promote to. This parameter is required, and must be set to @@FLMTOG. This variable is ignored for FUNCTN=BUILD.

Return codes

0

Explanation

Success

User response

None.

Project manager response

None.

4

Explanation

A WARNING message was produced.

User response

Look at the message. Fix the problem if necessary.

Project manager response

None.

01300

Explanation

The FUNCTN parameter is not specified correctly in the input options defined for the translator. This parameter is one of the OPTIONS parameters for this translator. This parameter is passed to the translator by way of the OPTIONS= parameter for calls to the FLMCSPDB translator. This parameter is not to be confused with the FUNCTN= parameter passed to SCLM using the FLMTRNSL macro; rather, it is a secondary parameter value for the OPTIONS= parameter that is passed to the translator using the FLMTRNSL macro.

Module

FLMCSPDB

User response

None.

Project manager response

Verify that "OPTIONS=(FUNCTN=BUILD...)" or "OPTIONS=(FUNCTN=PROMOTE...)" is specified for the FLMTRNSL macro invoking the FLMCSPDB translator.

01310

Explanation

The OPTION parameter is not specified correctly in the input options defined for the translator.

Module

FLMCSPDB

User response

None.

Project manager response

Verify that "OPTION=BIND" or "OPTION=FREE" is specified as an option for the translator in the language definition.

01320

Explanation

The GROUP parameter is not specified in the input options defined for the translator.

Module

FLMCSPDB

User response

None.

Project manager response

Verify that "GROUP=@@FLMGRP" is specified as an option for the translator in the language definition.

01330

Explanation

The TOGROUP parameter is not specified in the input options defined for the translator.

Module

FLMCSPDB

User response

None.

Project manager response

Verify that "TOGROUP=@@FLMTOG" is specified as an option for the translator in the language definition.

01340

Explanation

The MEMBER parameter is not specified as a PARM input for the translator.

Module

FLMCSPDB

User response

None.

Project manager response

Verify that "MEMBER=@@FLMMBR" is specified as an option for the translator in the language definition.

Verify that OPTOVER=Y on the FLMCNTRL macro in the project definition.

01350

Explanation

The SCLMINFO parameter is not specified in the input options defined for the translator.

Module

FLMCSPDB

User response

None.

Project manager response

Verify that "SCLMINFO=@@FLMINF" is specified as an option for the translator in the language definition.

01360

Explanation

The DBRMTYPE parameter is not specified in the input options defined for the translator.

Module

FLMCSPDB

User response

None.

Project manager response

Verify that "DBRMTYPE=dbrmtype" is specified as an option for the translator in the language definition. Where dbrmtype has been defined as a valid type in the project definition.

01370

Explanation

An error occurred while executing the DB2 CLIST member.

Module

FLMCSPDB

User response

Verify that DB2 is installed and that you have invoked it correctly from the DB2 CLIST. Trace the execution of your CLIST to verify the return code.

Project manager response

None.

01380

Explanation

The DB2 CLIST member does not contain DSN commands for the group being processed.

Module

FLMCSPDB

User response

Verify that the DB2 CLIST member has code specifying the DSN commands required for the group being processed.

Project manager response

None.

01390

Explanation

The PROJECT parameter is not specified in the input options defined for the translator.

Module

FLMCSPDB

User response

None.

Project manager response

Verify that "PROJECT=@@FLMPRJ" is specified as an option for the translator in the language definition.

01420

Explanation

The data set allocation (DSALLOC) service failed.

Module

FLMCSPDB

User response

Contact the project manager.

Project manager response

Verify that SCLM skeleton FLMLIBS has all necessary allocations for the CSP/AD and/or DB2 products.

21310

Explanation

SCLM received an error initializing the DB2 CLIST.

User response

None.

Project manager response

Verify that the DB2 CLIST exists.

21320

Explanation

SCLM received an error initializing the DB2 OUT data set.

User response

None.

Project manager response

Verify that the DB2 OUT data set exists.

21330

Explanation

SCLM received an error when it attempted to copy the DB2 CLIST to the DB2 OUT data set.

User response

None.

Project manager response

Verify that the DB2 OUT data set attributes are complementary with the DB2 CLIST data set.

21340

Explanation

SCLM could not find the FLMPROXY ddname.

User response

None.

Project manager response

Verify that the FLMPROXY ddname was passed to the translator.

21370

Explanation

Non-key groups are not supported.

User response

None.

Project manager response

Delete non-key groups or re-specify non-key groups as key groups in the project definition. See return code 30108 for more detail.

FLMDTLC DTL Processor Build translator

Purpose

This is the DTL Processor Build translator. It is called from SCLM builds of ISPF Dialog Tag Language to invoke the DTL Conversion Utility.

Parameters

The following parameters are expected as input for the translator:

- Project level qualifier of source data set
- Group level qualifier for the build level of the source data set
- Type level qualifier for the source data set
- Member name of source DTL
- ISPF application-id.

Here are the outputs expected

- Log listing with ISPDTLC information (FLMDTLC)
- Generated panel ($PANELS)
- Generated message member ($MSGS)
- Generated keylist member or command table ($TABLES).

Return codes

0

Explanation

Indicates successful completion.

User response

None.

Project manager response

None.

4

Explanation

DTL completed with a return code of 8.

User response

Only warnings were found during the DTL conversion. Check the DTL source code.

Project manager response

None.

8

Explanation

DTL completed with a return code of 16.

User response

At least one error was found during DTL conversion. Check the DTL source code.

Project manager response

None.

16

Explanation

DTL returned a return code of 20. Fatal error.

User response

Check the DTL source code.

Project manager response

None.

20

Explanation

DTL returned an unknown return code.

User response

Contact the project manager.

Project manager response

Review the messages in the DTL log. Contact IBM support if necessary.

FLMLPCBL COBOL Parser

Purpose

This is the COBOL parser translator that parses the source identified by the SOURCE DDNAME.

Functions

One of the functions of an SCLM parser is to determine all of a module's dependencies. FLMLPCBL determines all of the names that will be copied into the COBOL source.

FLMLPCBL examines each line of the member. Lines located in the IDENTIFICATION DIVISION (ID DIVISION) will not be examined for COPY statements or quoted strings. This will permit the use of a comment entry for each paragraph in the ID DIVISION without the need for an asterisk or slash in column 7.

The parser uses the following syntax rules to locate dependency names outside of the ID DIVISION:

- The search for tokens is restricted to columns 8-72. Column 7 is ignored except when it contains * and / (treated as a comment line) or - (treated as a concatenation). The use of - to concatenate strings for forming reserved words or dependency names is not supported.
- DBCS strings (delimited by shift-out and shift-in characters) in comments and quotes are allowed. In addition DBCS user-defined names are allowed.
- When a line that is not a comment line or a continuation line has COPY after column 7, the next token is taken as the name of a dependency. Note the following exceptions:
  - If the member name is enclosed in single or double quotes, the quotes are ignored.
  - When a line that is not a comment line or a continuation line has EXEC, SQL, and INCLUDE as its first three tokens after column 7, the next token is taken as the name of a dependency. SQLCA and SQLDA are not flagged as external dependencies.
  - When searching for the next token on a line and there are no more tokens on that line, the search continues with the next uncommented line.
  - Tokens inside quoted strings will be ignored (except for COPY member names). Reserved words inside quoted strings will not be counted as statements. COPY, EXEC SQL, and EXEC CICS* inside quoted strings will be ignored.
  - If a token is longer than 8 characters, it will not be added as a dependency.

FLMLPCBL recognizes COPY MEMBER where MEMBER is a 1-character to 8-character string with no separator periods. A separator period is *not* required after MEMBER.

COPY and MEMBER must be on the same line or on a continued line. However, splitting COPY or MEMBER by using a hyphen (-) in column 7 of the continued line is not supported. This is important to consider when using code generators that use the hyphen in column 7 to concatenate strings to form keywords and text names. Use of the hyphen to concatenate strings in order to form a MEMBER as in COPY MEMBER results in the COPY being ignored by FLMLPCBL. Use of the hyphen on the line after COPY when COPY is the last token on the line results in the COPY being ignored by FLMLPCBL.

FLMLPCBL can parse an odd number of quote delimiters if the first two nonblank characters after column 7 on a continuation line are two quote delimiters. FLMLPCBL expects to find an even number of quote delimiters in a literal. You might need to introduce a constant with a literal value that is also continued on the next line to produce an even number of quote delimiters. If you have an odd number of quote delimiters, the dependencies following the odd number of delimiters might be ignored. The following example illustrates a statement with an odd number of quote delimiters:

```
----+-*--1----+----2----+----3----+----4----+----5----+----6----+----7--
        01 TXT-COMMENT PIC X(160)
123456       VALUE 'THIS LITERAL HAS A QUOTE IN COLUMN 72 AND THE NEXT '
123457-     '' 2 QUOTE DELIMITERS RESULT IN AN ODD NUMBER OF QUOTE
123458-     'DELIMITERS FOR THIS STATEMENT '.
```

It is recommended to code DBCS and mixed SBCS and DBCS literals on a separate line especially when the literal contains blanks. For example you can reduce the chance of some of your dependencies being ignored by splitting a line like this:

```
001000              VALUE 'MQ GET/BROWSE çWáÔà{ñ .« '.
FFFFFF44444444444444ECDEC47DD4CCE6CDDEEC04E4E4C4448074444444444
0010000000000000000513450D4807531296625E865B4091BAFDB000000000
```

into two

```
001000              VALUE
FFFFFF4444444444444ECDEC4444444444444444444444444444444444444444
0010000000000000000513450000000000000000000000000000000000000000
```

and

```
001001                    'MQ GET/BROWSE çWáÔà{ñ .« '.
FFFFFF444444444444444444447DD4CCE6CDDEEC04E4E4C4448074444444444
00100100000000000000000000D4807531296625E865B4091BAFDB000000000
```

The parser also gathers statistics or metrics for each module to be parsed. SCLM saves 10 statistics, but this parser only generates 7. For COBOL, this parser defines the following statistics:

Total lines
The total number of records in the file.

Comment lines
The number of lines with a slash (/) or an asterisk (*) in column 7.

Noncomment lines
The number of total lines minus the number of comment lines.

Blank lines
The number of lines that contain only blanks after column 6. Any sequence numbers in the rightmost columns of the line are ignored.

Prolog lines
The number of comment lines that are found before the first noncomment line.

Total statements
The number of lines that are not comment or continuation lines whose first token after column 7 is one of the following reserved words:

```
   ACCEPT       DIVIDE       INSPECT      REWRITE
   ADD          ENTER        MERGE        SEARCH
```

```
ALTER        ENTRY        MOVE         SET
CALL         EVALUATE     MULTIPLY     SORT
CANCEL       EXAMINE      NOTE         START
CLOSE        EXIT         ON           STOP
COMPUTE      GO           OPEN         STRING
CONTINUE     GOBACK       PERFORM      SUBTRACT
COPY         GOTO         READ         TRANSFORM
DELETE       IF           RELEASE      UNSTRING
DISPLAY      INITIALIZE   RETURN       WRITE
```

In addition, any EXEC SQL and EXEC CICS statements are treated as program statements.

Comment statements
This value is always 0.

Control statements
This value is always 0.

Assignment statements
This value is always 0.

Noncomment statements
This is the same as Total statements.

Parameters

The following positional parameters, separated by commas, are expected as input to FLMLPCBL

@@FLMLIS
The address of the dependencies pointer. This parameter is required.

@@FLMSTP
The address of the statistics output buffer. This parameter is required.

@@FLMSIZ
The size of the dependency list buffer in bytes. This parameter is required.

SQL=
Maximum of eight characters specifying the name of the include set assigned to EXEC SQL INCLUDE dependencies.

INCLSET
When INCLSET is present, include set dependencies will be generated. That is, when COPY ABC OF XYZ statements are encountered, a dependency for copybook ABC is generated with an include set name of XYZ. This facilitates having copybooks with same name from different sources.

When DBCS is present, quote delimiters are ignored in any token that contains either shift in (X'0E') or shift out (X'0F').

Return codes

0

Explanation

Indicates successful completion.

User response

None.

Project manager response

None.

4

Explanation

The dependency list does not match the source code for one of the following reasons:

- Truncation to 8 characters
- No trailing quotation mark to match a leading quotation mark
- Token consists of only 1 quotation mark
- Token consists of only 2 quotation marks
- Token is split between 2 lines using a hyphen in column 7 for concatenation

The dependency is not added to the list. Processing continues.

User response

Change the syntax to fit the parser. The line causing the error is written to data set *userid*.SCLMERR.*member*.

Project manager response

None.

8

Explanation

The number of parsed dependencies exceeds the size of the $list_info array, which is specified by the BUFSIZE parameter on the FLMLANGL macro.

User response

Either reduce the number of parsed dependencies for the member or contact the project manager.

Project manager response

Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Reassemble and relink the project definition.

12

Explanation

FLMTRNSL parameters are incorrect or are not specified.

User response

Contact the project manager.

Project manager response

Verify that the FLMTRNSL parameters on the FLMTRNSL macro for the FLMLPCBL parser are valid.

16

Explanation

A GETMAIN error for I/O storage occurred Processing stops.

User response

Contact the project manager.

Project manager response

Contact your IBM service representative.

20

Explanation

A severe error occurred. The source to be parsed cannot be opened or the LRECL is less than 16. Processing stops.

User response

Contact the project manager.

Project manager response

Verify the LRECL of the source file is 16 or greater.

22

Explanation

An I/O error occurred in the DCB while reading input. Processing stops.

User response

Contact the project manager.

Project manager response

Contact your IBM service representative.

FLMLPFRT FORTRAN Parser

Purpose

This is the FORTRAN parser translator that parses the source identified by the SOURCE DDNAME.

Using FLMLPFRT

The FORTRAN parser uses the following rules:

- Source must be fixed 80 and must be of the fixed form input format. Comment characters ('C' and '*') are recognized in column 1, continuation characters are recognized in column 6, and source statements are recognized in columns 7-72.

- Includes recognized are of the forms

```
INCLUDE (NAME)
EXEC SQL INCLUDE NAME
```

- INCLUDE statements can span lines if continuation characters are used.
- EXEC SQL INCLUDE statements are recognized and dependencies are generated (SQLCA and SQLDA are not flagged as external dependencies). SQL includes can span lines if continuation characters are used. All other EXEC statements are not flagged as a dependency.
- Comments and the contents of quoted strings are ignored.
- DBCS strings (delimited by shift-out and shift-in characters) in comments and quotes are allowed.

FLMLPFRT collects the following statistics about the source to be parsed:

Total lines
The total number of records in the file.

Comment lines
The number of lines with a (C) or an asterisk (*) in column 1 plus continued comments. A continued comment is a line that has a nonblank continuation character in column 6 and that follows a comment line or a continued comment.

Noncomment lines
The number of lines that are not comment lines, continued comment lines, or blank lines.

Blank lines
The number of lines that contain only blanks.

Prolog lines
The number of comment lines that are found before the first noncomment line.

Total statements
Comment statements plus noncomment statements.

Comment statements
The number of comment lines minus the number of lines that are continued comments.

Control statements
This value is always 0.

Assignment statements
This value is always 0.

Noncomment statements
The number of noncomment lines minus the number of lines that are continued noncomments. A continued noncomment is a line that has a nonblank continuation character in column 6 and that follows a noncomment line or a continued noncomment.

Parameters

The following keyword parameters are expected as input for FLMLPFRT:

LISTINFO
Pointer to the SCLM list information record. This parameter is required and must be set to @@FLMLIS.

LISTSIZE
The size of the LISTINFO buffer. This parameter is required and must be set to @@FLMSIZ.

PARSEDSN
Data set name containing the member to be parsed. The SCLM variable @@FLMDSN is the recommended value. This parameter is required.

PARSEMEM
The name of the member to be parsed. The SCLM variable @@FLMMBR is the recommended value. This parameter is required.

SOURCEDD
The ddname of the source to be read. This parameter is optional. If a SOURCEDD is specified, it will override the PARSEDSN and PARSEMEM parameters.

STATINFO
Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

Return codes

FLMLPFRT uses ISPF services. When a failure is the result of an ISPF service error, the message returned by the ISPF service is logged in the user's ISPF log (if there is one).

0

Explanation

Indicates successful completion.

User response

None.

Project manager response

None.

1

Explanation

Data set name not found in parameter list.

User response

Contact the project manager.

Project manager response

Check the language definition syntax (PARSEDSN parameter on the OPTIONS parameter on the FLMTRNSL FORTRAN parser). Verify that PORDER=1 or PORDER=3 was used on the FLMTRNSL macro of the language definition. A PORDER of 0 or 2 in the FLMTRNSL macro for the FLMLPFRT parser will result in FLMLPFRT receiving control without the OPTION list. PORDER 0 and 2 are used for situations in which there are no OPTION lists.

2

Explanation

The statistical information address (@@FLMSTP) is not found in the parameter list.

User response

Contact the project manager.

Project manager response

Check the language definition syntax (STATINFO parameter on the OPTIONS parameter on the FLMTRNSL FORTRAN parser).

3

Explanation

The list information address (@@FLMLIS) is not found in the parameter list.

User response

Contact the project manager.

Project manager response

Check the language definition syntax (LISTINFO parameter on the OPTIONS parameter on the FLMTRNSL FORTRAN parser).

4

Explanation

A dependency name was encountered that had more than 8 characters. The name is ignored and processing continues.

User response

Check the source member for dependency names longer than 8 characters. Dependency names are restricted to a length of 1 to 8 characters.

Project manager response

The language definition can be changed for GOODRC=4 if it is acceptable to ignore the dependency names that are longer than 8 characters.

5

Explanation

The STATINFO, LISTINFO, and/or LISTSIZE keyword parameters are invalid.

User response

Check the language definition for the correct values for keyword parameters STATINFO, LISTINFO, and LISTSIZE.

Project manager response

The invalid keyword parameters for FLMLPFRT OPTIONS in the language definition should be corrected and the project definition assembled and linked.

8

Explanation

The number of parsed dependencies exceeds the size of the $list_info array, which is specified by the BUFSIZE parameter on the FLMLANGL macro.

User response

Either reduce the number of parsed dependencies for the member or contact the project manager.

Project manager response

Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

10

Explanation

The member name specified by the parse parameter is blank.

User response

None.

Project manager response

Check the language definition syntax and member specification.

12

Explanation

The LISTSIZE keyword parameter in the OPTIONS is too small. There is not enough room for one element in the dependency array.

User response

Contact the project manager.

Project manager response

Update the project definition, assemble, and link-edit. LISTSIZE must be set to @@FLMSIZ in order to get a proper value for BUFSIZE.

24

Explanation

Parser was not linked AMODE(24), RMODE(24).

User response

Contact the project manager.

Project manager response

Reinstall the parser by relinking it AMODE(24). See ISPF log for more details.

100

Explanation

The value for the PARSEDSN keyword is invalid.

User response

Check the language definition and verify that PARSEDSN keyword value is valid.

Project manager response

None.

101 - 199

Explanation

The data set specified by the PARSEDSN keyword could not be allocated.

User response

Verify that the data set designated by the keyword exists.

Project manager response

None.

201 - 299

Explanation

The data set specified by the PARSEDSN keyword could not be opened, or is already opened.

User response

Verify that the data set exists, is not in use, and has not been allocated with a disposition of SHR or MOD.

Project manager response

None.

401 - 499

Explanation

An error occurred reading the data set specified by the PARSEDSN keyword. The data set is either empty, not opened for input, or has exceeded its space capacity.

User response

Verify that the data set exists, it is not empty, and the space allocation will support the process.

Project manager response

None.

500

Explanation

An error occurred when closing the file or when freeing storage.

User response

Contact the project manager.

Project manager response

Contact your IBM Service Representative.

599

Explanation

An ABEND was detected during I/O or allocation of the data set to be parsed.

User response

Check if the data set member to be parsed exists. An improper value for STATINFO in the OPTIONS of the FLMTRNSL for the parser can be another cause.

Project manager response

Make sure the data set member exists. Make corrections to the project definition; assemble and link the project definition.

FLMLPGEN General Purpose Parser

Purpose

FLMLPGEN is a general purpose parser that can get dependency information and statistics for the following languages:

```
370 Assembler
PL/I
REXX
CLIST
TEXT
```

General information:

- Comments and the contents of quoted strings are ignored.
- DBCS strings (delimited by shift-out and shift-in characters) in comments and quotes are allowed.
- Total lines and blank lines are always counted.
- Control statements and assignment statements are always set to zero.

Using FLMLPGEN as an Assembler parser

The Assembler parser uses the following rules:

- Set LANG=A for Assembler in the option list of the OPTIONS parameter, in the FLMTRNSL macro, and in the language definition macro (FLMLANGL).

- COPY statements with a continuation character in column 72 will be ignored.
- Any opcode not recognized as a standard 370 opcode is considered to be an external dependency (see next item).
- Macros that are defined inline are not flagged as external dependencies.
- Vector, ESA, and z/Series opcodes are recognized.
- OPSYN, ISEQ, ICTL, and others that alter the language or defaults are ignored.
- EXEC SQL INCLUDE statements are recognized and dependencies are generated (SQLCA and SQLDA are not flagged as external dependencies). SQL includes can span lines. All other EXEC statements are not flagged as a dependency.

Using FLMLPGEN as a PL/I parser

The PL/I parser uses the following rules:

- In the language definition, set LANG=I or LANG=1 for PL/I.
- A modifier (I) can be specified after the LANG keyword. If LANG=I(I) or LANG=1(I) is specified, include set dependencies will be generated. That is, when DD(A) is encountered, copybook A will be generated with an include set name of DD. This facilitates having copybooks with same name from different sources.
- Statements are just the number of semicolons not in comments or quotes plus commas not in parentheses. The following example has six statements (note the first DCL statement counts as three statements, but the second only counts as one because the commas are in parentheses).

```
EXAMPLE:PROC;
    DCL ONE   FIXED(31),
        TWO   FIXED(31),
        THREE FIXED(31);

    DCL (A_ONE, AN_A_TWO, AN_A_THREE) FIXED(31);

END EXAMPLE;
```

- Include statements cannot span lines.
- Include statements can include a ddname (as per PL/I syntax).
- Only the first %INCLUDE on a line will be recognized. Multiple dependencies are allowed on one line

```
%INCLUDE A, B, DD1(C), DD2(D) ...
```

- Preprocessor labels on include statements cause those includes to be missed.
- EXEC SQL INCLUDE statements are recognized and dependencies are generated (SQLCA and SQLDA are not flagged as external dependencies). SQL includes can span lines.
- Multiple EXEC SQL INCLUDE statements can appear on one line and the dependencies will be generated.
- Dependencies are recognized from all ddnames.

Using FLMLPGEN as a CLIST, REXX or Generic parser

FLMLPGEN uses the following rules for the CLIST, REXX, and generic parsers.

- In the language definition, set LANG=C for CLIST, LANG=R for REXX, or LANG=T for Generic Parser.
- Source can be any format (fixed or variable) up to record length 255.
- Sequence numbers are ignored.
- Continuation of statements is recognized by the following characters:

+ and - for CLIST
, for REXX

- Open comments (/* only) for CLIST are allowed. They are considered closed at the end of the line if there is no continuation character.
- The REXX language can be used to gather statistics for other languages that use /* and */ as delimiters such as ISPF panels. (Statistics might not be correct if any commas are at the end of any lines.)

Using FLMLPGEN as a TEXT parser

FLMLPGEN uses the following rules for parsing TEXT members.

- In the language definition, set LANG=T for TEXT.
- Source can be any format (fixed or variable) and any valid record length.
- Sequence numbering is counted as nonblank lines.
- Only total lines and blank lines are counted.
- Control statements and assignment statements are always set to zero.

Parameters

The following keyword parameters are expected as input for FLMLPGEN:

LANG=A|T|R|C|I|1
Use the LANG= parameter to specify the language to use to parse the members. If you do not include the LANG= parameter, the members are parsed as 370 Assembler. Valid language values are:

LANG=A
Assembler only

LANG=T
TEXT... count lines only

LANG=R
REXX or similar languages that use /* and */ as comment delimiters

LANG=C
CLIST

LANG=I
PL/I. Append (I) to generate include set dependencies.

LANG=1
PL/I. Append (I) to generate include set dependencies.

Note: When LANG=I(I) or LANG=1(I), the parser generates an include set dependency for statements of the form INCLUDE DD(A). Copybook A will have an include set name of DD assigned.

LISTINFO
Pointer to the SCLM list information record. This parameter is required and must be set to @@FLMLIS.

LISTSIZE
The size of the LISTINFO buffer. This parameter is required and must be set to @@FLMSIZ.

SOURCEDD
The ddname of the source to be read. This parameter is required.

STATINFO
Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

SQL=
Maximum of eight characters specifying the name of the include set assigned to EXEC SQL INCLUDE dependencies.

Return codes

FLMLPGEN uses ISPF services. When a failure is the result of an ISPF service error, the message returned by the ISPF service is logged in the user's ISPF log (if there is one).

0

Explanation

Indicates successful completion.

User response

None.

Project manager response

None.

1

Explanation

Data set name not found in parameter list.

User response

Contact the project manager.

Project manager response

Check the language definition PORDER value and syntax.

2

Explanation

The statistical information address (@@FLMSTP) was not found in the parameter list.

User response

Contact the project manager.

Project manager response

Check the language definition syntax (STATINFO parameter on the OPTIONS parameter on the FLMTRNSL FLMLPGEN parser).

3

Explanation

The list information address (@@FLMLIS) was not found in the parameter list.

User response

Contact the project manager.

Project manager response

Check the language definition syntax (LISTINFO parameter on the OPTIONS parameter on the FLMTRNSL FLMLPGEN parser).

4

Explanation

Dependency name longer than 8 characters was recognized. The dependency is not added to the dependency list. Processing continues.

User response

Verify and correct the length of the dependency name.

Project manager response

None.

5

Explanation

Maximum list size (LISTSIZE) was not found in parameter list.

User response

None.

Project manager response

Check the language definition syntax (LISTSIZE parameter on the OPTIONS parameter on the FLMTRNSL FLMLPGEN parser).

6

Explanation

SOURCEDD parameter is greater than eight characters.

User response

Contact the project manager.

Project manager response

Check the language definition syntax (SOURCEDD parameter on the OPTIONS parameter on the FLMTRNSL, FLMLPGEN parser).

7

Explanation

The number of parsed dependencies exceeds the size of the $list_info array, that is specified by the BUFSIZE parameter on the FLMLANGL macro.

User response

Either reduce the number of parsed dependencies for the member or contact the project manager.

Project manager response

Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

9

Explanation

Bad value encountered for LISTINFO, LISTSIZE or STATINFO.

User response

Contact the project manager.

Project manager response

Check the language definition syntax (OPTIONS parameter on the FLMTRNSL, FLMLPGEN parser).

10

Explanation

Member name not found.

User response

None.

Project manager response

Check the language definition syntax.

12

Explanation

Invalid language.

User response

Contact the project manager.

Project manager response

Check the language definition syntax (LANG parameter on the OPTIONS parameter on the FLMTRNSL, FLMLPGEN parser).

22

Explanation

I/O error.

User response

Check the source code for a dependency name greater than 8 characters. The I/O error may occur when the TSO prefix is not set and the parser attempts to allocate an error data set. If the TSO prefix was not set then set the TSO prefix and run the parse again if you cannot locate the dependency name that is greater than 8 characters.

Project manager response

Verify the data sets used by the parser OPEN and CLOSE properly.

101 - 199

Explanation

The data set specified by the PARSEDSN keyword could not be allocated.

User response

Contact the project manager.

Project manager response

Verify that the data set designated by the keyword exists.

201 - 299

Explanation

The data set specified by the PARSEDSN keyword could not be opened, or is already opened.

User response

Verify that the data set exists, is not in use, and has not been allocated with a disposition of SHR or MOD.

Project manager response

None.

401 - 499

Explanation

An error occurred reading the data set specified by the PARSEDSN keyword. The data set is either empty, not opened for input, or has exceeded its space capacity.

User response

Verify that the data set exists, it is not empty, and the space allocation will support the process.

Project manager response

None.

500

Explanation

An error occurred when closing the file or when freeing storage.

User response

Contact the project manager.

Project manager response

Contact your IBM service representative.

599

Explanation

An ABEND was detected during I/O or allocation of the data set to be parsed.

User response

Check to see if the data set member to be parsed exists. An improper value for STATINFO in the OPTIONS of the FLMTRNSL for the parser can be another cause.

Project manager response

Make sure the data set member exists. Make any necessary corrections to the project definition; assemble and link the project definition.

FLMLRASM REXX Assembler Parser

Purpose

This is the assembler parser translator, written in REXX, that parses the source identified by the SOURCE DDNAME.

Functions

One of the functions of an SCLM parser is to determine all of a module's dependencies. FLMLRASM determines all of the names that are to be copied into the Assembler source.

The parser uses the following syntax rules to locate dependency names:

- The search for tokens is restricted to columns 2-71. Column 72 is checked for a non-null element (treated as a continuation). The use of non-null elements to continue strings for forming reserved words or dependency names is not supported.
- An opcode or dependency token that extends into the continuation column will not be added as a dependency; the parser return code will be set to 4, the line in error will be written to the error listing data set (*userid*.SCLMERR.LISTING), and processing will continue.
- When a line that is not a comment line or a continuation line has COPY after column 1, the next token is taken as the name of a dependency.

  Note: If the member name is enclosed in single or double quotes, the quotes are ignored.
- When searching for the next token on a line and there are no more tokens on that line, the search continues with the next continued line, if there is one. Comment statements must not appear between an instruction statement and its continuation lines.
- Tokens inside quoted strings will be ignored (except that quotation marks around a member following a COPY or EXEC SQL INCLUDE statement are removed).
- Labels starting in column 1 to the end of the token are considered white space.

FLMLRASM will generate a dependency for the MEMBER# token under the following conditions:

- MEMBER# is the first token of a statement and is not one of the opcodes for the z/Series processors (including assembler extended mnemonics, Vector facility and some obsolete 360/370 instructions).
- MEMBER# is the first token after a COPY or EXEC SQL INCLUDE instruction. It can be on a continued line.

The following example illustrates conditions under which dependencies will and will not be formed. Each MEMBER# token appears in an example of syntax that the parser recognizes as creating a dependency. A MEMBER# token can be from 1 to 8 characters. The BADCPY# statements in the example will not create dependencies for the following reasons:

- BADCPY1 follows an EXEC CICS instruction; dependencies are only generated for precompiler instructions (EXEC SQL INCLUDE).
- BADCPY2 first appears in a macro definition, so no dependency is created on subsequent appearances.
- BADCPY3 begins with an ampersand.
- BADCPY4 is not the first token of the statement in which it appears.

```
*<-Column 1                                                  Column 72->
 MEMBER1 rest of line
LABEL MEMBER2 rest of line
 COPY MEMBER3 rest of line
 COPY                                                                 X
              MEMBER4
* A COMMENT LINE ******************************************************
* DB2 PREPROCESSOR STATEMENTS  - each is 1 statement, 1 dependency
 EXEC SQL INCLUDE MEMBER5
 EXEC SQL INCLUDE                                                     X
              MEMBER6
* CICS PREPROCESSOR STATEMENT  - 1 statement, no dependency
 EXEC CICS BADCPY1
* Statements for which no dependency is generated
 MACRO                                                                X
              BADCPY2
 &BADCPY3   rest of line
* previously defined macros ignored
 BADCPY2                                                              X
              BADCPY4
* continued lines ignored, except after COPY & EXEC SQL INCLUDE
```

Another function of the parser is to gather statistics or metrics for each module to be parsed. There are ten such statistics saved by SCLM, but only 8 are generated by this parser. For assembler, this parser defines the ten statistics as follows:

Total lines
The total number of records in the file.

Comment lines
The number of lines with an asterisk in the first column.

Noncomment lines
The number of total lines minus the number of comment lines.

Blank lines
The number of lines that contain only blanks.

Prolog lines
The number of comment lines and blank lines that are found before the first noncomment line.

Total statements
The number of comment statements plus the number of noncomment statements.

Comment statements
This value is equal to the number of comment lines.

Control statements
This value is always 0.

Assignment statements
This value is always 0.

Noncomment statements
The number of statements whose first token is a reserved word, plus the number of EXEC SQL and EXEC CICS instructions.

Parameters

The following guidelines apply when specifying parameters:

- The order of the parameters is not important.
- See the language definitions provided by SCLM for the actual usage of the parameters for FLMLRASM.

The following keyword parameters, separated by commas, are required as input to FLMLRASM:

LISTINFO
Pointer to the SCLM list information record. This parameter is required and must be set to @@FLMLIS.

LISTSIZE
The size of the LISTINFO buffer. This parameter is required and must be set to @@FLMSIZ. The parser checks to make sure that the LISTSIZE parameter is large enough to hold at least one entry of 228 bytes.

STATINFO
Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

Return codes

0

Explanation

Indicates successful completion.

User response

None.

Project manager response

None.

4

Explanation

The dependency name does not match the source code for one of the following reasons:

- The dependency is greater than 8 characters. The error message in *userid*.SCLMERR.LISTING is:

  ```
  4
  line
  dependency
  ```

  where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded 8 characters.
- The dependency name flows into column 72.

  The error message in *userid*.SCLMERR.LISTING is:

  ```
  4
  line
  dependency
  ```

  where *line* is the source line that contains the dependency, and *dependency* is the dependency that extends into column 72. *dependency* will be positioned under its occurrence in *line* to show that it is too far over in the source file.
- The dependency name after a COPY is prefixed by an ampersand (&).

  The error message in *userid*.SCLMERR.LISTING is:

  ```
  4
  line
  &
  ```

  where *line* is the source line that contains the dependency that begins with an ampersand, and & is printed under its occurrence in *line*.
- Mismatched quotes - a single or double quote was found that did not have a matching closing quote.

  The error message in *userid*.SCLMERR.LISTING is:

  ```
  4
  mark line_no
  ```

  where *mark* is either a single or double quotation mark, and *line_no* is the line number that contains the unmatched quotation mark.

The dependency is not added to the list. Processing continues.

User response

Change the syntax to fit the parser.

Project manager response

None.

8

Explanation

The number of parsed dependencies exceeds the size of the $list_info array, which is specified by the BUFSIZE parameter on the FLMLANGL macro.

The error message in *userid*.SCLMERR.LISTING is:

```
8
line
dependency
```

where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded the space allocated for the list.

User response

Either reduce the number of parsed dependencies for the member or contact the project manager.

Project manager response

Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

10

Explanation

FLMTRNSL parameters are incorrect or are not specified.

The error message in *userid*.SCLMERR.LISTING is:

```
10
bad_parms
```

where *bad_parms* are the parameters that are incorrect or not specified.

User response

Contact the project manager.

Project manager response

Verify that the FLMTRNSL parameters for the FLMLRASM parser are valid and complete.

12

Explanation

Issued by TSOLNK; the parser was not found in the data set specified on the DSNAME parameter of the FLMTRNSL macro.

User response

Contact the project manager.

Project manager response

Verify that the value of the DSNAME parameter on the FLMTRNSL macro is correct.

16

Explanation

Error opening the error listings file.

User response

Contact the project manager.

Project manager response

Ensure that user has the authority to create and write to the file *userid*.SCLMERR.LISTING.

20

Explanation

Error closing the source file.

User response

Contact the project manager.

Project manager response

Contact your IBM service representative.

22

Explanation

An I/O error occurred in the DCB while reading input. Processing stops.

User response

Contact the project manager.

Project manager response

Contact your IBM service representative. Verify that the SOURCE DDNAME is allocated correctly. Verify that the data set and member to parse exist. Verify that the FLMALLOC macro is complete and valid for the parser.

40

Explanation

SCLM was not successful in invoking FLMLRASM using IKJEFTSR (TSOLNK).

User response

Contact the project manager.

Project manager response

FLMLRASM does not return a 40. A 40 can be encountered from SCLM for CALLMETH=TSOLNK. TSOLNK is used for executing interpretive REXX. A 40 means IKJEFTSR (TSOLNK) was not successful.

FLMLRCBL REXX COBOL Parser

Purpose

This is the COBOL parser translator, written in REXX, that parses the source identified by the SOURCE DDNAME.

Functions

One of the functions of an SCLM parser is to determine a module's dependencies. FLMLRCBL determines the names of dependencies that are to be copied into the COBOL source.

The parser uses the following syntax rules to locate dependency names:

- The search for tokens is restricted to columns 8 -72. Column 7 is ignored except when it contains '*' or '/' (treated as a comment line) or '-' (treated as a concatenation). The use of '-' to concatenate strings for forming reserved words or dependency names is not supported.
- When a line that is not a comment line or a continuation line has COPY after column 7, the next token is taken as the name of a dependency. Note the following exceptions.

  If the member name is enclosed in single or double quotes, the quotes are ignored and the member name is taken as the name of a dependency.
- When the 3 tokens EXEC, SQL, and INCLUDE are found in order on 1 or more uncommented lines after column 7, with no intervening text, the next token is taken as the name of a dependency. Note the following exceptions.

  If the member name is enclosed in single or double quotes, the quotes are ignored and the member name is taken as the name of a dependency.
- When searching for the next token on a line and there are no more tokens on that line, the search continues with the next uncommented line.
- Tokens inside quoted strings will be ignored, except for quoted member names following COPY statements. Reserved words inside quoted strings and comments will not be counted as statements.
- FLMLRCBL recognizes COPY or EXEC SQL INCLUDE anywhere in the source file (as long as they are not in quotation marks or comments).

Multiple COPY or EXEC SQL INCLUDE statements on any line or continued line are recognized.

The following example illustrates conditions under which dependencies will and will not be formed. Each MEMBER# token appears in an example of syntax that the parser recognizes as creating a dependency. A MEMBER# token must be from 1 to 8 characters. The BADCPY1 and BADCOPY02 statements in the example will not create dependencies for the following reasons:

- BADCPY1 and the COPY preceding it are inside a quoted string and are therefore ignored.
- BADCOPY02 is longer than 8 characters.

```
123456*<-Column 7                                        Column 72->
001010 FD TEST-FILE COPY MEMBER1.
001200 01  I-O-CNTL .     COPY   'MEMBER2'
001201 01  I-O-CNTL       COPY   "MEMBER3" .
```

```
001201 01  LABEL  PIC X VALUE 'EXTRA COPY
001201-                                BADCPY1 '.
001202 EXEC SQL INCLUDE MEMBER4
001202 EXEC SQL INCLUDE 'MEMBER5'
001202 EXEC SQL INCLUDE "MEMBER6"
001300     COPY
001300* copy across a comment line
001300            MEMBER7.
001400 01 TESTNAMX COPY MEMBER8 . COPY MEMBER9.
001401   77 TESTNAME . COPY BADCOPY02.
```

Another function of the parser is to gather statistics or metrics for each module to be parsed. SCLM saves 10 statistics; only 7 are generated by this parser. For COBOL, this parser defines the following 10 statistics:

Total lines
The total number of records in the file.

Comment lines
The number of lines with a slash (/) or an asterisk (*) in column 7.

Noncomment lines
The number of total lines minus the number of comment lines.

Blank lines
The number of lines that contain only blanks after column 6. Any sequence numbers in the rightmost columns of the line are ignored.

Prolog lines
The number of comment lines and blank lines that are found before the first noncomment line.

Total statements
The number of the following reserved words that appear on an uncommented line after column 7:

```
ACCEPT       DIVIDE       INSPECT     REWRITE
ADD          ENTER        MERGE        SEARCH
ALTER        ENTRY        MOVE         SET
CALL         EVALUATE     MULTIPLY     SORT
CANCEL       EXAMINE      NOTE         START
CLOSE        EXIT         ON           STOP
COMPUTE      GO           OPEN         STRING
CONTINUE     GOBACK       PERFORM      SUBTRACT
COPY         GOTO         READ         TRANSFORM
DELETE       IF           RELEASE      UNSTRING
DISPLAY      INITIALIZE   RETURN       WRITE
```

In addition, any EXEC SQL and EXEC CICS statements are treated as program statements.

Comment statements
This value is always 0.

Control statements
This value is always 0.

Assignment statements
This value is always 0.

Noncomment statements
This is the same as total statements.

Parameters

Use the following guidelines to specify parameters:

- The order of the parameters is not important.
- See the language definitions provided by SCLM for the actual usage of the parameters for FLMLRCBL.

The following keyword parameters, separated by commas, are required as input for FLMLRCBL:

LISTINFO
Pointer to the SCLM list information record. This parameter is required and must be set to @@FLMLIS.

LISTSIZE
The size of the LISTINFO buffer. This parameter is required and must be set to @@FLMSIZ. The parser checks to make sure that the LISTSIZE parameter is large enough to hold at least one entry of 228 bytes.

STATINFO
Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

Return codes

0

Explanation

Indicates successful completion.

User response

None.

Project manager response

None.

4

Explanation

The dependency name does not match the source code for one of the following reasons:

- Dependency name greater than 8 characters. The error message in *userid*.SCLMERR.LISTING is:

  ```
  4
  line
  dependency
  ```

  where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded 8 characters. The dependency is not added to the dependency list.
- Mismatched quotes. A single or double quote did not have a matching closing quote.

  The error message in *userid*.SCLMERR.LISTING is:

  ```
  4
  mark line_no
  ```

  where *mark* is either a single or double quotation mark that has no matching closing quote, and *line_no* is the line number of the line that contains the unmatched quotation mark.

The dependency is not added to the list. Processing continues.

User response

Change the syntax to fit the parser.

Project manager response

None.

8

Explanation

The number of parsed dependencies exceeds the size of the $list_info array, which is specified by the BUFSIZE parameter on the FLMLANGL macro.

The error message in *userid*.SCLMERR.LISTING is:

```
8
dependency
```

where *dependency* is the dependency that exceeded the space allocated for the list.

User response

Either reduce the number of parsed dependencies for the member or contact the project manager.

Project manager response

Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

10

Explanation

FLMTRNSL parameters are incorrect or are not specified.

The error message in *userid*.SCLMERR.LISTING is:

```
10
bad_parms
```

where *bad_parms* are the parameters that are incorrect or not specified.

User response

Contact the project manager.

Project manager response

Verify that the FLMTRNSL parameters for the FLMLRCBL parser are valid and complete.

12

Explanation

Issued by TSOLNK; the parser was not found in the data set specified on the DSNAME parameter of the FLMTRNSL macro.

User response

Contact the project manager.

Project manager response

Verify that the value of the DSNAME parameter on the FLMTRNSL macro is correct.

16

Explanation

Error opening the error listings file.

User response

Contact the project manager.

Project manager response

Ensure that user has the authority to create and write to the file *userid*.SCLMERR.LISTING.

20

Explanation

Error closing the source file.

User response

Contact the project manager.

Project manager response

Contact your IBM service representative.

22

Explanation

An I/O error occurred in the DCB while reading input. Processing stops.

User response

Contact the project manager.

Project manager response

Verify that the SOURCE DDNAME is allocated correctly. Verify that the data set and member to parse exist. Verify that the FLMALLOC macro is complete and valid for the parser. Contact your IBM service representative.

40

Explanation

SCLM was not successful in invoking FLMLRCBL using IKJEFTSR (TSOLNK).

User response

Contact the project manager.

Project manager response

FLMLRCBL does not return a 40. A 40 can be encountered from SCLM for CALLMETH=TSOLNK. TSOLNK is used for executing interpretive REXX. A 40 means IKJEFTSR (TSOLNK) was not successful.

FLMLRCIS MVS C/C++ parser with include set support

Purpose

The FLMLRCIS parser supports MVS C and C++ source files. The parser is written in REXX. The includes found by the parser are associated with an include set that is the set name from the include statement. For information about include sets, see “FLMINCLS macro” on page 542.

Functions

The parser uses the following syntax rules to locate dependency names:

- The search for tokens is restricted to uncommented text.

The character strings /* and */ are recognized as comment delimiters that can span lines. The character string // is recognized as a begin comment token where the comment ends at the end of the line.

- Include dependencies are generated when the first token on the line is #include.

  The dependency consists of the member or include name and the include set name in the format 'member.set', where set is the include set name. It can be surrounded by double quotes ("member.set") or by angle brackets (<member.set>).

- Tokens inside strings are ignored.

The following table illustrates how include and include-set names are derived from source statements.

Table 34. Examples of include and nclude-set names derived from source statements

Source statement	Include name	Include-set name
#include "abc"	abc	
#include "abc.h"	abc	h

Another function of the parser is to gather statistics or metrics for each module to be parsed. SCLM saves 10 statistics, but only 4 are generated by this parser. This parser defines the ten statistics as follows:

Total lines
The total number of records in the file.

Comment lines
This value is always 0.

Noncomment lines
This is the same as the total lines.

Blank lines
The number of lines that contain only blanks.

Prolog lines
This value is always 0.

Total statements
This value is always 0.

Comment statements
The total number of `/* */` pairs in the member.

Control statements
This value is always 0.

Assignment statements
This value is always 0.

Noncomment statements
This value is always 0.

Parameters

The following guidelines apply when specifying parameters:

- The order of the parameters is not important.
- See the language definition provided by SCLM for the actual use of the parameters for FLMLRCIS.

The following keyword parameters, separated by commas, are required as input to FLMLRCIS:

LISTINFO
Pointer to the SCLM list information record. This parameter is required and must be set to @@FLMLIS.

LISTSIZE
The size of the LISTINFO buffer. This parameter is required and must be set to @@FLMSIZ. The parser checks to make sure that the LISTSIZE parameter is large enough to hold at least one entry of 228 bytes.

STATINFO
Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

Return codes

0

Explanation

Indicates successful completion.

User response

None.

Project manager response

None.

4

Explanation

The dependency name does not match the source code for one of the following reasons:

- The include name is greater than 8 characters. The error message in *userid*.SCLMERR.LISTING is:

```
4
line
dependency
```

  where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded eight characters. The truncated form of the dependency is added to the dependency list.
- Unsupported form of include dependency. The error message in *userid*.SCLMERR.LISTING is:

```
4
line
dependency
```

  where *line* is the source line that contains the dependency, and *dependency* is the text of the desired include member.
- #include statement spans multiple lines. The error message in *userid*.SCLMERR.LISTING is:

```
4
line
mark
```

  where *line* is the source line that contains the mark, and *mark* is the mark (either a quotation mark or an angle bracket) that has no matching pair on the line.
- Mismatched quotes. A single or double quote was found that did not have a matching closing quote. The error message in *userid*.SCLMERR.LISTING is:

```
4
mark  line_no
```

  where *line_no* is the line number that contains the unmatched quotation mark, and *mark* is either a single or double quotation mark.

Processing continues.

User response

Change the syntax to fit the parser.

Project manager response

None.

8

Explanation

The number of parsed dependencies exceeds the size of the $list_info array, which is specified by the BUFSIZE parameter on the FLMLANGL macro. The error message in *userid*.SCLMERR.LISTING is:

```
8
line
dependency
```

where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded the space allocated for the list.

User response

Either reduce the number of parsed dependencies for the member or contact the project manager.

Project manager response

Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

10

Explanation

FLMTRNSL parameters are incorrect or are not specified.

The error message in *userid*.SCLMERR.LISTING is:

```
10
bad_parms
```

where *bad_parms* are the parameters that are incorrect or not specified.

User response

Contact the project manager.

Project manager response

Verify that the FLMTRNSL parameters for the FLMLRC2 parser are valid and complete.

12

Explanation

Issued by TSOLNK; the parser was not found in the data set specified on the DSNAME parameter of the FLMTRNSL macro.

User response

Contact the project manager.

Project manager response

Verify that the value of the DSNAME parameter on the FLMTRNSL macro is correct.

16

Explanation

Error opening the error listings file.

User response

Contact the project manager.

Project manager response

Ensure that user has the authority to create and write to the file *userid*.SCLMERR.LISTING.

20

Explanation

Error closing the source file.

User response

Contact the project manager.

Project manager response

Contact your IBM service representative.

22

Explanation

An I/O error occurred in the DCB while reading input. Processing stops.

User response

Contact the project manager.

Project manager response

Contact your IBM service representative. Verify that the SOURCE DDNAME is allocated correctly. Verify that the data set and member to parse exist. Verify that the FLMALLOC macro is complete and valid for the parser.

40

Explanation

SCLM was not successful in invoking FLMLRCIS using IKJEFTSR (TSOLNK).

User response

Contact the project manager.

Project manager response

FLMLRC2 does not return a 40. A 40 can be encountered from SCLM for CALLMETH=TSOLNK. TSOLNK is used for executing interpretive REXX. A 40 means IKJEFTSR (TSOLNK) was not successful.

FLMLRC2 C, C++, and Resource file parser for workstation source

Purpose

The FLMLRC2 parser supports C, C++ and resource files. The parser is written in REXX. The includes found by the parser are associated with an include set that is the extension from the include statement (see “FLMINCLS macro” on page 542).

Functions

The parser uses the following syntax rules to locate dependency names:

- The search for tokens is restricted to uncommented text.

  The character strings /* and */ are recognized as comment delimiters that can span lines. The character string // is recognized as a begin comment token where the comment ends at the end of the line.
- Include dependencies are generated in the following conditions:
  - The first token on the line is #include. The included file name can be surrounded by double quotes ("file.ext") or by angle brackets (<file.ext>).
  - Dependencies are generated for some resource compiler statements. The statements support options between the statement and the include name, so the include name is taken as the last token on the line. Some of these statements have a format for includes and a format that does not support includes. The parser only finds includes when the statement does not contain a comma. The following statements are recognized as include statements:

    ```
    BITMAP
    FONT
    ICON
    POINTER
    RESOURCE
    RCINCLUDE
    DLGINCLUDE
    ```
- Tokens inside strings are ignored.

Include names are generated after removing excess characters (all characters up to and including the rightmost directory separator character, if any, and all characters from the first '.' to the end of the file name). The default is \. Any underscore characters (_) or blanks are replaced by at-signs ('@'). Include names longer than eight characters are truncated to eight characters and a return code of 4 is issued. The include-set names are generated from the characters following the first '.' to the end of the file name. Include-set names are also truncated to eight characters and underscore characters and blanks are replaced by at-signs. The following table illustrates how include and include-set names are derived from source statements.

Table 35. Examples of include and include-set names derived from source statements

Source statement	Include name	Include-set name
#include "abc"	abc	
#include "abc.h"	abc	h
ICON 97, 101, 10, 10, 0, 0		
ICON ID_WINDOW mahjongg.ico	mahjongg	ico
#include "my file.h"	my@file	h

Another function of the parser is to gather statistics or metrics for each module to be parsed. SCLM saves 10 statistics, but only 4 are generated by this parser. This parser defines the ten statistics as follows:

Total lines
The total number of records in the file.

Comment lines
This value is always 0.

Noncomment lines
This is the same as the total lines.

Blank lines
The number of lines that contain only blanks.

Prolog lines
This value is always 0.

Total statements
This value is always 0.

Comment statements
The total number of /* */ pairs in the member.

Control statements
This value is always 0.

Assignment statements
This value is always 0.

Noncomment statements
This value is always 0.

Parameters

The following guidelines apply when specifying parameters:

- The order of the parameters is not important.
- See the language definitions provided by SCLM for the actual use of the parameters for FLMLRC2.
- The directory separator character defaults to \.

The DIR_SEPARATOR keyword parameter may be used to specify a directory separator character.

The following keyword parameters, separated by commas, are required as input to FLMLRC2

LISTINFO
Pointer to the SCLM list information record. This parameter is required and must be set to @@FLMLIS.

LISTSIZE
The size of the LISTINFO buffer. This parameter is required and must be set to @@FLMSIZ. The parser checks to make sure that the LISTSIZE parameter is large enough to hold at least one entry of 228 bytes.

STATINFO
Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

Return codes

0

Explanation

Indicates successful completion.

User response

None.

Project manager response

None.

4

Explanation

The dependency name does not match the source code for one of the following reasons:

- Dependency name greater than 8 characters. The error message in *userid*.SCLMERR.LISTING is:

  ```
  4
  line
  dependency
  ```

  where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded eight characters. The truncated form of the dependency is added to the dependency list.
- Unsupported form of include dependency. The error message in *userid*.SCLMERR.LISTING is:

  ```
  4
  line
  dependency
  ```

  where *line* is the source line that contains the dependency, and *dependency* is the text of the desired include member.
- #include statement spans multiple lines. The error message in *userid*.SCLMERR.LISTING is:

  ```
  4
  line
  mark
  ```

  where *line* is the source line that contains the mark, and *mark* is the mark (either a quotation mark or an angle bracket) that has no matching pair on the line.
- Mismatched quotes. A single or double quote was found that did not have a matching closing quote. The error message in *userid*.SCLMERR.LISTING is:

  ```
  4
  mark  line_no
  ```

  where *line_no* is the line number that contains the unmatched quotation mark, and *mark* is either a single or double quotation mark.

Processing continues.

User response

Change the syntax to fit the parser.

Project manager response

None.

8

Explanation

The number of parsed dependencies exceeds the size of the $list_info array, which is specified by the BUFSIZE parameter on the FLMLANGL macro. The error message in *userid*.SCLMERR.LISTING is:

```
8
line
dependency
```

where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded the space allocated for the list.

User response

Either reduce the number of parsed dependencies for the member or contact the project manager.

Project manager response

Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

10

Explanation

FLMTRNSL parameters are incorrect or are not specified.

The error message in *userid*.SCLMERR.LISTING is:

```
10
bad_parms
```

where *bad_parms* are the parameters that are incorrect or not specified.

User response

Contact the project manager.

Project manager response

Verify that the FLMTRNSL parameters for the FLMLRC2 parser are valid and complete.

12

Explanation

Issued by TSOLNK; the parser was not found in the data set specified on the DSNAME parameter of the FLMTRNSL macro.

User response

Contact the project manager.

Project manager response

Verify that the value of the DSNAME parameter on the FLMTRNSL macro is correct.

16

Explanation

Error opening the error listings file.

User response

Contact the project manager.

Project manager response

Ensure that user has the authority to create and write to the file *userid*.SCLMERR.LISTING.

20

Explanation

Error closing the source file.

User response

Contact the project manager.

Project manager response

Contact your IBM service representative.

22

Explanation

An I/O error occurred in the DCB while reading input. Processing stops.

User response

Contact the project manager.

Project manager response

Contact your IBM service representative. Verify that the SOURCE DDNAME is allocated correctly. Verify that the data set and member to parse exist. Verify that the FLMALLOC macro is complete and valid for the parser.

40

Explanation

SCLM was not successful in invoking FLMLRC2 using IKJEFTSR (TSOLNK).

User response

Contact the project manager.

Project manager response

FLMLRC2 does not return a 40. A 40 can be encountered from SCLM for CALLMETH=TSOLNK. TSOLNK is used for executing interpretive REXX. A 40 means IKJEFTSR (TSOLNK) was not successful.

FLMLRC37 REXX C370 Parser

Purpose

This is the C/370 parser translator, written in REXX, that parses the source identified by the SOURCE DDNAME.

Functions

One of the functions of an SCLM parser is to determine all of a module's dependencies. FLMLRC37 determines all of the names that will be copied into the C/370 source.

The parser uses the following syntax rules to locate dependency names:

- The search for tokens is restricted to uncommented text.
- When an uncommented line has #INCLUDE as the first token, followed by a token enclosed in double quotes ("MEMBER") or angle brackets (<MEMBER>), the enclosed token is accepted as the name of a dependency. Note the following exceptions.
  - When an uncommented line has EXEC, SQL, and INCLUDE as its first three tokens, the next token is accepted as the name of a dependency.
  - Tokens inside strings or comments are ignored. `/* */` pairs are recognized as comment delimiters by the FLMLRC37 parser. Lines starting with // are also recognized as comments.

Dependencies are generated after removing excess characters (all characters up to and including the rightmost /, if any, and all characters from the first period (.) to the end of the file name). Any underscore characters (_) are replaced by at sign characters (@). Dependency names longer than 8 characters are

truncated to 8 characters and a return code of 4 is issued. The following table illustrates how dependencies are derived from include directives.

Table 36. Examples of dependancies derived from include directives

#include Directive	Dependency Generated	Return Code
#include "abc"	ABC	0
#include <sys/abc/xx>	XX	0
#include "Sys/abc/xx.h"	XX	0
#include <sys/name_1>	NAME@1	0
#include "Name2/App1.App2"	APP1	0
#include "xx.h.a"	XX	0
#include <DD:PLAN(YEAR)>	*NONE*	4
#include <'USER.SRC.MYINCS'>	*NONE*	4
#include "abc456789"	ABC45678	4

The following example further illustrates conditions under which dependencies will and will not be formed. Each MEMBER# token appears in an example of syntax that the parser recognizes as creating a dependency. The BADCPY# statements will not create dependencies for the following reasons:

- BADCPY1 is inside comment delimiters.
- BADCPY2 is not inside quotes or angle brackets.
- BADCPY3 is inside a string.

```
/* #include "badcpy1" */
#include "member1"
#include <member2>
#include badcpy2
EXEC SQL INCLUDE member3
printf '#include badcpy3'
```

Another function of the parser is to gather statistics or metrics for each module to be parsed. SCLM saves 10 statistics, but only 4 are generated by this parser. For C/370, this parser defines the ten statistics as follows:

Total lines
The total number of records in the file.

Comment lines
This value is always 0.

Noncomment lines
This is the same as the total lines.

Blank lines
The number of lines that contain only blanks.

Prolog lines
This value is always 0.

Total statements
This value is always 0.

Comment statements
The total number of `/* */` pairs in the member.

Control statements
This value is always 0.

Assignment statements
This value is always 0.

Noncomment statements
This value is always 0.

Parameters

The following guidelines apply when specifying parameters:

- The order of the parameters is not important.
- See the language definitions provided by SCLM for the actual use of the parameters for FLMLRC37.

The following keyword parameters, separated by commas, are required as input to FLMLRC37

LISTINFO
Pointer to the SCLM list information record. This parameter is required and must be set to @@FLMLIS.

LISTSIZE
The size of the LISTINFO buffer. This parameter is required and must be set to @@FLMSIZ. The parser checks to make sure that the LISTSIZE parameter is large enough to hold at least one entry of 228 bytes.

STATINFO
Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

Return codes

0

Explanation

Indicates successful completion.

User response

None.

Project manager response

None.

4

Explanation

The dependency name does not match the source code for one of the following reasons:

- Dependency name greater than 8 characters. The error message in *userid*.SCLMERR.LISTING is:

```
4
line
dependency
```

  where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded eight characters. The truncated form of the dependency is added to the dependency list.
- Unsupported form of include dependency. The error message in *userid*.SCLMERR.LISTING is:

```
4
line
dependency
```

  where *line* is the source line that contains the dependency, and *dependency* is the text of the desired include member.
- #include statement spans multiple lines. The error message in *userid*.SCLMERR.LISTING is:

```
4
line
mark
```

  where *line* is the source line that contains the mark, and *mark* is the mark (either a quotation mark or an angle bracket) that has no matching pair on the line.
- Mismatched quotes. A single or double quote was found that did not have a matching closing quote. The error message in *userid*.SCLMERR.LISTING is:

```
4
mark  line_no
```

  where *line_no* is the line number that contains the unmatched quotation mark, and *mark* is either a single or double quotation mark.

Processing continues.

User response

Change the syntax to fit the parser.

Project manager response

None.

8

Explanation

The number of parsed dependencies exceeds the size of the $list_info array, which is specified by the BUFSIZE parameter on the FLMLANGL macro. The error message in *userid*.SCLMERR.LISTING is:

```
8
line
dependency
```

where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded the space allocated for the list.

User response

Either reduce the number of parsed dependencies for the member or contact the project manager.

Project manager response

Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

10

Explanation

FLMTRNSL parameters are incorrect or are not specified.

The error message in *userid*.SCLMERR.LISTING is:

```
10
bad_parms
```

where *bad_parms* are the parameters that are incorrect or not specified.

User response

Contact the project manager.

Project manager response

Verify that the FLMTRNSL parameters for the FLMLRC37 parser are valid and complete.

12

Explanation

Issued by TSOLNK; the parser was not found in the data set specified on the DSNAME parameter of the FLMTRNSL macro.

User response

Contact the project manager.

Project manager response

Verify that the value of the DSNAME parameter on the FLMTRNSL macro is correct.

16

Explanation

Error opening the error listings file.

User response

Contact the project manager.

Project manager response

Ensure that user has the authority to create and write to the file *userid*.SCLMERR.LISTING.

20

Explanation

Error closing the source file.

User response

Contact the project manager.

Project manager response

Contact your IBM service representative.

22

Explanation

An I/O error occurred in the DCB while reading input. Processing stops.

User response

Contact the project manager.

Project manager response

Contact your IBM service representative. Verify that the SOURCE DDNAME is allocated correctly. Verify that the data set and member to parse exist. Verify that the FLMALLOC macro is complete and valid for the parser.

40

Explanation

SCLM was not successful in invoking FLMLRC37 using IKJEFTSR (TSOLNK).

User response

Contact the project manager.

Project manager response

FLMLRC37 does not return a 40. A 40 can be encountered from SCLM for CALLMETH=TSOLNK. TSOLNK is used for executing interpretive REXX. A 40 means IKJEFTSR (TSOLNK) was not successful.

FLMLRDTL REXX DTL Parser

Purpose

This is the DTL Parser translator, written in REXX. Comments and split lines are not supported. The only recognized references are

- <:entity inclname system> or
- <!entity inclname system> or
- <:entity % inclname system> or
- <!entity % inclname system>
- <?inclname>
- <?inclname otherstuff>

Parameters

The following parameters are expected as input for the translator:

- Address of the storage to hold the list of included members
- Size of the storage to hold the list of included members.

Return codes

0

Explanation

Indicates successful completion.

User response

None.

Project manager response

None.

20

Explanation

Too many includes to fit in the storage provided.

User response

Increase the storage.

Project manager response

None.

FLMLRIPF Script and OS/2 IPF Source Parser

Purpose

The FLMLRIPF parser supports Script and OS/2 IPF source files. The parser is written in REXX. The includes found by the parser are associated with an include set that is the extension from the include statement (see “FLMINCLS macro” on page 542).

Functions

The parser uses the following syntax rules to locate dependency names:

- The search for tokens is restricted to uncommented text.

  Lines beginning with .* are recognized as comments.
- Include dependencies are generated in the following conditions:
  - The first token on the line is .im. The second token on the line is the include name. The include set will be the extension from the file name.
  - The first token on the line is :artwork. The token following name= is the include name.
- Tokens inside strings are ignored.

Include names are generated after removing excess characters (all characters up to and including the far-right directory separator character (default is \), if any, and all characters from the first period (.) to the end of the file name). Any underscore characters (_) or blanks are replaced by at-signs ('@'). Include names longer than eight characters are truncated to eight characters and a return code of 4 is issued. The include-set names are generated from the characters following the first period (.) to the end of the file name. Include-set names are also truncated to eight characters and underscore characters and blanks are replaced by at-signs. The following table illustrates how include and include-set names are derived from source statements.

Table 37. Examples of include and include-set names derived from source statements

Source statement	Include name	Include-set name
.im abc	abc	
:artwork name='tile_c_1.bmp' runin.	tile@c@1	bmp

Another function of the parser is to gather statistics or metrics for each module to be parsed. SCLM saves 10 statistics, but only 4 are generated by this parser. This parser defines the ten statistics as follows:

Total lines
The total number of records in the file.

Comment lines
This value is always 0.

Noncomment lines
This is the same as the total lines.

Blank lines
The number of lines that contain only blanks.

Prolog lines
This value is always 0.

Total statements
This value is always 0.

Comment statements
The total number of `/* */` pairs in the member.

Control statements
This value is always 0.

Assignment statements
This value is always 0.

Noncomment statements
This value is always 0.

Parameters

The following guidelines apply when specifying parameters:

- The order of the parameters is not important.
- See the language definitions provided by SCLM for the actual use of the parameters for FLMLRIPF.
- The directory separator character defaults to \.

The DIR_SEPARATOR keyword parameter may be used to specify a directory separator character.

The following keyword parameters, separated by commas, are required as input to FLMLRIPF

LISTINFO
Pointer to the SCLM list information record. This parameter is required and must be set to @@FLMLIS.

LISTSIZE
The size of the LISTINFO buffer. This parameter is required and must be set to @@FLMSIZ. The parser checks to make sure that the LISTSIZE parameter is large enough to hold at least one entry of 228 bytes.

STATINFO
Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

Return codes

0

Explanation

Indicates successful completion.

User response

None.

Project manager response

None.

4

Explanation

The dependency name does not match the source code for one of the following reasons:

- Dependency name greater than 8 characters. The error message in *userid*.SCLMERR.LISTING is:

```
4
line
dependency
```

  where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded eight characters. The truncated form of the dependency is added to the dependency list.
- Unsupported form of include dependency. The error message in *userid*.SCLMERR.LISTING is:

```
4
line
dependency
```

  where *line* is the source line that contains the dependency, and *dependency* is the text of the desired include member.
- #include statement spans multiple lines. The error message in *userid*.SCLMERR.LISTING is:

```
4
line
mark
```

  where *line* is the source line that contains the mark, and *mark* is the mark (either a quotation mark or an angle bracket) that has no matching pair on the line.
- Mismatched quotes. A single or double quote was found that did not have a matching closing quote. The error message in *userid*.SCLMERR.LISTING is:

```
4
mark  line_no
```

  where *line_no* is the line number that contains the unmatched quotation mark, and *mark* is either a single or double quotation mark.

Processing continues.

User response

Change the syntax to fit the parser.

Project manager response

None.

8

Explanation

The number of parsed dependencies exceeds the size of the $list_info array, which is specified by the BUFSIZE parameter on the FLMLANGL macro. The error message in *userid*.SCLMERR.LISTING is:

```
8
line
dependency
```

where *line* is the source line that contains the dependency, and *dependency* is the dependency that exceeded the space allocated for the list.

User response

Either reduce the number of parsed dependencies for the member or contact the project manager.

Project manager response

Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

10

Explanation

FLMTRNSL parameters are incorrect or are not specified.

The error message in *userid*.SCLMERR.LISTING is:

```
10
bad_parms
```

where *bad_parms* are the parameters that are incorrect or not specified.

User response

Contact the project manager.

Project manager response

Verify that the FLMTRNSL parameters for the FLMLRIPF parser are valid and complete.

12

Explanation

Issued by TSOLNK; the parser was not found in the data set specified on the DSNAME parameter of the FLMTRNSL macro.

User response

Contact the project manager.

Project manager response

Verify that the value of the DSNAME parameter on the FLMTRNSL macro is correct.

16

Explanation

Error opening the error listings file.

User response

Contact the project manager.

Project manager response

Ensure that user has the authority to create and write to the file *userid*.SCLMERR.LISTING.

20

Explanation

Error closing the source file.

User response

Contact the project manager.

Project manager response

Contact your IBM service representative.

22

Explanation

An I/O error occurred in the DCB while reading input. Processing stops.

User response

Contact the project manager.

Project manager response

Contact your IBM service representative. Verify that the SOURCE DDNAME is allocated correctly. Verify that the data set and member to parse exist. Verify that the FLMALLOC macro is complete and valid for the parser.

40

Explanation

SCLM was not successful in invoking FLMLRIPF using IKJEFTSR (TSOLNK).

User response

Contact the project manager.

Project manager response

FLMLRIPF does not return a 40. A 40 can be encountered from SCLM for CALLMETH=TSOLNK.

TSOLNK is used for executing interpretive REXX. A 40 means IKJEFTSR (TSOLNK) was not successful.

FLMLSS General Purpose Parser

Purpose

This translator provides an interface to the general purpose SYNTRAN parser. Parsing criteria are provided to this translator through tables.

General information:

- Comments and the contents of quoted strings are ignored.
- DBCS strings (delimited by shift-out and shift-in characters) in comments and quotes are allowed.
- Total lines and blank lines are always counted.
- Control statements and assignment statements are always set to zero.
- The FLMLSS FLMPC370 parser is not case-sensitive.
- Dependencies are truncated to 8 characters before being added to the dependency list.
- Dependencies are ONLY found if they are outside comments. Comment syntax for each table is listed below. No other comment syntax is supported.

  Note: This comment syntax does not match that allowed by some compilers.

Table Name
Syntax

FLMPALST
* indicates a comment that ends at the end of the line.

FLMPBOOK
.* or .CM in column 1 or following a semicolon (;) indicates a comment that ends at the end of the line.

FLMC370
/* or */ turns comments on or off. These delimiters are used interchangeably.

In the following example, bold text indicates areas considered to be comments by the FLMLSS parser.

```
/* Comment 1 */
     #include <i1>
/* Comment 2 /*
     #include <i2>
/* Comment 3
/* Comment 4 */
#include <i3>
/*#include <i4>
```

In the example, include dependencies are found for i1, i2, and i4, but not for i3.

FLMDBRM
No comments are processed.

FLMPJOV
Two types of comments are supported. " turns the first type of comment on or off. % turns the second type of comment on or off. This allows for nesting of comments. Comments are allowed between the !COPY or !COMPOOL statement and the copy or compool name.

In the following example, bold text indicates areas considered to be comments by the FLMLSS parser.

```
        !COMPOOL %COMMENT1%('I1');
        !COMPOOL "COMMENT2"('I2');
        !COMPOOL %COMMENT3('I3');
        !COMPOOL "COMMENT4('I4');
        !COMPOOL "COMMENT5%COMMENT6%"('I5');
        !COMPOOL %COMMENT7"COMMENT8"%('I6');
        !COMPOOL "COMMENT9%COMMENT10%('I7');
        !COMPOOL %COMMENT11"COMMENT12"('I8');
```

In the example, include dependencies are found for I1, I2, I5, and I6, but not for I3, I4, I7 or I8.

FLMPPAS

Two types of comments are supported. (* or *) turns the first type of comment on or off and is used interchangeably. /* or */ turns the second type of comment on or off and is used interchangeably. The two comment delimiters let you nest comments.

In the following example, bold text indicates areas considered to be comments by the FLMLSS parser.

```
         %INCLUDE I1; (* COMMENT 1 *)
         %INCLUDE I2; /* COMMENT 2 */
         %INCLUDE I3; (* COMMENT 3 (*
         %INCLUDE I4; *) COMMENT 4 */
         %INCLUDE I5; (* COMMENT 5 /*NESTED COMMENT 5 */ *)
         %INCLUDE I6; /* COMMENT 6 (*NESTED COMMENT 6 *) */
         %INCLUDE I7; (* COMMENT 7
         %INCLUDE I8; (*COMMENT 8
         %INCLUDE I9;
    /* COMMENT 9
         /* COMMENT 10 */
         %INCLUDE I10;
```

In the example, include dependencies are found for I1, I2, I3, I4, I5, I6, I7 and I9 but not for I8 and I10.

FLMPSCRP

.* or .CM in column 1 or following a semicolon (;) indicates a comment that ends at the end of the line.

Parameters

The following keyword parameters are expected as input to FLMLSS

CONTIN

The column in which the continuation line indicator is set in the input file. If you specify 0 for this parameter, the parser will not concatenate continued lines. The default is column 72.

EOLCOL

The maximum number of characters from each input line to be processed by the parser. The parser ignores any information past this point. The default is 0.

LISTINFO

Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMLIS.

LISTSIZE

The size of the listinfo buffer. This parameter is required and must be set to @@FLMSIZ.

PTABLEDD

The ddname assigned to the parser data set load module. This parameter is required.

SOURCEDD

The ddname assigned to the source file to be parsed. This parameter is required.

STATINFO

Pointer to the SCLM statistics information record. This parameter is required and must be set to @@FLMSTP.

TBLNAME
The name of the parser table load module. This parameter is required. The following tables are provided with SCLM:

FLMPALST
Architecture definition

FLMPBOOK
BookMaster

FLMPC370
C/370

FLMPDBRM
DBRM

FLMPJOV
JOVIAL

FLMPPAS
PASCAL

FLMPSCRP
Script

Return codes

0

Explanation

Indicates successful completion.

User response

None.

Project manager response

None.

4

Explanation

Indicates a warning condition. The limit of 3000 characters for concatenated continuation lines has been exceeded in the input file. The parser ignores any information past the 3000-character limit, but will continue parsing with the next line that is not a continuation line.

User response

In order to remove this warning, modify the input file so that concatenated continuation lines will not exceed the 3000-character limit. If the information past the 3000-character limit is not important, there is no need to change the source file.

Project manager response

None.

8

Explanation

Indicates an error condition. The parser completed successfully, but detected a syntax error in the file being parsed.

User response

Check the input file for syntax errors.

Project manager response

None.

12

Explanation

Indicates an error condition. Unable to load the SCLM table for the parser.

User response

Contact the project manager.

Project manager response

Verify that the user has access to the table through proper library concatenations.

16

Explanation

An invalid input parameter was specified, or a required input parameter was not specified.

User response

Contact the project manager.

Project manager response

Verify that the input parameters are specified correctly in the FLMTRNSL macro that defines this parser. Reassemble the project definition. Verify that no errors occurred. Relink the project definition.

20

Explanation

The number of parsed dependencies exceeds the size of the $list_info array, which is specified by the BUFSIZE parameter on the FLMLANGL macro.

User response

Either reduce the number of parsed dependencies for the member or contact the project manager.

Project manager response

Increase the buffer size (BUFSIZE=) on the FLMLANGL macro for the appropriate language definition. Be sure that LISTSIZE on the FLMTRNSL macro is set to @@FLMSIZ. Reassemble and relink the project definition.

24

Explanation

Indicates an error condition. The parser was unable to load the SCLM table (FLMTABLE) or the multicultural support table.

User response

Contact the project manager.

Project manager response

Verify that SCLM was installed correctly.

FLMLTWST Workstation Build translator

Purpose

The FLMLTWST translator is used to perform compiles, links, or other services on an ISPF connected workstation. It cannot be used when ISPF is accessed from a web browser via the ISPF JAVA environment.

This translator is used for languages that have the source in SCLM and the compiler, linker, or other tools on the workstation. FLMLTWST uses the ISPF SELECT service to execute workstation commands and the FILEXFER ISPF service to transfer files between the host and the workstation. This section describes the FLMLTWST translator as supplied with the ISPF product.

The FLMLTWST translator is written in REXX to let the project manager customize it to fit the needs of the project and workstation tools.

The translator does the following tasks:

- Initialization

  Parses and validates the parameters.
- Reads action information

  The action definitions are read from the ddname specified by the ACTINFODD parameter. This information includes:

  – The names of actions that can be specified with the ACTION parameter
  – The workstation commands and parameters for each action
  – Message file names
  – The mapping between SCLM type names and workstation extensions and subdirectories
  – File transfer format (ASCII or BINARY).
- Gets user-specific information

  Gets information such as which directory to store the files in on the workstation and the response file name.
- Retrieves information from the build map

Gets the list of source members, includes, compiler options, and outputs from the build map. The build map information is obtained by calling the FLMTBMAP translator. The following build map keywords are processed. All other keywords are ignored.

SINC, SINC*

Members on SINC statements:

- Will be transferred to the workstation.
- Can be added to the workstation command depending on the workstation extension that the member's type is mapped to, and the parameter information specified for the workstation command in the FLMLTWST logic.

Innn

Include members identified by Innn statements in the build map will be transferred to the workstation.

COMP, LIST, LMAP, LOAD, OBJ, OUTx

Output members identified by these statements in the build map:

- Will be transferred from the workstation to the ddname associated with that output keyword.
- Can be added to the workstation command depending on the workstation extension that the member's type is mapped to, and the parameter information specified for the workstation command in the FLMLTWST logic.

CMD

The processing of the CMD statement depends on the blank delimited keyword that follows the command statement. CMD statements that do not have one of the keywords listed below will cause an error.

PARMS

The string following this keyword will be added to the workstation command. If multiple CMD PARMS appear in the architecture definition, they will be added to the workstation command in the order they appear. Where the parameters appear in the command in relation to the other parameters (input and output files) is determined by the information in the setup part of the FLMLTWST translator.

There will be no separator character following the value of PARMS in the language definition. If a separator character is desired then one should appear in the PARMS keyword as the last character.

If CMD ACTION statements are present in the build map, the parameters apply only to the workstation command for the action that they follow. If they appear before any CMD ACTION statements, they apply to the workstation command for the action from the ACTION parameter on the FLMLTWST translator definition.

ACTION

The string following this keyword must be a valid action (see the ACTION parameter for this translator). You can use the ACTION keyword to have FLMLTWST issue multiple workstation commands. The first workstation command is the command from the action specified on the ACTION parameter.

The following example shows how to add a binary resource file to an .exe file by specifying ACTION=LINKEXE on the FLMTRNSL in the language definition and using an architecture member.

```
*
LKED EXE               * use the EXE language
*
CMD PARMS /O+ /Ss
*
KREF OBJ               * OBJBIN is generated by an OBJ keyword
*
INCLD SAMPLE C         * includes SAMPLE OBJBIN
*
INCLD COMMON C         * includes COMMON OBJBIN
*
```

```
LOAD SAMPLE EXEBIN
LMAP SAMPLE LMAP
*
CMD ACTION RCEXE       * Add resources to the .exe
*
KREF OUT1              * RESBIN is generated by an OUT1 keyword
*
INCLD SAMPLE RC        * includes SAMPLE RESBIN
*
```

This causes two workstation commands to be issued. First the sample.exe file is generated; then the resource compiler adds the resources in the sample.res file to the sample.exe file. The sample.exe file is not stored into SCLM until after the resource compiler has run.

- Generates a response file if needed.

  Workstation compilers and other tools that support response files will have one generated. The response file contains the parameters for the compiler or other tool.
- Transfers the inputs (source, includes, and so on) to the workstation.

  The source members, includes, and response file are transferred to the workstation using the FLMTXFER translator. If multiple workstation commands are being issued, the response files for all commands after the first will be transferred to the workstation just before issuing the command.
- Runs the workstation command.

  Constructs the workstation command and sends it to the workstation.
- Transfers the outputs (obj, exe, dll, and so on) to the host.

  Transfers the outputs to the host using the FLMTXFER translator. The SCLM outputs are placed in ddnames allocated by FLMALLOC so that build can place them into the hierarchy.

Parameters

The following parameters are specified in the OPTIONS list for the FLMTRNSL macro that has COMPILE=FLMLTWST. All parameters are keyword parameters and can be specified in any order. Parameters must be separated by commas. Extraneous parameters are ignored without any messages being produced. An FLMALLOC is required for the following data definitions:

- MESSAGEDD
- MSGXFERDD
- RESPONSEDD
- FILESDD
- BMAPDD
- USERINFODD
- ACTINFODD

ACTION=COMPILE|action_name
This parameter is optional. If not specified, it defaults to COMPILE. The valid values are specified in the ACTINFO data set.

BMAPINFO=@@FLM$MP
This parameter is required and must be specified in the options list with the value from @@FLM$MP.

BLDINFO=@@FLMBIO
This parameter is required and must be specified in the options list with the value from @@FLMBIO.

SCLMINFO=@@FLMINF
This parameter is required and must be specified in the options list with the value from @@FLMINF.

PARMS=command_parms
This parameter is optional. If specified, the string that follows it will be added as the first parameter on all workstation commands.

MESSAGEDD=dd_name
This parameter is optional. If not specified, it defaults to MESSAGE. This is the ddname where messages are written.

MSGXFERDD=dd_name
This parameter is optional. If not specified, it defaults to MSGXFER. This is the ddname to temporarily hold messages from the workstation command. The messages are appended to the data set specified by the MESSAGEDD parameter. The FLMALLOC for this ddname must specify CATLG=Y.

RESPONSEDD=dd_name
This parameter is optional. If not specified, it defaults to RESPONSE. This is the ddname used to generate the response file for the workstation if the workstation command requires a response file. The FLMALLOC for this ddname must specify CATLG=Y.

FILESDD=dd_name
This parameter is optional. If not specified, it defaults to FILES. This ddname is used to communicate between the FLMLTWST and FLMTXFER translators. See the description of the content of this ddname in the FLMTXFER translator description.

BMAPDD=dd_name
This parameter is optional. If not specified, it defaults to BMAP. This ddname is used to communicate between the FLMLTWST and FLMTBMAP translators. See the description of the content of this ddname in the FLMTBMAP translator description.

USERINFODD=dd_name
This parameter is optional. If not specified, it defaults to USERINFO. This ddname contains the information about the workstation where the tools will be run. Each line in the data set allocated to the ddname can contain either a comment or keyword and its value. Comment lines begin with an asterisk (*) and are ignored. Lines that contain invalid keywords are ignored. Keywords and their values must be separated by one or more spaces.

ACTINFODD=dd_name
This parameter is optional. If not specified, it defaults to ACTINFO. This ddname contains the information about the workstation actions such as COMPILE and LINKEXE. Each line in the data set allocated to the ddname contains either a comment or a keyword. Comment lines begin with an asterisk (*) and are ignored. Lines that contain invalid keywords are ignored. Keywords on the statement must be separated by at least one space.

output_keyword=dd_name
These parameters can be used to specify the ddnames to hold the output for each build output keyword. These parameters are not required. If not specified, the ddname is the same as the keyword. An example of these parameters is OBJ=SYSIN,LIST=SYSPRINT. This example defines FLMALLOC statements for the SYSIN and SYSPRINT ddnames with IOTYPE=O or P. If these parameters had not been specified, there would be FLMALLOC statements for OBJ and LIST ddnames with IOTYPE=O or P.

The FLMALLOC statements for the output ddnames must specify CATLG=Y. All allocations must be IOTYPE=O or P. If CATLG=Y is not specified, the file transfer will fail.

USERINFODD statements

Valid keywords for statements in the USERINFODD data set are:

Keyword
Value Description

Asterisk (*)
Comment lines start with asterisks and are ignored.

RESPONSE_FILE
The name of the response file on the workstation. The file name must include the full path, because the DATA_DIR value will not be prefixed to the file name. The default is response.fil.

DEL_CMD
The delete command to be used on the workstation. The default is DEL. The message files that are to be created by a workstation command will first be deleted using this command. This is done so that message files with the same name from previous commands are not transferred to the host after completion of a workstation command.

DATA_DIR
The base directory on the workstation where the files are stored. DATA_DIR must include the full directory name. All SCLM members are transferred to and from this directory and its subdirectories. The subdirectories are based on the subdirectory value specified in the ACTINFO file. This defaults to '\'. The FLMLTWST translator supplied by IBM appends the subdirectory to the DATA_DIR value before appending the file name.

MODE
MODE specifies how the command is to start on the workstation. MODE may be:

- MIN (minimized - the default)
- MAX (maximized)
- VIS (visible, if possible)
- INVIS (invisible, if possible).

A MODE value specified in USERINFODD will override a MODE value specified in ACTINFODD. For more information about MODE, see the ISPF SELECT service in the *z/OS ISPF Services Guide*.

WSDIR
WSDIR specifies the workstation directory. This is the directory from which the workstation command will be executed. The default is the directory from which ISPF Client/Server is running. The WSDIR value from the ACTINFODD data set will be concatenated after the WSDIR value from the USERINFODD data set.

ACTINFODD statements

Statements in the ACTINFODD file are composed of a keyword and a value. The keywords TYPE, EXTENSION, TRANSFER_FORMAT, and SUBDIRECTORY are used to define the SCLM-type-to-workstation-file-extension mapping information to SCLM. The other keywords in this file are used to specify information about each workstation command. Keywords CPARM, EXTENSION, MESSAGE_FILE, and RPARM apply only to the previous ACTION keyword. Valid keywords for statements in the ACTINFO data set are:

Keyword
Value Description

Asterisk (*)
Comment lines start with asterisks and are ignored.

WSDIR
WSDIR specifies the workstation directory for executing the command. The WSDIR value from the ACTINFODD data set will be concatenated after the WSDIR value from the USERINFODD data set.

QUOTE
The character to use for quoting strings in CPARM, CPARMSEP, and RPARM statements. This character is used for statements that follow this QUOTE statement until the next QUOTE statement is found. The default quote character is a single quote (').

ACTION
The name of an action that can be specified on the ACTION parameter to FLMLTWST.

COMMAND
The command to be issued on the workstation for the previous action. If multiple COMMAND statements are found following an ACTION statement only the last COMMAND is used.

MODE
MODE specifies how the command is to start on the workstation. MODE may be:

- MIN (minimized - the default)

- MAX (maximized)
- VIS (visible, if possible)
- INVIS (invisible, if possible).

A MODE value specified in USERINFODD will override a MODE value specified in ACTINFODD. For more information about MODE, see the ISPF SELECT service in the *z/OS ISPF Services Guide*. The default MODE is MIN.

CPARM
Parameters to add to the last workstation COMMAND. The parameters are added to the command in the order they are found in the file. Each parameter is made up of three parts: a prefix, a type name, and a suffix. The type name indicates that the parameters on this CPARM statement only apply to members in SCLM of that type. If no type name is specified, the parameter is added to the command. In the parameter string, the SCLM type name is replaced with the name of SCLM members of that type that are inputs or outputs to the build. The parameters added to the command are composed by concatenating the prefix, the workstation file name for the SCLM member, and the suffix. If multiple members match the type on the CPARM statement, the prefix and suffix are concatenated to each file name. See the examples at the end of the description of FLMLTWST for more information.

- A prefix string. The string must be surrounded by quotes if it contains blanks.
- An SCLM type name. If the type name is specified, the parameters are added if there is an input or output of this type to the command. If there are inputs and outputs of this type, the file name containing each input and output is added to the workstation command preceded by the prefix string and followed by the suffix string.

  No blanks are placed between the file name and the prefix and suffix strings. To get blanks between the prefix and suffix strings and the file name, use quotes around the strings and put the blank inside the quotes.
- A suffix string. The string must be surrounded by quotes if it contains blanks. This string defaults to blank if not specified.

The following variables can be specified in the prefix and suffix strings:

&CMD_PARMS&
Is replaced with the parameters specified on "CMD PARMS" statements in the architecture definition if there are any. "Null" will be used when "CMD PARMS" is not found. &CMD_PARMS& must be present in order to use a CMD PARMS statement in the architecture definition.

&RESPONSE_FILE&
Is replaced with the response file name.

&DATA_DIR&
Is replaced with the base directory from the user information.

CPARMSEP
The value to be used as a separator between the command parameter strings specified by the CPARM keywords. No separator character will be added to the end of the parameter string. Also, there will be no separator character following the value of PARMS in the language definition. If you want a separator character, then it should appear in the PARMS keyword as the last character.

- CPARMSEP NULL results in no separator character.
- CPARMSEP by itself results in a blank being used as the separator character.
- CPARMSEP with a quoted string will have the quotes removed from the front and back if the quote characters match the value of the QUOTE keyword. (If there is no QUOTE keyword the single quote default will be used.) An error message will appear if the first character is a quote (as per the value of the QUOTE keyword) and the last character is not a quote.
- No separator character will be added to the end of the parameter string.

RPARM
Parameters to add to the response file for the last workstation command. The format of this statement is the same as the CPARM statement. Use CPARM if the parameters are specified as part of

the workstation command. Use RPARM if the parameters are to be put in a response file that the workstation command will read. Only use response file parameters if the workstation command supports a response file. If parameters are specified in a response file, make sure the response file name is specified on one of the CPARM statements if the workstation command requires it.

The variables described for CPARM can also be used on RPARM statements.

TYPE
The name of an SCLM type to transfer to the workstation. The type name can include a single asterisk (*) as a wildcard character.

EXTENSION
The workstation extension to use for the types from the previous TYPE statement. A single asterisk (*) can be included and is replaced with any characters that matched the * in the TYPE statement. The default value is *.

The value is either a single asterisk or a character string. Strings using an asterisk and other characters (such as H*) will result in the asterisk being used as part of the extension.

SUBDIRECTORY
The subdirectory to use for the file names derived from the previous TYPE statement. The subdirectory is placed between the data directory as specified by the DATA_DIR keyword in the USERINFODD user information and the file name to construct the fully-qualified workstation file name. The default value is a \.

TRANSFER_FORMAT
The transfer format for files of the types from the previous TYPE statement. Valid values are:

ASCII
Translate the file to ASCII format. This is the default.

BINARY
Perform no translation.

WSCASE
The case for workstation file names. Valid values are:

UPPER
Transfer the files to or from the workstation with the workstation file name in uppercase letters. This is the default setting.

LOWER
Transfer the files to or from the workstation with the workstation file name in lowercase letters.

ULOWER
Transfer the files to or from the workstation with the workstation file name having an initial capital letter followed by lowercase letters.

MESSAGE_FILE
The name of a message file that was created on the workstation. It will be written to the ddname specified by the MESSAGEDD parameter. This statement can be used to get the messages from the workstation command into the BUILD.LISTxx data set. Multiple MESSAGE_FILE statements can be specified. Each MESSAGE_FILE statement applies to the previous ACTION. The default message file name is c:\sclm.msg.

Environment

The FLMTXFER translator must have access to ISPF services. FLMLTWST must be invoked by specifying CALLMETH=ISPLNK for the FLMTRNSL macro.

Return codes

In addition to the return codes listed here, messages can be written to the ddname specified by the MESSAGEDD parameter.

User Information example

Here is a sample USERINFO data set, which can be found in the FLMWBUSR member in the ISPF sample library.

```
*
* Data_dir. Default is c:\.
* The subdirectory value from the actinfo file or the default of '\'
* will be appended to this value before the name of the workstation
* file. This directory and its subdirectories will be where SCLM
* transfers files to/from on the workstation.
*
data_dir     c:\wb
*
* Wsdir. Default is ISPF Client/Server directory.
* Directory from which the command will be executed. If a WSDIR
* keyword is also specified in the ACTINFO file, then it will be
* concatenated to the end of what is specified here.
* For commands that must run from the directory holding the data
* files (such as the resource compiler), this value should be the
* same as the data_dir value.
*
wsdir        c:\wb
*
* Response_file. Default is c:\response.fil.
* Fully qualified name of the response file to contain all RPARM
* statements for an action. The name should include the drive and
* any subdirectories.
*
response_file c:\wb\response.fil
*
* Mode. Workstation build default is minimized. This will override any
* value set in the ACTINFO file.
* This is the mode for the workstation command. Valid values are:
* MAXimized, MINimized, VISible and INVISible. See the WSCMDV section
* of the ISPF SELECT service for additional information.
*
mode   max
*
* Del_cmd. Default is 'del'.
* This keyword specifies the delete command to be used to remove the
* message file(s) from previous builds or build steps from the
* workstation before executing the workstation command.
*
del_cmd erase
```

ACTINFO example

The following example shows an ACTINFO data set and can be found in the FLMWBAIO member in the ISPF sample library.

```
/*********************************************************************/
*                                                                   *
* SCLM type to workstation file name mapping                        *
*                                                                   *
* The following statements define the mapping between SCLM type names *
* and workstation file extensions as well as the transfer format      *
* for the data.  The statements are processed in order.  If the       *
* member being processed does not match the first TYPE criteria, then *
* it will be compared to the next TYPE criteria and so on until a     *
* match is found.                                                   *
*                                                                   *
/*********************************************************************/
*                                                                   *
* Types containing BINARY data                                      *
*                                                                   *
* This mapping indicates that the host SCLM type will be the          *
* workstation file extension followed by BIN.   The BIN pattern in    *
* the type name will be used to indicate that members in these types  *
* contain binary data.  As an example, a member stored in the OBJBIN  *
* type in SCLM, will be transferred as binary (no ASCII-to-EBCIDIC    *
* conversion) with the workstation file name 'member.OBJ'.            *
*                                                                   *
/*********************************************************************/
TYPE            *BIN
EXTENSION       .*
TRANSFER_FORMAT BINARY
```

```
SUBDIRECTORY    \
*************************************************************************
*                                                                       *
* Types containing ASCII data                                           *
*                                                                       *
* This mapping indicates that any members whose type does not match     *
* the previous criteria (TYPE * will always match), will be processed *
* as ascii/text files.  The workstation file extension will be the      *
* same as the SCLM type.  As an example, a member in the CPP type in    *
* SCLM will have the workstation file name 'member.CPP'.                *
*                                                                       *
*************************************************************************
TYPE            *
EXTENSION       .*
TRANSFER_FORMAT ASCII
SUBDIRECTORY    \
*
*************************************************************************
*************************************************************************
*                                                                       *
* Workstation Commands                                                  *
*                                                                       *
* The following statements define the actions/commands to be executed *
* on the workstation.                                                   *
*                                                                       *
*************************************************************************
*
* C and C++ compile to generate object
*
ACTION      COMPILE
COMMAND     icc
RPARM -Fd
RPARM -c
RPARM -Gm+
RPARM -O+
RPARM &CMD_PARMS&
RPARM /Fo OBJBIN
RPARM /Fl LST
RPARM '' C
RPARM '' CPP
CPARM @&RESPONSE_FILE&
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Preprocessor only
*
ACTION      PREPROCESS
COMMAND     icc
RPARM /Pe+
RPARM &CMD_PARMS&
RPARM '' C
RPARM '' CPP
RPARM '' IPF
RPARM '' RC
CPARM @&RESPONSE_FILE&
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Dummy file for resource DLLs
* (compile with /Ge- option)
*
ACTION      DUMMY
COMMAND     icc
RPARM -Fd
RPARM -c
RPARM /Ge-
RPARM &CMD_PARMS&
RPARM /Fo OBJBIN
RPARM /Fl LST
RPARM '' C
RPARM '' CPP
CPARM @&RESPONSE_FILE&
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Link object into exe using CSET++
```

```
*
ACTION      LINKEXE
COMMAND     icc
RPARM /Ge+
RPARM &CMD_PARMS&
RPARM /Fe EXEBIN
RPARM /Fm MAP
RPARM '' OBJBIN
RPARM '' DEF
CPARM @&RESPONSE_FILE&
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Link object into dll using CSET++
*
ACTION      LINKDLL
COMMAND     icc
RPARM /Ge-
RPARM &CMD_PARMS&
RPARM /Fe DLLBIN
RPARM /Fm MAP
RPARM '' OBJBIN
RPARM '' DEF
CPARM @&RESPONSE_FILE&
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Link object into exe or dll using LINK386
*
ACTION      LINK386
COMMAND     link386
CPARMSEP NULL
CPARM &CMD_PARMS&
CPARM '' OBJBIN
CPARM ,
CPARM '' DLLBIN
CPARM '' EXEBIN
CPARM ,
CPARM '' MAP
CPARM ,
CPARM '' LIBBIN
CPARM ,
CPARMSEP
CPARM '' DEF
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Generate res file
*
ACTION      RC
COMMAND     rc
CPARM -r
CPARM '-i &DATA_DIR&'
CPARM &CMD_PARMS&
CPARM '' RC
CPARM '' RESBIN
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Apply res file to an exe or dll
*
ACTION      RCEXE
COMMAND     rc
CPARM &CMD_PARMS&
CPARM '' RESBIN
CPARM '' EXEBIN
CPARM '' DLLBIN
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Generate hlp file
*
```

```
ACTION      IPFC
COMMAND     ipfc
CPARM '' IPF
CPARM &CMD_PARMS&
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
*
* Generate hlp file - input file specified on CMD statement
*
ACTION      IPFCP
COMMAND     ipfc
CPARM &CMD_PARMS&
CPARM 1>&DATA_DIR&\stdout.msg
CPARM 2>&DATA_DIR&\stderr.msg
MESSAGE_FILE &DATA_DIR&\stdout.msg
MESSAGE_FILE &DATA_DIR&\stderr.msg
```

The following example shows an architecture definition and the resulting workstation commands using the previous ACTINFO data set. This architecture definition can be found in the ISPF sample library, member FLMWBSLE.

The architecture definition:

```
*
LKED EXE                * link language
*
KREF OBJ                * include generated object modules
*
INCLD MAHJONGG C        * MAHJONGG SOURCE
INCLD TILE     C        * TILE SOURCE
SINC  MAHJONGG DEF      * DEF source
*
LOAD  MAHJONGG EXEBIN   * Generated .exe file
LMAP  MAHJONGG MAP      * Generated .map file
*
* Run resource compiler after the link completes
*
CMD ACTION RCEXE
*
KREF OUT1               * include generated .res file
*
INCLD MAHJONGG RC       * Source which produces MAHJONGG RESBIN
*
```

The language EXE (as specified by the LKED keyword) uses the action LINKEXE. This results in the following command and response file:

The command

```
icc @c:\wb\response.fil 1>c:\wb\stdout.msg 2>c:\wb\stderr.msg
```

The contents of the response file, c:\wb\response.fil

```
/Ge+
/Fec:\wb\MAHJONGG.EXE
/Fmc:\wb\MAHJONGG.MAP
c:\wb\MAHJONGG.OBJ
c:\wb\TILE.OBJ
c:\wb\MAHJONGG.DEF
```

The CMD ACTION statement results in a second action, RCEXE. RCEXE issues the following command:

```
rc c:\wb\MAHJONGG.RES c:\wb\MAHJONGG.EXE 1>c:\wb\stdout.msg 2>c:\wb\stderr.msg
```

Language definition example

“Sample language definition” on page 314 shows a language definition using FLMLTWST to compile C or C ++ source on the workstation. It also includes a description of the items used in the language definition. This sample can be found in the ISPF macro library as member FLM@WICC.

0

Explanation

The workstation build was successful.

Project manager response

None.

1-900,
904-908, >999

Explanation

Workstation command error.

User response

Review the messages from the workstation command. See the ISPF SELECT service in the *z/OS ISPF Services Guide* for additional information about problems with executing the command.

Project manager response

None.

901

Explanation

The call to FLMTBMAP failed.

User response

Check the return code from FLMTBMAP.

Project manager response

None.

902

Explanation

The call to FLMTXFER to transfer the inputs to the workstation failed.

User response

Check the return code and messages from FLMTXFER. Messages are in the ddname specified by the MESSAGEDD parameter. See the FILEXFER ISPF service for additional information about problems with transferring files.

Project manager response

None.

903

Explanation

The call to FLMTXFER to receive outputs from the workstation failed.

User response

Check the return code and messages from FLMTXFER. Messages are in the ddname specified by the MESSAGEDD parameter. See the FILEXFER ISPF service for additional information about problems with transferring files.

Project manager response

None.

999

Explanation

An error occurred in FLMLTWST.

User response

Check the messages from FLMLTWST.

Project manager response

None.

FLMTBMAP Build Map Print - Build translator

Purpose

The FLMTBMAP translator generates a DBUTIL (Database Contents Utility) style printout of the information in the build map of the member being built. The information in the report matches the build map that will be saved if the build of the member is successful. The information in this report may be different from the information in the build map currently stored in VSAM.

Information about outputs that do not have the member name specified do not show up in the report. The output information will be missing in the following conditions:

- An architecture member is being built and the output is the result of an output keyword with an '*' as the member name,
- A nonarchitecture member is being built and the output is the result of an FLMALLOC with IOTYPE=P and no DFLTMEM was specified.

For architecture members that include the outputs of other members through INCL or INCLD statements, the outputs that are included are identified by a SINC* keyword before the output.

FLMTBMAP is a build translator and can be run only within the build environment.

For an example of the usage of this translator, see "FLMLTWST Workstation Build translator" on page 608.

Parameters

All parameters are keyword parameters and can be specified in any order. Parameters must be separated by commas. Extraneous parameters are ignored without any messages being produced.

BMAPINFO=@@FLM$MP
This parameter is required and must be specified in the options list with the value from @@FLM$MP.

SCLMINFO=@@FLMINF
This parameter is required and must be specified in the options list with the value from @@FLMINF.

BLDINFO=@@FLMBIO
This parameter is required and must be specified in the options list with the value from @@FLMBIO.

BMAPDD=dd_name
This parameter is optional. It specifies the ddname where the build map report will be written. If it is not specified, the FLMTBMAP translator will attempt to write the report to a ddname of BMAP. This parameter will be truncated to 8 characters.

Return codes

0

Explanation

The report was generated successfully.

User response

None.

Project manager response

None.

4

Explanation

The build map is empty, no report was produced.

User response

None.

Project manager response

None.

8

Explanation

The ddname for the report is not allocated.

User response

Contact the project manager.

Project manager response

Ensure that the language definition has an FLMALLOC for the ddname specified by the BMAPDD parameter.

12

Explanation

The value for BMAPINFO is not valid.

User response

Contact the project manager.

Project manager response

Ensure that the BMAPINFO parameter is specified in the options list, that the parameter has a value of @@FLM$MP, and that the translator has FUNCTN=BUILD.

16

Explanation

The value for SCLMINFO is not valid.

User response

Contact the project manager.

Project manager response

Ensure that the SCLMINFO parameter is specified in the options list and that the parameter has a value of @@FLMINF.

FLMTMJI Interface to JOVIAL Compiler

Purpose

This translator provides an interface to the Jovial compiler. The translator provides memory management for the compiler. The Jovial compiler can require a single large contiguous area of memory in which to work. FLMTMJI attempts to provide an area of memory at least 512K for the compiler.

Consider using this translator to invoke the Jovial compiler if you receive abends from the compiler. The abends typically result from large builds.

This translator assumes that the entry point name of the Jovial compiler is JOVIAL. Furthermore, FLMTMJI must be in the same load module data set as the Jovial compiler unless the compiler is in the MVS link pack area (LPA).

Parameters

FLMTMJI passes its parameters directly to the Jovial compiler. You should pass the parameters you would normally send directly to the compiler.

Return codes

FLMTMJI uses the Jovial compiler return codes with the exception of return code 64.

64

Explanation

Less than 512K of contiguous memory is available for use by the compiler. This can happen either when there is little memory available or when available memory becomes fragmented (for example, a build involving many compilations).

User response

Try the build again with a larger region size.

Project manager response

None.

FLMTMMI Interface to DFSUNUB0 (phase 2 of MFSUTL and MFSTEST)

Purpose

This translator invokes DFSUNUB0 via the TSO Service Facility. At completion, a return code is passed to the calling program in register 15. Ddname substitution lists are not supported in this translator.

FLMTMMI invokes DFSUNUB0 using the TSO service facility IKJEFTSR with link flags of hexadecimal '00000002'. These flags direct the TSO service facility to invoke DFSUNUB0 from an authorized environment. DFSUNUB0 may be invoked directly as the value of the COMPILE keyword parameter for FLMTRNSL if COMPRESS is not in the OPTION list.

Note: DFSUNUB0 can not be directly invoked from SCLM when using the COMPRESS option. The COMPRESS option causes IEBCOPY to execute, which would then result in an abnormal termination because the SCLM address space is not an authorized environment.

DFSUPAA0 (phase 1 of MFSUTL and MFSTEST) is invoked directly by the IMS MFS language definitions. You might need to change the options list for DFSUPAA0 based on your installation.

Parameters

The parameters in the options list are passed directly to the DFSUNUB0. For CSP/370AD map groups, the parameters should match those found in the CSP/370AD-generated preparation JCL. For other IMS applications these parameters can be changed as required.

This translator produces multiple outputs. The outputs are stored outside of the control of SCLM.

Currently, the options list that is included with the IMS MFS language definitions for the FLMTMMI translator contains the following parameters:

FLM@MVTS
The following option list is required for MFSTEST: TEST,DEVCHAR=C

FLM@MVUT
The following option list is required for MFSUTIL: COMPRESS,NOCOMPREND,UPDATE,DEVCHAR=C

Verify the value assigned to DEVCHAR to ensure that it matches the value used for the target IMS region. Check with your IMS system programmer to determine the correct value.

Return codes

This translator passes back one of the following return codes:

- 0 - Successful completion.
- 4-20 - The return code set by DFSUNUB0. For more information on these return codes, refer to IMS in IBM Knowledge Center (www.ibm.com/support/knowledgecenter/SSEPH2).
- 40 - Error invoking DFSUNUB0. This should be accompanied by messages in the JOB listing describing the error. If the BUILD or PROMOTE function was invoked in the foreground, ensure that the WTPMSG option is enabled in the TSO PROFILE to see these messages.

FLMTMSI Interface to SCRIPT/VS

Purpose

This translator provides an interface to SCRIPT/VS via the TSO Service Facility.

SCRIPT is a TSO command and needs to be invoked by using the TSO command processor interface. It cannot be invoked directly by FLMTRNSL. FLMTMSI builds a SCRIPT command with a concatenation of the string following '/' in the OPTIONS list.

Parameters

The options string passed to this translator should contain the user ID to be used by SCRIPT/VS, delimited by a slash (/), and followed by a list of desired SCRIPT/VS options. For example,

```
OPTIONS=(@@FLMUID/DEV(3800N8),CH(GT12,GB12),TW,CO,      C
B(7,7),M(DELAY,TRACE,ID))
```

Return codes

For information relating to the return codes 0 through 20 and return code 40, refer to *DCF: SCRIPT/VS Messages*.

24

Explanation

SCLM did not allocate TEXTOUT.

User response

Contact the project manager.

Project manager response

Verify that an FLMALLOC macro has been coded for this translator with a ddname of TEXTOUT for the SCRIPT/VS output file. Reassemble the project definition. Verify that no errors occurred. Relink the project definition. For more information see “FLMALLOC macro” on page 494.

28

Explanation

SCLM did not allocate TEXTIN.

User response

Contact the project manager.

Project manager response

Verify that an FLMALLOC macro has been coded for this translator with a ddname of TEXTIN for the SCRIPT/VS source file. Reassemble the project definition. Verify that no errors occurred. Relink the project definition. For more information see “FLMALLOC macro” on page 494.

36

Explanation

The user ID was not specified in the input, or the parameter list has an incorrect format.

User response

Contact the project manager.

Project manager response

Verify that the options list is in the correct format for this translator. Reassemble the project definition. Verify that no errors occurred. Relink the project definition.

FLMTPRE

Purpose

FLMTPRE is the precompile processor called before a translator that processes input lists. FLMTPRE supports both non-Ada and Ada input lists. It initializes the ADA library file (DDNAME=ADALIB) with the names of the sublibraries required to perform the ADA compile by the next translator. It also initializes the input list file (DDNAME=ADAIN) with the names of the source members to be compiled. The translator that processes the data sets uses this list of data sets as input.

The input list data set allocated to the ADAIN ddname contains a list of members in compilation order to be built. Each member is listed in the data set using its fully qualified name enclosed in single quotes. Here is a description of the ADAIN format and a sample input list.

```
Start
Column   Length    Description
   1       56      fully qualified project partitioned data set name,
                   enclosed in single quotes.
'PROJECT1.RELEASE.SOURCE(SUB4)'
'PROJECT1.INT.SOURCE(PROC2)'
'PROJECT1.STAGE.SOURCE(SUB2)'
'PROJECT1.USER.SOURCE(PROC1)'
'PROJECT1.INT.SOURCE(MAIN)'
```

If a non-Ada language is used, direct the ddname ADALIB to NULLFILE:

```
   FLMALLOC IOTYPE=W,DDNAME=ADALIB
   FLMCPYLB NULLFILE
```

and do not specify parameters SUBLIB1...SUBLIB8 on the FLMTRNSL macro for calling FLMTPRE.

The names of the sublibrary data sets will be placed in the Ada Library file by FLMTPRE. The names are listed in the following order:

```
sublibraries controlled by SCLM
sublibraries not controlled by SCLM
```

You can pass up to 8 sublibraries NOT under SCLM control to FLMTPRE using the SUBLIB# parameter (as noted below). These sublibraries are usually system-level or runtime sublibraries.

The CU qualifier used to create the sublibrary names is the SUFFIX specified on the input parameter string. If the SUFFIX parameter is not specified, the cu_qual on the FLMLANGL macro is used to generate sublibrary names.

Parameters

The following keyword parameters are expected as input to FLMTPRE

DDMSGS
This is an optional parameter to specify a ddname for messages. The default is FLMTMSGS.

FLM_INFO
This parameter is used to access the list of members to be placed into the input list file. The name of each member is placed into the input list file, and then the input list file is allocated to the ddname ADAIN. This parameter is required and must be set to @@FLMINF.

SUBLIB1...SUBLIB8
Sublibrary not under SCLM control to be added. These are optional parameters. You can specify up to 8 of these sublibraries. When not specified, these parameters default to DUMMY.

SUFFIX
Suffix to use when generating sublibrary names (that is, the CU qualifier). This parameter is optional. If it is not specified, the CU qualifier on the FLMLANGL macro is used to generate the sublibrary name.

Return codes

0

Explanation

Indicates a successful completion.

User response

None.

Project manager response

None.

4

Explanation

The sublibrary name is longer than MVS allows. The sublibrary name is formed by concatenating the suffix to the project data set name specified for the group and type being processed.

User response

Contact the project manager.

Project manager response

The physical data set name concatenated with the suffix cannot exceed 44 characters. The data set name must be shortened in the project definition.

8

Explanation

The FLM_INFO parameter does not specify a valid SCLM information record pointer.

User response

Contact the project manager.

Project manager response

The FLM_INFO parameter is either missing or does not specify a valid pointer to the SCLM information record. This parameter should be specified as follows: FLM_INFO=@@FLMINF. Correct this parameter on the translator definition in the project definition being used. Regenerate the project definition. Submit the job again.

FLMTPST

Purpose

This translator is the Input List compiler post-compile processor. The Input list feature supports translators that allow the user to specify a list of input data sets for each invocation. Because the translator performing the processing will be operating on a list of files, a list of return codes must be provided to SCLM to correctly manage the build. The FLMTPST translator passes return code information back to SCLM.

The FLMTPST translator takes as input the file allocated to the ADAOUT ddname. The data set pointed to by the ddname ADAOUT must contain lines beginning with * or RC=XX, with the first nonblank character in column one. Each line within the data set that contains RC= in columns 1-3 is processed. Lines beginning with an asterisk (*) are considered comments and are ignored. Each line containing RC= in columns 1 through 3 must follow this format:

```
RC=XX 'DATA SET NAME(MEMBER)'
```

The data set name is the same as that specified in the Input list generated by the FLMTPRE translator. The lines in the ADAOUT file must match the order of the lines in the ADAIN file that was generated by FLMTPRE. If the order does not match, an error is generated and the FLMTPST translator stops. Here is the format and a sample of the ADAOUT file.

```
Start
Column   Length    Description
  1        1       The character "*" followed by anything up to 255 characters
                      - OR -
  1        3       String 'RC='
  4        2       Two-character integer return code
  6        1       Blank space
  7       56       fully qualified project partitioned data set name,
                   enclosed in single quotes.
```

```
RC=00 'PROJECT1.RELEASE.SOURCE(SUB4)'
RC=04 'PROJECT1.INT.SOURCE(PROC2)'
RC=00 'PROJECT1.STAGE.SOURCE(SUB2)'
RC=04 'PROJECT1.USER.SOURCE(PROC1)'
RC=00 'PROJECT1.INT.SOURCE(MAIN)'
```

Parameters

FLMTPST requires the following keyword parameter

FLM_INFO
Pointer to the SCLM information record. This parameter must be set to @@FLMINF.

Return codes

0

Explanation

Indicates a successful completion.

User response

None.

Project manager response

None.

4

Explanation

At lease one, but not all, of the members processed were built successfully.

User response

Check the build messages produced to determine which members failed to build successfully.

Project manager response

None.

8

Explanation

The FLM_INFO parameter does not specify a valid SCLM information record pointer.

User response

Contact the project manager.

Project manager response

The FLM_INFO parameter is either missing or does not specify a valid pointer to the SCLM information record. This parameter should be specified as follows: FLM_INFO=@@FLMINF. Correct this parameter on the translator definition in the project definition being used. Regenerate the project definition. Submit the job again.

12

Explanation

The compiler output file allocated to ddname ADAOUT contains some unexpected information.

User response

Contact the project manager.

Project manager response

Verify that the file allocated to the ADAOUT ddname is the output file generated by the input list compiler. To do this, look at the contents of the file allocated to ADAOUT. Specify the PRINT=Y parameter on the FLMALLOC macro for the ADAOUT file allocation for the input list compiler translator definition. Regenerate the project definition you are using, and submit the job again. The job output will contain the contents of the file allocated to ADAOUT.

For IBM Ada/370, the compiler documentation lists the contents of the output file generated by the input list compile. Verify that the contents of the ADAOUT file printed in the job output conform to the documented values in the compiler documentation. If they do not match, report this problem to your IBM service representative.

When the information in the ADAOUT file is not as expected, contact your IBM service representative for assistance.

16

Explanation

Indicates that a line in the output file allocated to ddname ADAOUT has an unexpected format.

User response

Contact the project manager.

Project manager response

See the project manager response described for return code 12 of this translator.

20

Explanation

Indicates the file allocated to the ADAOUT ddname is empty.

User response

Verify that the input list compiler was successfully invoked and produced an output file. If necessary, contact the project manager.

Project manager response

Verify that the ADAOUT ddname was allocated for the invocation of the input list compiler. If necessary, add an FLMALLOC macro for the ADAOUT ddname to the input list compiler translator definition. Regenerate the project definition and submit the job again. If this problem recurs, report this problem to your IBM service representative for assistance.

24

Explanation

Indicates that the ADAOUT ddname is not allocated.

User response

Contact the project manager.

Project manager response

This could indicate improper usage of the FLMTPST translator. The FLMTPST translator should be invoked only after the input list compiler has been invoked. Verify that the language definition being used is for input list processing and that the FLMTPST translator is being invoked after the input list translator.

Also, verify that there is an FLMALLOC macro specified for the ADAOUT ddname for a previous build translator in the current Ada language definition. If either of these problems are present, change the language definition, the project definition, or both to correct the problem. Regenerate the project definition and submit the job again.

FLMTXFER Workstation Transfer - Build translator

Purpose

The FLMTXFER translator uses the FILEXFER service to send and receive files from a workstation. When sending files to the workstation, the source data sets on the host (MVS) system can be SCLM members, sequential data sets, or members of partitioned data sets. When receiving files from the workstation, the target data sets on the host (MVS) system can be sequential data sets or members of partitioned data sets.

When transferring SCLM members to the workstation, SCLM keeps track of which members have been sent to the workstation during a build and only sends each member to the workstation one time during the build.

For an example of the usage of this translator, see "FLMLTWST Workstation Build translator" on page 608.

Parameters

All parameters are keyword parameters and can be specified in any order. Parameters must be separated by commas. Extraneous parameters are ignored without any messages being produced.

COMMAND=PUT|GET
This parameter is required and must be specified in the options list. The valid values are:

PUT
Use this value to send files to the workstation.

GET
Use this value to retrieve files from the workstation.

This parameter will be truncated to 3 characters.

BLDINFO=@@FLMBIO
This parameter is required and must be specified in the options list with the value from @@FLMBIO.

SCLMINFO=@@FLMINF
This parameter is required and must be specified in the options list with the value from @@FLMINF.

MESSAGEDD=dd_name
This parameter is optional. If not specified, it defaults to MESSAGE. This is the ddname where messages will be written. This parameter will be truncated to 8 characters.

FILESDD=dd_name
This parameter is optional. If not specified, it defaults to FILES. This is the ddname containing the list of files to transfer. This parameter will be truncated to 8 characters. The data set allocated to this ddname must list one transfer specification per line. Each part of the transfer specification must be separated from other parts by one or more spaces. The following information must be provided with each transfer specification:

transfer format
This is a single character value that specifies the format of the file transfer.

A
Translate to ASCII

B
No translation

host data
This specifies what data set or member is the source or target of the transfer. If COMMAND=PUT is specified, this is the source of the file transfer. If COMMAND=GET is specified, this is the target of the file transfer. The format varies depending on the data being transferred.

For COMMAND=PUT, use IOTYPE=P to take advantage of the build caching function.

Data
Format

SCLM member
member.type

This format is only valid for COMMAND=PUT. The member name must be separated from the type name by a period. The member must be in the scope of the build. These members are tracked during the build and only sent to the workstation once even if FLMTXFER is called multiple times with the same member. FLMTXFER finds the member in the SCLM hierarchy and generates a fully qualified data set name with the member name for the file transfer.

Data set
'data.set.name'

This format transfers a fully qualified data set name. The data set must be sequential or specify the member name. The data set name must be surrounded by quotes. SCLM does not track the data sets sent to the workstation. If FLMTXFER is called multiple times to transfer the same data set, the data set is transferred each time.

ddname
DDNAME:member

The member name is optional, but the colon (:) must be specified. The data set allocated to the ddname must be cataloged (CATLG=Y on the FLMALLOC). FLMTXFER gets the data set name allocated to the ddname and specifies it in the file transfer command.

workstation file name
The fully qualified workstation file name including the drive and path, if they apply.

Environment

The FLMTXFER translator must have access to ISPF services. It must be called from an FLMTRNSL with CALLMETH=ISPLNK.

Return codes

In addition to the return codes listed here, messages can be written to the ddname specified by the MESSAGEDD parameter.

FILES DD example

The following examples show the content of the FILES ddname. The first shows how to send compiler inputs to a workstation and the second how to retrieve the outputs.

Given COMMAND=PUT, in the first example the following transfers take place:

1. The data set allocated to the RESPONSE ddname is sent to c:\temp\response.file in ASCII format.
2. The member PMLINES in type C is found in the SCLM hierarchy and sent to c:\temp\PMLINES.c in ASCII format.
3. The member PMLINES in type H is found in the SCLM hierarchy and sent to c:\temp\PMLINES.h in ASCII format.
4. The data set 'PROJ1.C.LIB' is sent to c:\temp\proj1.lib in BINARY format.

```
A RESPONSE: c:\temp\response.fil
A PMLINES.C c:\temp\PMLINES.c
A PMLINES.H c:\temp\PMLINES.h
B 'PROJ1.C.LIB' c:\temp\proj1.lib
```

Given COMMAND=GET, in the second example the following transfers take place:

1. The file c:\temp\PMLINES.obj is sent to the member PMLINES in the data set allocated to the OBJ ddname in BINARY format.

2. The file c:\temp\PMLINES.lst is sent to the member PMLINES in the data set allocated to the LIST ddname in ASCII format.
3. The file c:\temp\temp.msg is sent to the data set 'SCLMUSR.C.MSGS' in ASCII format.

```
B OBJ:PMLINES c:\temp\PMLINES.obj
A LIST:PMLINES c:\temp\PMLINES.lst
A 'SCLMUSR.C.MSGS' c:\temp\temp.msg
```

0

Explanation

The transfer was successful.

User response

None.

Project manager response

None.

8

Explanation

An error occurred. See the ddname specified by the MESSAGEDD parameter for more information.

User response

Refer to the generated messages. See *z/OS ISPF Messages and Codes* for an explanation of the messages.

Project manager response

None.

12

Explanation

The ddname specified by the MESSAGEDD parameter is not allocated.

User response

Contact the project manager.

Project manager response

Ensure that the ddname for messages is allocated by an FLMALLOC in the language definition or by the translator that calls FLMTXFER.

16

Explanation

The value for SCLMINFO is not valid.

User response

Contact the project manager.

Project manager response

Ensure that the SCLMINFO parameter is specified in the options list and that the parameter has a value of @@FLMINF.

SCLM parser restrictions

The SCLM parsers gather statistics on various language constructs. This section describes the constructs that the SCLM parsers cannot identify. Because a user-defined parser can be used to replace an SCLM parser, the restrictions of the SCLM-supplied parsers can be overcome, if necessary.

Unsupported constructs do not necessarily prevent members from being used in SCLM. Invalid constructs, however, prevent statistics from being gathered accurately and can result in SCLM finding too many or too few include references. Extra or missing includes can result in dependency processing errors being detected by the build and promote processors.

SCLM does not support two general types of language constructs: non-explicit references and separation of references. Both of these constructs involves include and compool references.

Non-explicit references

SCLM-supplied parsers do not support include references that are not explicitly stated on a single line of code.

The following list shows three kinds of non-explicit reference constructs.

- **Conditional References**

  Conditional references are include reference constructs that depend on information outside the scope of a single line. For the assembler language parser, for instance, all macros are considered include references whether or not they are defined within that assembly source member. All include references must exist as SCLM members, or they must exist in data set FLMSYSLB references.

- **Dynamic References**

  Dynamic references are references that involve a variable. SCLM does not support macro names passed as parameters in assembler language for include references. The following source statements for SCRIPT/VS depict a simple case of a dynamic imbed reference that SCLM does not support:

  ```
  .set count = 1
  .im member
  ```

- **Variable Delimiters**

  The delimiters you use to identify information must have fixed values. For example, SCLM does not support the following format of the `.DM` script keyword:

  ```
  .DM name /.im seg1/.im seg2/.im seg3/
  ```

  where / can be any character. This character delimits statements in the macro. SCLM does not find imbed statements entered in the `.DM` macro when the macro appears in this way.

  SCLM also does not support the following format of the `.DM` macro:

  ```
  .DM name ON
  .im seg1
  .im seg2
  .im seg3
  .DM OFF
  ```

Separation of references

Generally, you *must* separate include reference verbs of a language from referenced member names with blanks only, and they must appear on the same line. However, there are two exceptions:

- For PL/I includes and JOVIAL members, when coding !COPY or !COMPOOL statements, you can insert comments between !COPY and the include member name. (Note that only JOVIAL uses !COMPOOL.)
- The following parsers support include references on separate lines:
  - FLMLPCBL COBOL parser
  - FLMLRCBL REXX COBOL parser
  - FLMLRASM REXX Assembler parser.

SCLM does not support the following Pascal source statement because a comment separates the referenced member name.

```
%INCLUDE (* comment *) MEMNAME;
```

The include reference verb and the reference name must reside on the same line. SCLM does not support the following Pascal statement:

```
%INCLUDE
INCLMEM ;
```

Chapter 23. SCLM Variables and Metavariables

This chapter lists the SCLM variables and metavariables you can use in various stages of SCLM processing.

SCLM variable and metavariable descriptions

SCLM variables are character strings that SCLM replaces with a value. SCLM replaces these variables with eight-character values, with these exceptions:

- @@FLMBD4 variable has a value with a maximum length of 10
- @@FLMCD4 variable has a value with a maximum length of 10
- @@FLMDO*x* variable has a value with a maximum length of 44 (*x* is an integer between 0 and 9).
- @@FLMDSD variable has a value with a maximum length of 44
- @@FLMDSF variable has a value with a maximum length of 44
- @@FLMDSN variable has a value with a maximum length of 44
- @@FLMDST variable has a value with a maximum length of 44
- @@FLMICN variable has a value with a maximum length of 110
- @@FLMID4 variable has a value with a maximum length of 10
- @@FLMINC variable contains an address in decimal character format
- @@FLMINF variable contains an address in decimal character format
- @@FLMLIS variable contains an address in decimal character format
- @@FLMMD4 variable has a value with a maximum length of 10
- @@FLMPD4 variable has a value with a maximum length of 10
- @@FLMSTP variable contains an address in decimal character format
- @@FLMXCN variable has a value with a maximum length of 110
- @@FLM$C4 variable has a value with a maximum length of 10
- @@FLM$MP variable has a value with a maximum length of the build map.
- @@FLM$UD variable has a value with a maximum length of 128
- @@FLM$XD variable has a value with a maximum length of 110
- @@FLM$XN variable has a value with a maximum length of 110
- @@FLM$XU variable has a value with a maximum length of 110

In addition to these variables, SCLM has metavariables that represent SCLM internal tracking data. Table 41 on page 640 lists the SCLM metavariables and their corresponding SCLM variables. Use a metavariable in place of a combination of single SCLM variables. Variables are listed in the order in which their data values appear in the database contents utility report. There are metavariables for the fixed portion of the data and for the long (repeating) portion of the data. Table 40 on page 639 lists the SCLM metavariables and a short description of each.

You can use SCLM variables in the following places:

- On the FLMINCLS macro TYPES parameter. The following variables are supported for this parameter:
  - @@FLMCRF
  - @@FLMECR
  - @@FLMETP
  - @@FLMTYP
- With the PARM and PARMX architecture definition keywords

© Copyright IBM Corp. 1990, 2019

- On the FLMTRNSL macro OPTIONS parameter
- On the FLMALLOC macro MEMBER parameter. The following variables are supported for this parameter:
  - @@FLMMBR
  - @@FLMONM
- On the FLMCPYLB macro. The following variables are supported for FLMCPYLB statements associated with an IOTYPE I, IOTYPE A, or IOTYPE H FLMALLOC macro:
  - @@FLMALT
  - @@FLMDBQ
  - @@FLMDSN
  - @@FLMGRB
  - @@FLMGRP
  - @@FLMMBR
  - @@FLMPRJ
  - @@FLMSRF
  - @@FLMTYP
  - @@FLMUID

  Note that @@FLMDSN and @@FLMGRP reflect the group where the member being built resides. Use @@FLMGRB for the name of the group where the build is being performed.
- On the Database Contents Utility line format parameter (DBUTIL)
- On the DSNAME parameter on the FLMCNTRL and FLMALTC macros. The following variables are supported for these parameters:
  - @@FLMGRP
  - @@FLMPRJ
  - @@FLMTYP
- On the EXPACCT and EXPXREF parameters of the FLMCNTRL and FLMALTC macros. The following variables are supported for these parameters:
  - @@FLMGRP
  - @@FLMPRJ
  - @@FLMUID
- On the VERPDS parameter of the FLMCNTRL and FLMALTC macros. The following variables are supported for these parameters:
  - @@FLMDSN
  - @@FLMGRP
  - @@FLMPRJ
  - @@FLMTYP

Many of the variables can be used only for certain translator types and the SCLM utilities. Table 38 on page 633 lists the SCLM variables in alphabetical order by description and indicates for which translator types they can be used. Table 39 on page 636 lists the SCLM variables in alphabetic order by variable name.

SCLM variable and metavariable tables

The following tables illustrate SCLM variables and metavariables and their SCLM functions. Pass these variables to a translator using the OPTIONS= parameter of the FLMTRNSL macro.

Note:

1. Variables marked with a P are passed to PDS member (PDSDATA=Y on the FLMTRNSL macro) translators.
2. Variables marked with an I are passed to Ada Intermediate translators (PDSDATA=N on the FLMTRNSL macro.)
3. Variables marked with an E are passed to the External dependency translators (such as CSP/370AD.)
4. Variables marked with a √ are passed to the DBUTIL service.
5. Certain variables are passed to multiple translators depending on their function and data.

SCLM variable descriptions, variable names, and their SCLM functions

This table lists the SCLM variables in alphabetic order by their short description.

Table 38. SCLM Variable Descriptions, Names, and Their SCLM Functions

SCLM Short Description	Variable	Build	Copy	Parse	Purge	Verify	Utils
Access Key	@@FLMACK						√
Accounting Group	@@FLMGRP	P	P I E	P	P I E	P	√
Accounting Group Data Set Name	@@FLMDSN	P	P	P	P	P	√
Accounting Member	@@FLMMBR	P	P	P	P	P	√
Accounting Record Type	@@FLMATP						√
Accounting Status	@@FLMSTA						√
Accounting Type	@@FLMTYP	P	P	P	P	P	√
Alternate Project Definition	@@FLMALT	P	P I	P	P I	P	√
Assignment Statements	@@FLMASG						√
Authorization Code	@@FLMACD						√
Authorization Code Change	@@FLMACC						√
Blank Lines	@@FLMBLL						√
Buffer Size in Bytes	@@FLMSIZ	P E	E	P	E	E	
Build Group	@@FLMGRB	P					√
Build Map	@@FLM$MP	P					√
Build Map Information	@@FLMBIO	P					
Build Map Date	@@FLMMDT		P		P	P	√
Build Map Date with 4-character year	@@FLMMD4		P		P	P	√
Build Map Name	@@FLMMNM						√
Build Map Time	@@FLMMTM		P		P	P	√
Build Map Type	@@FLMMSC						√
Build Mode	@@FLMBMD					E	
Calling Function Name	@@FLMFNM		P I		P I	P	
Change Code	@@FLM$CC						√
Change Code Date	@@FLM$CD						√

Table 38. SCLM Variable Descriptions, Names, and Their SCLM Functions (continued)

SCLM Short Description	Variable	Build	Copy	Parse	Purge	Verify	Utils
Change Code Date with 4-character year	@@FLM$C4						✓
Change Code (Exposed During Parse Phase)	@@FLMCAA			P			
Change Code Time	@@FLM$CT						✓
Change Date	@@FLMCDT		P		P	P	✓
Change Date with 4-character year	@@FLMCD4		P		P	P	✓
Change Group	@@FLMCLV						✓
Change Time	@@FLMCTM		P		P	P	✓
Change User ID	@@FLMCUS						✓
Comment Lines	@@FLMCML						✓
Comment Statements	@@FLMCMS						✓
Control Statements	@@FLMCNS						✓
Creation Date	@@FLMIDT						✓
Creation Date with 4-character year	@@FLMID4						✓
Creation Time	@@FLMITM						✓
CREF Type	@@FLMCRF						
CU List	@@FLMLST		I		I		
Database Qualifier	@@FLMDBQ	P	I		I		✓
Data Set Name for OUT0	@@FLMDO0	P E					
Data Set Name for OUT1	@@FLMDO1	P E					
Data Set Name for OUT2	@@FLMDO2	P E					
Data Set Name for OUT3	@@FLMDO3	P E					
Data Set Name for OUT4	@@FLMDO4	P E					
Data Set Name for OUT5	@@FLMDO5	P E					
Data Set Name for OUT6	@@FLMDO6	P E					
Data Set Name for OUT7	@@FLMDO7	P E					
Data Set Name for OUT8	@@FLMDO8	P E					
Data Set Name for OUT9	@@FLMDO9	P E					
DDNAME Substitution List	@@FLMDDN			P			
Default Type	@@FLMSRF	P					
Dependencies Pointer	@@FLMLIS	P E	E	P	E	E	
Destination Group	@@FLMGRD		P		P	P	
Destination Group Data Set Name	@@FLMDSD		P		P	P	
Dynamic Includes Pointer	@@FLMINC	P					
Extended CREF Type	@@FLMECR						

Table 38. SCLM Variable Descriptions, Names, and Their SCLM Functions (continued)

SCLM Short Description	Variable	Build	Copy	Parse	Purge	Verify	Utils
Extended Type of Source Member	@@FLMETP						
Function Invocation Date	@@FLMFDT	P	P		P	P	
Function Invocation Time	@@FLMFTM	P	P		P	P	
Group Found	@@FLMGRF		P		P	P	
Group Found Data Set Name	@@FLMDSF		P		P	P	
Include	@@FLM$IN						✓
Include-Sets for Includes	@@FLM$IS						✓
Language	@@FLMLAN		P		P	P	✓
Language Version	@@FLMLVS						✓
Member Version	@@FLMMVR						✓
Number of Change Codes	@@FLMNCC						✓
Number of Includes	@@FLMNIN						✓
Number of Noncomment Lines	@@FLMNCL						✓
Number of Noncomment Statements	@@FLMNCS						✓
Number of User Entries	@@FLMNUE						✓
Output Member Name	@@FLMONM						
OUT0 Member Name	@@FLMOU0	P					
OUT1 Member Name	@@FLMOU1	P					
OUT2 Member Name	@@FLMOU2	P					
OUT3 Member Name	@@FLMOU3	P					
OUT4 Member Name	@@FLMOU4	P					
OUT5 Member Name	@@FLMOU5	P					
OUT6 Member Name	@@FLMOU6	P					
OUT7 Member Name	@@FLMOU7	P					
OUT8 Member Name	@@FLMOU8	P					
OUT9 Member Name	@@FLMOU9	P					
Predecessor Date	@@FLMBDT						✓
Predecessor Date with 4-character year	@@FLMBD4						✓
Predecessor Time	@@FLMBTM						✓
Project	@@FLMPRJ	P	P I	P	P I	P	✓
Prolog Lines	@@FLMPRL						✓
Promote Date	@@FLMPDT						✓
Promote Date with 4-character year	@@FLMPD4						✓
Promote Time	@@FLMPTM						✓

Table 38. SCLM Variable Descriptions, Names, and Their SCLM Functions (continued)							
SCLM Short Description	**Variable**	**Build**	**Copy**	**Parse**	**Purge**	**Verify**	**Utils**
Promote User ID	@@FLMPUS						✓
SCLM Internal Data Pointer	@@FLMINF	P E	P I E		P I E	P E	
SCLM Version	@@FLMVER						✓
Statistics Pointer	@@FLMSTP			P			
Sysprint DDNAME	@@FLMDDO		P I		P I	P	
System User ID	@@FLMUID	P	P		P	P	
Target Group	@@FLMTOG		P I E		P E	P	
Target Group Data Set Name	@@FLMDST		P		P	P	
Top CU Name	@@FLMCUN	P					
Total Lines	@@FLMTLL						✓
Total Statements	@@FLMTLS						✓
Translator Version	@@FLMTVS						✓
User Data Entry	@@FLM$UD						✓

SCLM variables and their SCLM functions

This table lists the SCLM variables in alphabetic order by variable name.

Table 39. SCLM Variables and Their SCLM Functions							
Variable	**SCLM Short Description**	**Build**	**Copy**	**Parse**	**Purge**	**Verify**	**Utils**
@@FLMACC	Authorization Code Change						✓
@@FLMACD	Authorization Code						✓
@@FLMACK	Access Key						✓
@@FLMALT	Alternate Project Definition	P	P I	P	P I	P	✓
@@FLMASG	Assignment Statements						✓
@@FLMATP	Accounting Record Type						✓
@@FLMBDT	Predecessor Date						✓
@@FLMBD4	Predecessor Date with 4-character year						✓
@@FLMBIO	Build Map Information	P					
@@FLMBLL	Blank Lines						✓
@@FLMBMD	Build Mode					E	
@@FLMBTM	Predecessor Time						✓
@@FLMCAA	Change Code (Exposed During Parse Phase)			P			
@@FLMCDT	Change Date		P		P	P	✓
@@FLMCD4	Change Date with 4-character year		P		P	P	✓
@@FLMCLV	Change Group						✓

Table 39. SCLM Variables and Their SCLM Functions (continued)

Variable	SCLM Short Description	Build	Copy	Parse	Purge	Verify	Utils
@@FLMCML	Comment Lines						✓
@@FLMCMS	Comment Statements						✓
@@FLMCNS	Control Statements						✓
@@FLMCRF	CREF Type						
@@FLMCTM	Change Time		P		P	P	✓
@@FLMCUN	Top CU Name	P					
@@FLMCUS	Change User ID						✓
@@FLMDBQ	Database Qualifier	P	I		I		✓
@@FLMDDN	DDNAME Substitution List			P			
@@FLMDDO	Sysprint DDNAME		P I		P I	P	
@@FLMDO0	Data Set Name for OUT0	P E					
@@FLMDO1	Data Set Name for OUT1	P E					
@@FLMDO2	Data Set Name for OUT2	P E					
@@FLMDO3	Data Set Name for OUT3	P E					
@@FLMDO4	Data Set Name for OUT4	P E					
@@FLMDO5	Data Set Name for OUT5	P E					
@@FLMDO6	Data Set Name for OUT6	P E					
@@FLMDO7	Data Set Name for OUT7	P E					
@@FLMDO8	Data Set Name for OUT8	P E					
@@FLMDO9	Data Set Name for OUT9	P E					
@@FLMDSD	Destination Group Data Set Name		P		P	P	
@@FLMDSF	Group Found Data Set Name		P		P	P	
@@FLMDSN	Accounting Group Data Set Name	P	P	P	P	P	✓
@@FLMDST	Target Group Data Set Name		P		P	P	
@@FLMECR	Extended CREF Type						
@@FLMETP	Extended Type of Source Member						
@@FLMFDT	Function Invocation Date	P	P		P	P	
@@FLMFNM	Calling Function Name		P I		P I	P	
@@FLMFTM	Function Invocation Time	P	P		P	P	
@@FLMGRB	Build Group	P					
@@FLMGRD	Destination Group		P		P	P	
@@FLMGRF	Group Found		P		P	P	
@@FLMGRP	Accounting Group	P	P I E	P	P I E	P	✓
@@FLMIDT	Creation Date						✓

Table 39. SCLM Variables and Their SCLM Functions (continued)

Variable	SCLM Short Description	Build	Copy	Parse	Purge	Verify	Utils
@@FLMID4	Creation Date with 4-character year						✓
@@FLMINC	Dynamic Includes Pointer	P					
@@FLMINF	SCLM Internal Data Pointer	P E	P I E		P I E	P E	
@@FLMITM	Creation Time						✓
@@FLMLAN	Language		P		P	P	✓
@@FLMLIS	Dependencies Pointer	P E	E	P	E	E	
@@FLMLST	CU List		I		I		
@@FLMLVS	Language Version						✓
@@FLMMBR	Accounting Member	P	P	P	P	P	✓
@@FLMMDT	Build Map Date		P		P	P	✓
@@FLMMD4	Build Map Date with 4-character year		P		P	P	✓
@@FLMMNM	Build Map Name						✓
@@FLMMSC	Build Map Type						✓
@@FLMMTM	Build Map Time		P		P	P	✓
@@FLMMVR	Member Version						✓
@@FLMNCC	Number of Change Codes						✓
@@FLMNCL	Number of Noncomment Lines						✓
@@FLMNCS	Number of Noncomment Statements						✓
@@FLMNIN	Number of Includes						✓
@@FLMNUE	Number of User Entries						✓
@@FLMONM	Output Member Name						
@@FLMOU0	OUT0 Member Name	P					
@@FLMOU1	OUT1 Member Name	P					
@@FLMOU2	OUT2 Member Name	P					
@@FLMOU3	OUT3 Member Name	P					
@@FLMOU4	OUT4 Member Name	P					
@@FLMOU5	OUT5 Member Name	P					
@@FLMOU6	OUT6 Member Name	P					
@@FLMOU7	OUT7 Member Name	P					
@@FLMOU8	OUT8 Member Name	P					
@@FLMOU9	OUT9 Member Name	P					
@@FLMPDT	Promote Date						✓
@@FLMPD4	Promote Date with 4-character year						✓
@@FLMPRJ	Project	P	P I	P	P I	P	✓

Table 39. SCLM Variables and Their SCLM Functions (continued)							
Variable	**SCLM Short Description**	**Build**	**Copy**	**Parse**	**Purge**	**Verify**	**Utils**
@@FLMPRL	Prolog Lines						√
@@FLMPTM	Promote Time						√
@@FLMPUS	Promote User ID						√
@@FLMSIZ	Buffer Size in Bytes	P E	E	P	E	E	
@@FLMSRF	Default Type	P					
@@FLMSTA	Accounting Status						√
@@FLMSTP	Statistics Pointer			P			
@@FLMTLL	Total Lines						√
@@FLMTLS	Total Statements						√
@@FLMTOG	Target Group		P I E		P E	P	
@@FLMTVS	Translator Version						√
@@FLMTYP	Accounting Type	P	P	P	P	P	√
@@FLMUID	System User ID	P	P		P	P	
@@FLMVER	SCLM Version						√
@@FLM$CC	Change Code						√
@@FLM$CD	Change Code Date						√
@@FLM$C4	Change Code Date with 4-character year						√
@@FLM$CT	Change Code Time						√
@@FLM$IN	Include						√
@@FLM$IS	Include-Sets for Includes						√
@@FLM$MP	Build Map						√
@@FLM$UD	User Data Entry						√

SCLM metavariable descriptions, metavariable names, and their SCLM functions

This table lists the SCLM metavariables in alphabetic order by description. Metavariables are only used with the DBUTIL service.

Table 40. SCLM Metavariable Descriptions, Names, and Their SCLM Functions							
SCLM Short Description	**Metavariable**	**Build**	**Copy**	**Parse**	**Purge**	**Verify**	**Utils**
Account Report Fixed	@@FLM#AF						√
Account Report Long	@@FLM#AL						√

SCLM metavariable contents

Table 41 on page 640 lists the SCLM metavariables and their corresponding SCLM variables. A metavariable represents a list of predefined SCLM variables. Specifying a metavariable is equivalent to specifying its corresponding list of SCLM variables in the order listed in Table 41 on page 640.

Table 41. SCLM Metavariables and Their Corresponding Variables	
Metavariable	**Variable**
@@FLM#AF	@@FLMPRJ @@FLMACK @@FLMBLL @@FLMALT @@FLMIDT @@FLMPRL @@FLMGRP @@FLMITM @@FLMTLS @@FLMTYP @@FLMMDT @@FLMCMS @@FLMMBR @@FLMMTM @@FLMCNS @@FLMVER @@FLMBDT @@FLMASG @@FLMSTA @@FLMBTM @@FLMNCS @@FLMCDT @@FLMPDT @@FLMNUE @@FLMCTM @@FLMPTM @@FLMNIN @@FLMCLV @@FLMPUS @@FLMNCC @@FLMCUS @@FLMDBQ @@FLMNCU @@FLMMVR @@FLMTVS @@FLM$IN @@FLMLAN @@FLMMNM @@FLM$IS @@FLMATP @@FLMMSC @@FLM$CC @@FLMLVS @@FLMTLL @@FLM$CD @@FLMACD @@FLMCML @@FLM$CT @@FLMACC @@FLMNCL
@@FLM#AL	@@FLM$XT @@FLM$XN @@FLM$UD

Description of group variables

This section further explains the use of group variables. Table 42 on page 640 lists each group variable and associated group data set name variable. This shows the relationship between SCLM groups and the data sets defined in the project definition for each group.

Table 43 on page 641 is an example that lists the values of each group variable during the phases of a promote. After Table 42 on page 640 is an overall description of the four group variables and why each is needed. Each group variable has a corresponding data set name variable due to the flexible data set name capability.

Table 42. SCLM Group Variable List

Group Variable	Group Data Set Name Variable	Description
@@FLMGRP	@@FLMDSN	Accounting Group and Accounting Group Data Set Name
@@FLMGRF	@@FLMDSF	Group Found and Group Found Data Set Name
@@FLMTOG	@@FLMDST	Target Group and Target Group Data Set Name
@@FLMGRD	@@FLMDSD	Destination Group and Destination Group Data Set Name

The following hierarchy will be used in the description:

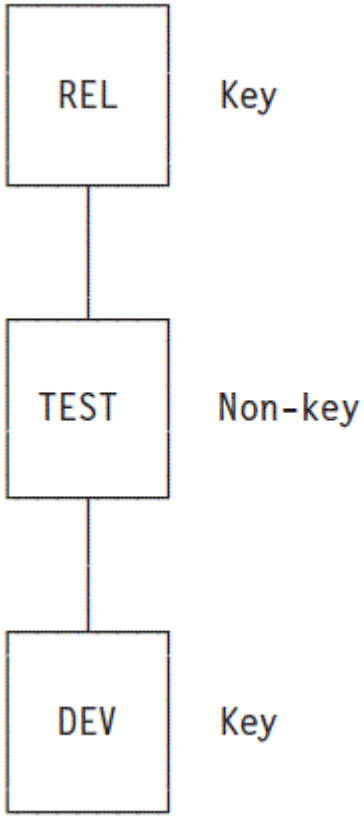


Figure 211. Hierarchy Example for Group Description

Given the preceding hierarchy, Table 43 on page 641 describes what each group variable would contain during each translator phase of a PROMOTE from TEST to REL.

Table 43. SCLM Group Variable Description

Translator	Accounting Group	Group Found	Target Group	Destination Group
Verify	TEST	TEST	REL	REL
Copy	TEST	TEST	REL	REL
Purge key	DEV	TEST	DEV	REL
Purge non-key	TEST	TEST	TEST	REL

The purge translator is invoked twice during this promote due to the promotion from a non-key group to a key group.

Appendix A. Accessibility

Accessible publications for this product are offered through IBM Knowledge Center (www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

If you experience difficulty with the accessibility of any z/OS information, send a detailed email message to mhvrcfs@us.ibm.com.

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS ISPF User's Guide Vol I*

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Knowledge Center with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or `3.1` or `3.1.1`. To hear these numbers correctly, make sure that the screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number `3.1`) are mutually exclusive alternatives. If you hear the lines `3.1 USERID` and `3.1 SYSTEMID`, your syntax can include either `USERID` or `SYSTEMID`, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

© Copyright IBM Corp. 1990, 2019

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you must refer to separate syntax fragment OP1.

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element
The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that the syntax elements NOTIFY and UPDATE are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! indicates a default syntax element
The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In the example, if you include the FILE keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, the default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

*** indicates an optional syntax element that is repeatable**
The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3* , 3 HOST, 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.

3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line `6.1+ data area`, you must include at least one data area. If you hear the lines `2+, 2 HOST,` and `2 STATE`, you know that you must include `HOST, STATE,` or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

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 grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for the Knowledge Centers. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Site Counsel
2455 South Road

© Copyright IBM Corp. 1990, 2019

Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming Interface Information

This publication primarily documents information that is NOT intended to be used as Programming Interfaces of ISPF.

This publication also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of ISPF. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

```
+---------------------Programming Interface information----------------------+

+------------------End of Programming Interface information------------------+
```

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and Trademark information (www.ibm.com/legal/copytrade.shtml).

Glossary of SCLM Terms

A

access key
An identifier used to restrict access to a member.

accounting information
Accounting information is stored in the SCLM VSAM accounting data sets and consists of accounting and build map records.

accounting record
An SCLM control data record containing statistical, historical, and dependency information for a member under SCLM control.

action bar
The area at the top of an ISPF panel that contains choices that give you access to actions available on that panel. When you select an action bar choice, ISPF displays an action bar *pull-down menu*.

alternate project definition
A project definition that provides a version of the project environment which differs from the default project definition.

application
Software that performs a function for an end user.

API
Application Programming Interface

APT
Application Programming and Test

architecture
The organization of software components to form integrated applications.

architecture definition
A means of organizing components of an application into conceptual units. It is SCLM's method of defining an application's configuration. It describes how the components of an application fit together and is used to drive both the build and promote functions. Architecture definitions are used to group components into applications, sub-applications, and load modules.

architecture member
Defines an individual software component, which may be a collection of other architecture members, by specifying its relationship to other software components of an application.

audit information
Information associated with a member which describes when a member was modified, how it was modified, and who modified it. This information is stored in the SCLM VSAM audit data sets.

audit trail
See *audit information*.

authorization code
An identifier used by SCLM to control authority to update and promote members within a hierarchy. These codes can be used to allow concurrent development without the risk of module collisions (overlaid changes).

authorization group
An identifier associated with a set of authorization codes.

B

build
The process of transforming inputs into outputs through the invocation of translators specified in the language definition. Compilers, preprocessors, and linkage editors are examples of translators that might be invoked at build time.

© Copyright IBM Corp. 1990, 2019

build map
Internal data record containing a complete analysis of the database at the time of the build; it includes the names of all referenced members and the last change date and version number of each member.

C

change code
An 8-character identifier used to indicate the reason for an update or modification to a member controlled by SCLM.

code
Program(s) written in a language that is subject to a given translation process.

compilable member
A member recognized by the compiler or translator as an independent unit or a controlling unit for the language.

component
See *software component*.

concurrent updates
Concurrent updates occur when two programmers update the same member at the same time. This is supported through the use of authorization codes and the Edit Compare tool or alternate project definitions.

configuration management
See *software configuration management*.

configuration management plan
See *software configuration management plan*

control data
Information that SCLM stores about each member under its control. The control data is stored in the accounting and audit VSAM data sets defined for a project.

copylib
A library containing include referenced source code.

cross-reference record
Internal data record containing Ada compilation unit/member relationship information.

D

data base
SCLM-controlled VSAM data sets for a project.

database administrator
See *project administrator*.

ddname substitution list
A string of ddnames allocated for the translator. The ddname substitution list is usually documented in the Programmer's Guide for compilers and linkage editors.

default architecture definition
Architecture definition that is generated by SCLM when one is not specified as input to a build. This is done when a source member is built directly.

default project definition
The main project definition used by an SCLM project.

dependency
Dependency describes a relationship between a source member and the members it includes. A source member has a dependency on a member which it includes.

dependency information
Information on dependencies is stored in the SCLM accounting record.

development group
All groups in the lowest level of the hierarchy are known as "development groups". These groups represent end-nodes with no other lower groups promoting into them.

development layer
Layer of an SCLM hierarchy consisting of development groups.

development life cycle
The process followed to create an application. The process starts at the program requirements gathering phase, moves to the design phase, the development phase, and continues to the release of the final product.

downward dependency
A dependency indicating a compilation unit which must be compiled after the current compilation unit is compiled.

draw down
During edit, SCLM copies the member from its first occurrence in a key group in the library concatenation into a development group and locks it.

dynamic include
An include for a source member that cannot be resolved until after the translator invocation.

dynamic reference
A reference that involves a variable.

E

editable/non-editable
Source members (created by an edit session) are editable; members produced by a processor during a build are non-editable.

ellipsis
Three dots that follow a pull-down choice. When you select a choice that contains an ellipsis, ISPF displays a *pop-up* window.

F

function key
In previous releases of ISPF, a programmed function (PF) key. *This is a change in terminology only*.

G

group
A set of project data sets with the same middle-level qualifier in the SCLM logical naming convention.

H

hierarchical view
A path of groups (concatenation) through the hierarchy. The path may start at any group in the hierarchy and follows the promote path to the topmost group in the hierarchy.

hierarchy
The organization of groups in a ranked order, where each group is subordinate to the one above it.

I

include
A member that is required to complete a compilation of the member that references it.

include-set
An include-set is used to associate an included member name with the type or types in the project which are searched to find a member with that name.

integrate
To merge two or more software components of an application into a single software application.

K

key group
Data is copied into this group and then purged from the previous group, effectively "moving" the data. Non-key groups are used when a simple copy is desired.

L

language definition
Specifies the set of translators to be executed for SCLM functions PARSE, VERIFY, BUILD, COPY, and PURGE. A language definition is composed of one FLMLANGL macro followed by an FLMTRNSL macro for each translator to be executed for members of SCLM libraries whose language attribute matches the value of the LANG keyword in the FLMLANGL macro.

layer
A given tier of the hierarchy, made up of groups of equivalent rank.

level
See *layer*.

library (MVS)
A partitioned data set.

lock
When a user locks a member, only that user can change it. All other users are unable to change that member until the member is promoted or unlocked. When you lock a member, you specify an authorization code. If two users need to change a part, they can use different authorization codes.

lock service
Restricts (locks) a member to a development group.

M

maximum promotable group
The topmost group to which a member can be promoted.

member
The discrete element of an SCLM database, representing a single data type of a software component.

metavariable
A variable that includes many other SCLM variables.

migrate
Registering software components in SCLM: this includes identifying the component language, and possibly the change code and authorization code.

migration
The process of introducing members into SCLM control. Migration locks the member, parses it according to the requested language, and stores the information in the accounting data base. You can user the migration utility to enter a large number of members into a project's data base, such as during conversion to SCLM.

Modal pop-up window
A type of window that requires you to interact with the panel in the pop-up before continuing. This includes canceling the window or supplying information requested.

Modeless pop-up window
A type of window that allows you to interact with the dialog that produced the pop-up before interacting with the pop-up itself.

N

nested dependencies
Nested dependencies occur when a source member includes another member, which in turn includes another member. SCLM tracks nested dependencies, so that when a member changes, any member that includes it is rebuilt, no matter how many levels of nesting there are.

non-key group
A group that data is copied into (as opposed to moved into) during promotion.

P

parser
A program that reads an editable member to determine dependency and statistical information about the member. This information is stored in the SCLM accounting data base.

predecessor date/time
The last modified date/time stamp taken from the previous version of the current member.

point-and-shoot text
Text on a screen that is cursor sensitive.

pop-up window
A bordered temporary window that displays over another panel.

predecessor verification
The process of verifying that the previous version of a member has not changed.

predecessors
Previous versions of a member existing at a higher level within the same hierarchical view.

primary commands
Editing commands that are entered on the Command line.

primary group
A key or non-key group with two or more groups promoting into it that must be allocated when a hierarchy is to be accessed.

private library
A partitioned data set or partitioned data set extended belonging to a group in the development layer of the hierarchy.

project
A collection of libraries representing an integrated SCLM data base, under a single high-level qualifier.

project administrator
The person who maintains an SCLM project.

project definition
Defines the SCLM library structure, project control information, and language definitions. A project definition is a load module used by SCLM at run time. The source code for a project definition is composed of macros.

project definition data
Project definitions and language definitions which are used to create and control an SCLM project.

project environment
Information which makes up an SCLM project. There are three types of information:

- Project Definition Data
- User Applications Data
- Control Data

project identifier
The name assigned to the project definition.

Project Partitioned Data Sets
MVS Partitioned Data Sets where user application data is stored.

promote
The process of moving an application or its components from one level in the project hierarchy to the next. Promotion out of a development group removes the lock on editable members that were successfully promoted.

promote path
The link between two groups along which data moves from one subordinate group to the next group in the hierarchy.

pull-down menu
A list of numbered choices extending from the selection you made on the action bar. The action bar selection will be highlighted. You can select an action either by typing in its number and pressing Enter or by selecting the action with your cursor. ISPF displays the requested panel. If your choice contains an *ellipsis* (...), ISPF displays a *pop-up window*. When you exit this panel or pop-up, ISPF closes the pull-down and returns you to the panel from which you made the initial action bar selection.

push button
A rectangle with text inside. Push buttons are used in windows for actions that occur immediately when the push button is selected (available only when you are running in GUI mode).

S

SCLM_id
Identifier used to communicate information between the SCLM services. There is a unique SCLM_id generated for each invocation of the INIT service.

scope
The set of members (including architecture definitions) that will be processed (for example verified, copied, compiled, or purged) by build or promote.

service
An SCLM function available via a command or programming interface.

service parameter list
The options supplied when invoking an SCLM service.

software component
Any input or output member associated with an application, which together make up all or a member of the application.

software configuration management
The method of controlling and integrating software components to produce high quality applications. Provides a common point of integration for all planning and implementation activities for a project.

software configuration management plan
A formalized procedure for software configuration management.

subapplications
Separate parts of an application being developed within a project. Once the project is completed, the parts are integrated to form the final product.

syslib
A library containing source code not under SCLM control. No dependency information is maintained for members in a syslib.

T

text
Data present in its natural language form (not translatable).

traceability
Capability to access and maintain records of information about a software component, including when the component was last changed and why.

translator
A load module, CLIST, or REXX program that receives control from SCLM for execution. The name of the translator is specified as the value of the COMPILE keyword for the FLMTRNSL macro. Examples of translators are compilers, assemblers, linkage editors, text processors, DB2 preprocessors, CICS preprocessors, utilities, and customer tools.

type
The third qualifier of the SCLM naming convention for project partitioned data sets. Typically identifies the kind of data maintained for a project hierarchy. Examples of types are SOURCE, OBJECT and LOAD.

U

unlock
To make a member (formerly locked out) available for updating (usually associated with promote).

unlock service
Removes the restriction (unlocks) on a member to a development group.

upward dependency
A dependency indicating a compilation unit that must be compiled before the current compilation unit is compiled.

V

Version
A copy of a member as it existed at a previous point in time.

Versioning
A function that enables you to retrieve a version of a member. Useful for "backing out" changes.

Index

Special Characters

@@FLM#AF 639
@@FLM#AL 639
@@FLM$C4 634, 639
@@FLM$CC 633, 639
@@FLM$CD 633, 639
@@FLM$CT 634, 639
@@FLM$IN 635, 639
@@FLM$IS 635, 639
@@FLM$MP 633, 639
@@FLM$UD 636, 639
@@FLMACC 633, 636
@@FLMACD 633, 636
@@FLMACK 633, 636
@@FLMALT 633, 636
@@FLMASG 633, 636
@@FLMATP 633, 636
@@FLMBD4 635, 636
@@FLMBDT 635, 636
@@FLMBIO 633, 636
@@FLMBLL 633, 636
@@FLMBMD 633, 636
@@FLMBTM 635, 636
@@FLMCAA 634, 636
@@FLMCD4 634, 636
@@FLMCDT 634, 636
@@FLMCLV 634, 636
@@FLMCML 634, 637
@@FLMCMS 634, 637
@@FLMCNS 634, 637
@@FLMCRF 634, 637
@@FLMCTM 634, 637
@@FLMCUN 636, 637
@@FLMCUS 634, 637
@@FLMDBQ 634, 637
@@FLMDDN 634, 637
@@FLMDDO 636, 637
@@FLMDO0 634, 637
@@FLMDO1 634, 637
@@FLMDO2 634, 637
@@FLMDO3 634, 637
@@FLMDO4 634, 637
@@FLMDO5 634, 637
@@FLMDO6 634, 637
@@FLMDO7 634, 637
@@FLMDO8 634, 637
@@FLMDO9 634, 637
@@FLMDSD 634, 637
@@FLMDSF 635, 637
@@FLMDSN 633, 637
@@FLMDST 636, 637
@@FLMECR 634, 637
@@FLMETP 635, 637
@@FLMFDT 635, 637
@@FLMFNM 633, 637
@@FLMFTM 635, 637
@@FLMGRB 633, 637
@@FLMGRD 634, 637
@@FLMGRF 635, 637
@@FLMGRP 633, 637
@@FLMGRP variable 28
@@FLMID4 634, 638
@@FLMIDT 634, 637
@@FLMINC 100, 634, 638
@@FLMINF 636, 638
@@FLMITM 634, 638
@@FLMLAN 635, 638
@@FLMLIS 634, 638
@@FLMLST 634, 638
@@FLMLVS 635, 638
@@FLMMBR 633, 638
@@FLMMD4 633, 638
@@FLMMDT 633, 638
@@FLMMNM 633, 638
@@FLMMSC 633, 638
@@FLMMTM 633, 638
@@FLMMVR 635, 638
@@FLMNCC 635, 638
@@FLMNCL 635, 638
@@FLMNCS 635, 638
@@FLMNIN 635, 638
@@FLMNUE 635, 638
@@FLMONM 635, 638
@@FLMOU0 635, 638
@@FLMOU1 635, 638
@@FLMOU2 635, 638
@@FLMOU3 635, 638
@@FLMOU4 635, 638
@@FLMOU5 635, 638
@@FLMOU6 635, 638
@@FLMOU7 635, 638
@@FLMOU8 635, 638
@@FLMOU9 635, 638
@@FLMPD4 635, 638
@@FLMPDT 635, 638
@@FLMPRJ 635, 638
@@FLMPRL 635, 639
@@FLMPTM 635, 639
@@FLMPUS 636, 639
@@FLMSIZ 633, 639
@@FLMSRF 634, 639
@@FLMSTA 633, 639
@@FLMSTP 636, 639
@@FLMTLL 636, 639
@@FLMTLS 636, 639
@@FLMTOG 636, 639
@@FLMTVS 636, 639
@@FLMTYP 633, 639
@@FLMUID 636, 639
@@FLMVER 636, 639
$acct_info 368
$list_info
    accounting records 371

$msg_array 368
$stats_info 370

A

access key
  definition of 165
  incorrect 401
  locking a member 430
  purpose for 431
  resetting 462
  variable 636
  verification 431
accessibility
  contact IBM 643
  features 643
accounting data set
  creating 18
  space computation 20
  specifying 27
  synchronizing 70
accounting group
  definition of 397
  variable 637
accounting information
  change codes 168
  field descriptions 164, 181
  include reference 169
  selection criteria 181
accounting information, field format 368
accounting member variable 633, 638
accounting record type
  definition of 182
accounting record type variable 633, 636
accounting records
  DBACCT service 394
  DELETE service 400
  deleting 157
  DELGROUP service 402
  field descriptions 164
  historical information 165
  metavariables 639
  panel 164
  retrieve 394
  statistical information 166
  variables 632
accounting statistics report 187
accounting status
  definition of 164
accounting status variable 633, 639
accounting type variable 633, 639
ACCT control option 27
ACCT2 control option 27
ACCTINFO Service 380
action bar
  Migration Utility - Entry panel choices 178
  View - Entry panel choices 147
action reason values 206
ALIAS keyword, format 278
allocating
  number of data sets 13
  project data sets 12
  SCLM data sets 13, 17
allocating SCLM data sets pointer parameters 367
allocating SCLM data sets, Output Disposition 254
alternate project definition
  creating 75
  defining 24
alternate project definition, selecting 146
application
  controlling 274
  defining 274
  sample 283
application components 274
architecture
  scope 182
architecture definition
  compilation control 272
  converting JCL decks 112
  creating 75, 277
  fields 182
  generic 275
  high-level 274
  kinds of 271
  language 277
  link-edit control 273
  overview 271
  sample 283
  statement
    format 277
    optional LIST 273
    optional LMAP 274
    uses 278
  synchronization with 287
  understanding 258
  use of 271
architecture member 271
architecture report
  architecture information 188
  cross-reference information 188
  utility 188
architecture type 8
arrays
  accounting information 368
  list information 371
  message 368
  statistical information 370
assemble project definition 38
assignment statement
  in accounting records 168
assignment statement variable 633, 636
assistive technologies 643
audit and version selection 205
audit and version utility
  hierarchy view 205
  member record 208
audit control data sets
  allocation of 20
  protecting 23
  specifying 27
audit control data sets, specifying 17
audit information, storing in a VSAM data set 203
audit version delete notify (ADVNTF) 61
audit version delete user exit routine
  parameters 61
  requirements 61
  specification 61
audit version delete user exit routine, specifying 61

audit version delete verify (AVDVFY) 61
AUTHCODE Service 384
authorization code
  definition of 8
  for concurrent development and maintenance 11
  for controlling
    member updates 9
    SCLM promotions 9
    test versions of members 9
  update panel 173
  variable 633, 636
  verification
    LOCK service 431, 433
    MIGRATE service 436
    SAVE service 453
authorization code change
  definition of 165
authorization code change variable 633, 636
authorization code usage 9
authorization group, defining 25
automatic ordering
  compile 272
AVDNTF 54
AVDVFY 54

B

back-bind 305
BACKEDUP status 217, 223
backup of project environment 70
batch processing 252
bind control file 305
BIND DB2 application 302
bind options
  DB2CLIST member 303
BKMBRLVL parameter 221
blank lines variable 633, 636
BLDEXT1 54
BLDEXT1, Build Notify user exit 58
BLDINIT 54
BLDINIT, Build Initial user exit 58
BLDNTF 54
BLDNTF, Build Notify user exit 58
browse mode 149
buffer size
  definition of 547
  variable 633, 639
Build
  by change code 275
build and promote user exit routine, specifying 57
build function
  architecture member 247
  build 240
  build map
    accounting records 166
    contents 172
    date verification 247
    deleting 160
    record 170
  build map variables 633, 638
  example 243
  function summary 237
  generating a report 239
  information 243
build function *(continued)*
  modes 239
  panel 238
  parameters 389
  report 240
  scopes 239
build map information variable 633
build map table 421
build map variable 633, 639
BUILD service 388
build support
  workstation support 307
build user exit routine specification 391
Build, using 264
build/promote user exit routine
  data set 60
  example 64
  parameters 58
  requirements 58
  specification 57

C

call format
  C 364
  COBOL 364
  FORTRAN 363
  Pascal 364
  PL/I 364
calling function name variable 633, 637
CC architecture definitions, writing 111
CCEXITS service 392
CCODE
  in architecture statements 278
CCSAVE 53
CCSAVE, save change code exit 55
CCVFH, verify change code 55
CCVFY, verify change code exit 53
change code
  accounting records 168
  array record 371
  deleting 168
  during parse phase 634, 636
  input 155
  list of 168
  report 186
  variables 633, 639
change code verification routine
  creating 55
  example 56
  specifying 55
change code verification routine, VERCC 55
change promote processing (NOPROM) 176
character parameters 367
cleanup report 188
cleanup, project 269
CLIST
  command procedure 361
  variable 360
CMD statement
  format 279
  restriction 279
  use of 273
code

code *(continued)*
copying 80
parsing 80
translating 80
code, authorization
definition of 8
for concurrent development and maintenance 11
for controlling
member updates 9
SCLM promotions 9
test versions of members 9
update panel 173
variable 633, 636
verification
LOCK service 431, 433
MIGRATE service 436
SAVE service 453
code, change
accounting records 168
deleting 168
input 155
list of 168
report 186
variables 633, 639
code, return
BUILD service 391
CCEXITS service 393
DBACCT service 395
DBUTIL service 399
DELETE service 402
DELGROUP service 406
DSALLOC service 409
EDIT service 412
END service 413, 416
EXPORT service 418
FREE service 419
general categories 377
GETBLDMP service 423
GOODRC 560
IMPORT service 428
INIT service 430
LOCK service 433
MIGRATE service 437
PARSE service 443
PROMOTE service 447
RPTARCH service 450
SAVE service 454
SCLMINFO service 457
START service 458
STORE service 461
UNLOCK service 464
VERDEL service 466
VERHIST service 468
VERINFO service 471
VERRECOV service 474
command
data set conventions 360
DEFINE 156
EXECUTE 179
FLMCMD 359
interactive processing 361
invocation format 359
line 146
primary 146
command *(continued)*
QUIT 362
service invocation 359
SETSSI 274
SUBMIT 179
command macros
Save 152
SCOMPARE 153
SCOPY 153
SCREATE 154
SMOVE 154
SPROF 155
SREPLACE 156
command processing,interactive 361
command shell, SCLM 252
commands
reading syntax diagrams xxiv
comment lines 167
comment lines variable 634, 637
comment statements 167
comment statements variable 634, 637
Compare Type 210, 211
compilation control architecture member
requirement 272
use of 272
compile errors 77
compiler
options override 28, 272
used by SCLM 32
compiler processed components 272
components
application and subapplication 274
compiler processed 272
link-edit processed 273
processing conditionally saved 94
concurrent development and maintenance 10
conditional mode
build 239
promote 247
conditionally saved components 94
configuring the input list translators 101
considerations, performance 360
contact
z/OS 643
contention, data 251
control data sets
allocating 18
protecting 23
specifying to project definition 26
control options
ACCT 27
ACCT2 27
change code verification routine specification 55
DASDUNIT 29, 527
DSNAME 28
EXPACCT 27
MAXLINE 28
MAXVIO 29
OPTOVER 28
user exits 57, 61, 62
VERPDS 28
VERS 28
VERS2 28
VIOUNIT 29

control statements
in accounting records 168
validation 277
control statements variable 634, 637
controlling member
test versions 9
updates 9
conversion to SCLM
architecture definitions 75
initialization of non-key groups 75
introduction of fixes 77
prerequisites 75
project definitions 75
registration of members 76
converting JCL decks 112
converting JCL to SCLM language definitions 118
COPY statement
format 279
use of 279
creating object modules 272
CREF statement
use of 248
cross project support 69
cross reference variables 633, 636
cross-project support 69
cross-reference
report 188
CU list variable 634, 638

D

DASDUNIT control option 29, 527
data contention 251
data set
accounting 27
allocation 17
attributes 17
concatenations 254
exit output 60, 63
flexible naming 12
naming convention 12
overflow 251
overlay 254
secondary accounting 27
synchronizing 70
data set naming conventions
ALTC parameter 541
FLMGROUP 540
using FLMALTC 510
data set prefix, unit of work 227
data set protection 367
database
accounting records 164
backup 70
historical information 165
organization 140
recovery 70
statistical information 166
database contents utility
field names 180
pattern examples 367
report 183
selection criteria
accounting information 181
database contents utility *(continued)*
selection criteria *(continued)*
architecture definition 182
tailored data set
definition of 183
example 185
options 184
report 185
using 266
database qualifier
format 370
variable 634, 637
date_check parameter 282
DB2 language definitions 298
DB2 support
bind control file 305
binding on different LPARs 305
CLIST member, creating 302
DBRM name 303
defining language definitions 298
generating a project environment 296
generic architecture definition 301
high-level architecture definition 301
LEC architecture definition 300
REBIND option 305
restrictions 296
DB2CLIST member
and REXX 295
bind options 303
creating 302
generic example 303
processing flow 300
DBACCT service 394
DBRM name 303
DBUTIL service 395
DDNAME parameters 366
ddname substitution list
defining new language to SCLM 102
use of 35, 495
variable 634, 637
ddnames
service command panels 365
decoding, member 333
default project definition 4
default type
use of 283
default type variable 634, 639
default type, size 102
DEFINE command 156
defining
authorization groups 25
generic architecture members 275
language definition 79
project 3
subapplication 274
translator definition 79
defining a new language
defining a preprocessor 113
determining what information goes where 101
how to write CC architecture definitions 111
step-by-step 103
defining an SCLM project, prerequisites 39
defining software component 509
definition, architecture

definition, architecture *(continued)*
compilation control 272
converting JCL decks 112
creating 75, 277
fields 182
generic 275
high-level 274
kinds of 271
language 277
link-edit control 273
overview 271
sample 283
statement
format 277
optional LIST 273
optional LMAP 274
uses 278
synchronization with 287
understanding 258
use of 271
delete from group utility
delete mode 215
example report 215
Delete Notify exit, DELNTF 62
DELETE service 400
delete user exit routine
data set 63
parameters 63
requirements 62
specification 62
Delete Verify exit, DELVFY 62
deleting
accounting records 157
build map records 157
change codes 168
cross-reference records 160
data sets 253
from a key group 160
intermediate records 157
members 157
user data entry records 170
DELGROUP service 402
DELINIT 54
DELINIT, initial delete exit 62
DELNTF 54
DELNTF, Delete Notify exit 62
DELVFY 54
DELVFY, Delete Verify exit 62
dependencies pointer variable 634, 638
dependency
information 177
dependency errors 77
dependency processing
include 293
development activity examples 257
development and maintenance, concurrent 10
development cycle example 261
development scenario 257
dialog interface
Build (option 4) 237
Edit (option 2) 149
main menu 145
Promote (option 5) 245
Utilities (option 3) 156
dialog interface *(continued)*
View (option 1) 146
virtual region size 143
dialog interface, modifying delete from group 71
different versions of SCLM, running 3
directory blocks 17
DOWN command 163
drawdown feature 142, 149
drawing down a member 269
DSALLOC service 407
dynamic includes
definition of 99
pointer 100
tracking 99
using 99
dynamic includes variable 634, 638

E

Easy Cmds option 252, 365
edit
change code support 155
commands
Save 152
SCOMPARE 153
SCOPY 153
SCREATE 154
SMOVE 154
SPROF 155
SREPLACE 156
drawdown feature 149
function 149
panel 150
process 149
records and field names 150
Edit Entry panel 150
Edit Profile Panel 155
EDIT service 410
editable members 76
Editable types, and package backout 217
editing a member 265
editions, comparing SCLM and ISPF 152
editor, using 262
elements
FLMNPROM macro 549
encoding, member 333
END service 413
ENDEC service 414
ensuring synchronization of hierarchy 287
errors
compile 77
dependency 77
hierarchy 77
establish authorization codes 8
EXECUTE command 179
exit routine
audit version delete 61
build 57
delete 62
example 64
output data sets 60, 63
promote 57
specification 57, 61, 62
EXPACCT control option 27, 525

Export
report example 196
EXPORT
accounting data set creation 20
accounting data set, specifying 27
export accounting data set 20
Option 6 195
utility
overview of 195
use of 195
Utility panel 195
EXPORT service 416
exporting
SCLM data sets 195
extended CREF type variable 634
extended scope
architecture 183
build 239
promote 247
Extended Type 563
external compare option 210

F

feature, drawdown 142
feedback xxxi
field name metavariables 639
field name variables 633, 636
FILE format 360
flexible data set names
ALTC parameter 541
FLMGROUP 540
using FLMALTC 510
flexible data set naming
cross-project support 69
flexible naming 12
FLM@2ASM language definition 298
FLM@2C language definition 298
FLM@2CBE language definition 298
FLM@2CBF language definition 298
FLM@2CCE language definition 298
FLM@2CO2 language definition 298
FLM@2COB language definition 298
FLM@2FRT language definition 298
FLM@2PLE language definition 298
FLM@2PLO language definition 298
FLM@BD0 language definition 298
FLM@BD2 language definition 298
FLM@EASM language definition 298
FLM@EC language definition 298
FLM@ECO2 language definition 298
FLM@ECOB language definition 298
FLM@EPLO language definition 298
FLM@WBCC sample language definition 312
FLM@WBRC sample language definition 312
FLM@WDUM sample language definition 311
FLM@WEXE sample language definition 311
FLM@WICC sample language definition 311
FLM@WIPF sample language definition 311
FLM@WLNK sample language definition 311
FLM@WRC sample language definition 311
FLM@WTLK sample language definition 312
FLM@WXOLC sample language definition 312
FLM00CVE sample exec 127
FLMABEG macro
assembling and linking the project definition 38
creating project definition 25
FLMAEND macro 26, 493
FLMAGRP macro 25, 494
FLMALLOC macro
defining language definitions 35
FLMALLOC macro, defining language definitions 494
FLMALTC macro 28, 510
FLMATVER macro 513
FLMCMD command
CLIST command procedure 361
command line format 360
data set example 361
FILE format 360
interactive processing 361
invocation format 359
parameters 359
FLMCMD services 252, 365, 366
FLMCNTRL macro 516
FLMCPYLB macro
defining language definitions 35
FLMCSPDB translator 566
FLMDTLC translator 570
FLMGROUP macro 26, 540
FLMINCLS 35
FLMINCLS macro 316, 542
FLMLANGL macro
defining language definitions 35
FLMLNK subroutine interface
call format 362
character parameters 367
parameter conventions 363
pointer parameters 367
FLMLPCBL parser 571
FLMLPFRT parser 574
FLMLPGEN parser
used as a CLIST or REXX parser 579
used as a generic parser 579
used as a PL/I parser 579
used as a TEXT parser 580
used as an Assembler parser 578
FLMLRASM REXX Assembler parser 583
FLMLRB 35
FLMLRBLD macro 548
FLMLRC2 C, C++, and Resource file parser for workstation source 594
FLMLRC37 REXX C370 parser 597
FLMLRCBL REXX COBOL parser 587
FLMLRCIS MVS C/C++ parser with include set support 590
FLMLRDTL translator 601
FLMLRIPF Script and OS/2 IPF Source Parser 601
FLMLSS parser 605
FLMLTWST 307
FLMLTWST translator 608
FlMNPROM macro 549
FLMPROJ macro 549
FLMSYSLB 35
FLMSYSLB macro 36, 551
FLMTBMAP translator 619
FLMTCOND 35, 118
FLMTCOND macro 552
FLMTMJI translator 621
FLMTMSI translator 622

FLMTOPTS 35, 118
FLMTOPTS macro 556
FLMTPRE translator 623
FLMTPST translator 625
FLMTRNSL
  defining language definitions 35, 38
  defining translators 79
FLMTRNSL FUNCTN parameter 79
FLMTRNSL macro 316
FLMTXFER translator 627
FLMTYPE macro 26, 562
FLMXFER translator 308
forced mode, build 240
fragments, syntax diagrams xxiv
FREE DB2 application 302
FREE service 419
function invocation variables
  build group 637
  date 635, 637
  time 635, 637
functions
  build 237
  edit 149
  promote 245
  that use data sets 13
  utilities 156
  view 146

G

generic architecture definition
  DB2 support 301
generic architecture member
  restriction 275
  use of 275
generic output specifying the generic architecture member 275
GETBLDMP service 420
GOODRC 560
group
  defining authorization codes for 26
  definition of 139
  development layer 140
  development library 431
  guidelines for defining 141
  key
    overview 141
    promote report 248
  non-key
    overview 141
    promote report 248
  non-key testing techniques, primary 6
  primary non-key 6
  staging layer 141
  test 6
  variables description 640
  verification 150, 431
group found variable 635, 637
group_list
  FLMLRBLD macro 549
  FLMTCOND macro 553
  FLMTOPTS macro 556

H

HIER command 163
hierarchical view 140
hierarchy
  conversion errors 77
  defining 4
  description 140
  ensuring synchronization 287
  group concatenation 140
  moving data through 141
  promoting data 140
  search order 140
hierarchy navigation 230
hierarchy view
  unit of work utility 225
high-level architecture definition
  DB2 support 301
high-level architecture member
  application modularity 275
  controlling dialog software 275
  use of 274
history view, version utility 207

I

IDCAMS utility 18
impact assessment techniques 293
IMPORT
  Option 7 199
  utility
    using 199
IMPORT service 426
importing
  SCLM data 199
  SCLM data sets 199
INCL statement
  format 277
  use of 273
INCLD statement, use of 273, 277
include 293
include reference
  definition of 169
  panel 169
include reference variable 635, 639
include-sets for includes variable 635
information
  build report 243
Information Management 127
information mode
  build 240
INIT service 429
initial and save change code exit routine
  parameters 55
  specification 55
initial delete exit, DELINIT 62
INITIAL status 223
initialize parameter variables 363
input list translators 100
installing sample project data sets 42
interactive command processing 361
intermediate variables 633–639
internal security, SCLM 338
INVTARG status 223

ISAPACK flag 216
ISPF variables 372
ISPF-supplied line commands 227

J

JCL
converting to SCLM language definitions 118
job statement 252
JOVIAL 272

K

key group 141
key groups 141, 248
keyboard
navigation 643
PF keys 643
shortcut keys 643
keywords
assembler call statement 363
buildmap 173
FLMALLOC macro 494
FLMLANGL macro 545
FLMLRBLD macro 548
FLMPROJ macro 549
FLMTRNSL macro 557
in architecture member statements 278
keywords, syntax diagrams xxiv
KREF
in architecture statements 280

L

language
architecture member 277
constructs 629
variable 635, 638
language definitions
DB2 298
defining 32
general 32
macros 35
modify 32
new 79
sample 298, 311
SCLM-supplied 32
using multiple translators 79
language definitions using the edit function 155
language restrictions
on non-explicit references 629
on separation of references 630
layer, staging 140, 141
leaving member on promotion
relationship with SCLM 319
LEC architecture definition
DB2 support 300
library concatenations 140
library utility
browse accounting record 164
browse statistics 166
change code list 168
change promote processing 176
library utility *(continued)*
include list 169
member selection list 161
options 159
panel 157
reset member lock 174
Sets the Accounting Status 176
transfer ownership 174
understanding 263
update authorization code 173
user data entries 169
Where-used 174
limited scope 239
line commands 146
link project definition 38
LINK statement
format 280
use of 248
link-edit control architecture member
requirement 273
restriction 274
use of 273
link-edit processed components 273
linkage editor
creating 273
include 273
multiple 273
override options 273
producing 273
specify options 273
SSI field 274
using 273
verification 273
list commands
unit of work 227
list information array 371
LIST statement
format 281
use of 273
listing data set
temporary
compiler processed components 273
link-edit processed components 274
listing data set, output specification 390
Listing Type 210
listings
compressing 273, 274
saving
compiler processed components 273
link-edit processed components 274
LKED statement
format 281
use of 273
LMAP statement
format 281
use of 274
load module 8
LOAD statement
format 281
use of 281
load type 8
LOCATE command 163
LOCK service
invocation of 430

M

macro
  FLMABEG 25, 493
  FLMAEND 26, 493
  FLMAGRP 25, 494
  FLMALLOC
    using 35, 38
  FLMALTC 27, 28, 510
  FLMATVER 27, 513
  FLMCNTRL 27, 525
  FLMCPYLB 35, 38, 538
  FLMGROUP 26, 540
  FLMINCLS 35, 316, 542
  FLMLANGL
    using 35
  FLMLRB 35
  FLMLRBLD 548
  FLMNPROM 549
  FLMPROJ 549
  FLMSYSLB 35, 551
  FLMTCOND 35, 552
  FLMTOPTS 35, 556
  FLMTRNSL 35, 38, 316, 557
  FLMTYPE 26, 562
  initial 151
  instructions 491
  parameter, maximum length 492
  SCLM variables used in 492
  user-defined 156
macro call operand, maximum length 492
Main Menu panel
  action bar choices 146
  fields 146
maximum report lines 28
maximum VIO limit 29
MAXLINE control option 28
MAXVIO control option 29
member
  architecture 271
  definition of 139
  deleting 157, 160
  encoding and decoding 333
  historical information 177
member level locking
  maintaining SCLM administrators 255
member lock, resetting 174, 176
member promotion
  leaving member 319
member selection list
  accounting records 163
  library utility 161
memory, insufficient 143
messages
  ABEND 251
  array 368
  data set 254
  DBUTIL service 399
  ISPF 275
  output specification 390
  promote 248
  RPTARCH service 450
metavariables
  cross-reference 639
metavariables *(continued)*
  field names 639
  functions 639
  list of 639
  report 633, 639
  uses for 639
MIGRATE service 434
migration utility 177, 178
mixed mode 149, 151
MODBKUP status 223
modes
  browse 149
  build 239
  mixed 149, 151
  promote 247
modify control options 26
modify language definitions 35
modifying delete from group dialog interface 71
module, load 8
module, object
  creating 272
  include 273
  specify options 272
MOVE command 154
multicultural support 62, 377, 458, 608
multiple partitions, running SCLM in 3
multiple translator usage 79
MVS limitations 141

N

name
  FLMPROJ macro 549
  language definition 155
  profile 151
naming conventions of architecture members 277
National Language Support, *See* multicultural support
navigation
  keyboard 643
navigation, hierarchy 230
NEWBKUP status 223
NEXTGRP Service 438
NLS, *See* multicultural support
non-key group
  definition 141
  overview 141
  promote report 248
non-promotable
  FLMNPROM macro 549
noncomment lines 167
noneditable members 76
NOPROM service
  invocation of 441
normal scope
  build 239
  promote 247
notation conventions 491
NRETRIEV command
  SCLM considerations 144
number of versions to keep 28

O

OBJ statement
  format 281
object module
  creating 272
  include 273
  specify options 272
object type 8
OBSOLETE status 223
OPTFLAG 560
options, control
  ACCT 27
  ACCT2 27
  change code verification routine specification 55
  DASDUNIT 29, 527
  DSNAME 28
  EXPACCT 27
  MAXLINE 28
  MAXVIO 29
  OPTOVER 28
  user exits 57, 61, 62
  VERPDS 28
  VERS 28
  VERS2 28
  VIOUNIT 29
OPTOVER control option 28, 560
ordering compiler inputs automatically 272
output
  creating generic 275
  sending to a data set 254
Output
  build outputs 288
  default output member names 288
  languages of output members 288
  multiple build outputs 288
  sequential build outputs 288
output member name variable 635, 638
OUTx statement 281
overflow, data set 251

P

package backout utility
  backup phase 217
  delete package 223
  list members in package 222
  overview 216
  package functions 221
  restore command 224
  restore package 222
  restore phase 219
Package details file
  cleanup procedure 221
package functions option 221
Package Member Details panel 223
packages
  backing up 563
  deleting 402
packed data set
  editing 152
packed data set, saving 434, 451
panels
  build 238
panels *(continued)*
  controlling software for 275
  database contents-tailored 184
  edit 150
  main menu 145
  member selection list
    accounting records 163
  promote 246
  SCLM edit profile 155
parameters
  ACCTINFO service 380
  AUTHCODE service 384
  BUILD service 389
  character 367
  DBACCT service 394
  DBUTIL service 397
  DDNAME 366
  DELETE service 401
  DELGROUP service 404
  DSALLOC service 408
  EDIT service 393, 411
  END service 413
  ENDEC service 415
  EXPORT service 417
  FLMABEG macro 493
  FLMAEND macro 493
  FLMAGRP macro 494
  FLMALLOC macro 497
  FLMALTC macro 510
  FLMATVER macro 513
  FLMCNTRL macro 525
  FLMCPYLB macro 538
  FLMGROUP macro 540
  FLMINCLS macro 542
  FLMLANGL macro 545
  FLMLRBLD macro 548
  FLMPROJ macro 549
  FLMSYSLB macro 551
  FLMTCOND macro 552
  FLMTOPTS macro 556
  FLMTRNSL macro 557
  FLMTYPE macro 562
  FREE service 419
  IMPORT service 427
  INIT service 429
  LOCK service 432
  MIGRATE service 436
  NEXTGRP service 438
  NOPROM service 441
  PARSE service 442
  pointer 367
  PROMOTE service 445
  RPTARCH service 449
  SAVE service 452
  START service 458
  STORE service 460
  UNLOCK service 463
  VERDEL service 464
  VERHIST service 466
  VERINFO service 468
  VERRECOV service 471
PARM statement
  format 281
  use of 273

PARMx statement
format 282
use of 272
PARSE service
invocation of 442
parser
invoking 82, 84
user-defined 82
writing 83
parser restrictions 629
parser volume 151
partitioned data set, storing version of SCLM member 203
Pascal
integer variable 377
program sample 477
patterns for selection criteria 181, 367
performance considerations 360
personal lists
NRETRIEV command 143
PL/I program sample 487
pointer parameters
$acct_info 368
$list_info 371
$msg_array 368
$stats_info 370
precedence system 182
precedence verification 431
predecessor, definition of 431
primary
commands 146
group 142
primary non-key groups 6
printing data sets 253
PRMCOPY 54
PRMCOPY, Promote Copy user exit 58
PRMEXT1 54
PRMEXT1, Promote Verify user exit 58
PRMEXT2 54
PRMEXT2, Promote Copy user exit 58
PRMEXT3 54
PRMEXT3, Promote Purge user exit 58
PRMINIT 54
PRMINIT, Promote Initial user exit 58
PRMPRURGE, Promote Purge user exit 58
PRMPURGE 54
PRMVFY 54
PRMVFY, Promote Verify user exit 58
processing
batch 252
errors 251
processing interactive command 361
program sample, Pascal 477
program sample, PL/I 487
PROJDEFS data sets
allocation 11
naming convention 11
protecting 23
project
controls 26
converting to SCLM 75
define new languages for 79
defining 3
environment backup and recovery 70
name 25
project cleanup 269
project definition
alternate 4, 24
assembly of 38
data 4
generation of 4
linkage of 38
primary 4
sample of 46
specification 23
project environment
backup and recovery 70
definition of 3
generation of 3
protecting 23
project environment, definition 139
project manager scenario 39
project partitioned data sets
allocation of 12
naming convention 12, 28
protecting 23
project-defined line commands 227
PROM statement
format 283
use of 275
Promote
by change code 275
promote function
data contention 251
data set overflow 251
error messages 247, 248
generating a report 247
modes 247
panel 246
processing 247
report 248
scopes 247
Promote function
package backout 220
PROMOTE service 444
promoting members 265
propagating applications 294
protect SCLM data sets 26
purge process 251

R

RACF (Resource Access Control Facility) 23
READ access 23
REBIND option 305
rebuilding a changed member 265
records
accounting 164
build map 170
user data entries 170
recovery of database 70
REFRESH command 163
repeatable items, syntax diagrams xxiv
report
accounting statistics 187
architecture information 188, 190
build 242
change code 186
cleanup 188

report *(continued)*
cross-reference information 188
cutoff 189, 450
data set 254
database contents utility 183
examples 183, 190, 242
lines, maximum 28
output specification 390, 437
promote 248
source listing 187
tailored 184, 185
variables 185
report only mode
build 240
promote 247
requirements for workstation build
workstation build requirements 307
reset member lock 174
Resource Access Control Facility (RACF) 23
Restored Date/Time field 224
RESTORED status 217, 223
retrieve option 212
return codes
BUILD service 391
CCEXITS service 393
DBACCT service 395
DBUTIL service 399
DELETE service 402
DELGROUP service 406
DSALLOC service 409
EDIT service 412
END service 413, 416
EXPORT service 418
FREE service 419
general categories 377
GETBLDMP service 423
GOODRC 560
IMPORT service 428
INIT service 430
LOCK service 433
MIGRATE service 437
PARSE service 443
PROMOTE service 447
RPTARCH service 450
SAVE service 454
SCLMINFO service 457
START service 458
STORE service 461
UNLOCK service 464
VERDEL service 466
VERHIST service 468
VERINFO service 471
VERRECOV service 474
REUSEDAY parameter 219
RPTARCH service 448

S

sample language definitions 298
sample program
Pascal 477
PL/I 487
sample project
installing the project data sets 42
sample project *(continued)*
overview 40
sample project utility, SCLM 254
save change code exit, CCSAVE 55
SAVE command 152
SAVE service 451
SCLM
defining a new language 101
defining a preprocessor 113
hierarchy 140
installing a project database 39
leaving member behind 319
running different versions 3
support for DB2 295
support for workstation builds 307
SCLM administrators
maintaining 255
SCLM Build 238
SCLM Build Entry Panel 238
SCLM command shell 252
SCLM commands 252
SCLM editor, using 262
SCLM Explorer 230
SCLM internal data pointer
definition of 368
variable 636, 638
SCLM introduction 139
SCLM language definitions, *See* language definitions
SCLM metavariables
account report fixed (@@FLM#AF) 639
account report long (@@FLM#AL) 639
SCLM sample project utility 254
SCLM Search 232
SCLM security 337
SCLM services
data set protection 367
general discussion 359
performance considerations 360
SCLM variables
access key (@@FLMACK) 633, 636
accounting group (@@FLMGRP) 633, 637
accounting group data set name (@@FLMDSN) 633, 637
accounting member (@@FLMMBR) 633, 638
accounting record type (@@FLMATP) 633, 636
accounting status (@@FLMSTA) 633, 639
accounting type (@@FLMTYP) 633, 639
alternate project definition (@@FLMALT) 633, 636
assignment statements (@@FLMASG) 633, 636
authorization code (@@FLMACD) 633, 636
authorization code change (@@FLMACC) 633, 636
blank lines (@@FLMBLL) 633, 636
buffer size in bytes (@@FLMSIZ) 633, 639
build group (@@FLMGRB) 633, 637
build map (@@FLM$MP) 633, 639
build map date (@@FLMMD4) 633, 638
build map date (@@FLMMDT) 633, 638
build map information (@@FLMBIO) 633, 636
build map name (@@FLMMNM) 633, 638
build map time (@@FLMMTM) 633, 638
build map type (@@FLMMSC) 633, 638
build mode (@@FLMBMD) 633, 636
calling function name (@@FLMFNM) 633, 637
change code (@@FLM$CC) 633, 639
change code data (@@FLM$C4) 634

SCLM variables *(continued)*
change code data (@@FLM$CD) 633
change code date (@@FLM$C4) 639
change code date (@@FLM$CD) 639
change code during parse phase (@@FLMCAA) 634, 636
change code time (@@FLM$CT) 634, 639
change date (@@FLMCD4) 634, 636
change date (@@FLMCDT) 634, 636
change group (@@FLMCLV) 634, 636
change time (@@FLMCTM) 634, 637
change user ID (@@FLMCUS) 634, 637
comment lines (@@FLMCML) 634, 637
comment statements (@@FLMCMS) 634, 637
control statements (@@FLMCNS) 634, 637
creation date (@@FLMID4) 634, 638
creation date (@@FLMIDT) 634, 637
creation time (@@FLMITM) 634, 638
CREF type (@@FLMCRF) 634, 637
CU list (@@FLMLST) 634, 638
data set name for OUT0 (@@FLMDO0) 634, 637
data set name for OUT1 (@@FLMDO1) 634, 637
data set name for OUT2 (@@FLMDO2) 634, 637
data set name for OUT3 (@@FLMDO3) 634, 637
data set name for OUT4 (@@FLMDO4) 634, 637
data set name for OUT5 (@@FLMDO5) 634, 637
data set name for OUT6 (@@FLMDO6) 634, 637
data set name for OUT7 (@@FLMDO7) 634, 637
data set name for OUT8 (@@FLMDO8) 634, 637
data set name for OUT9 (@@FLMDO9) 634, 637
database qualifier (@@FLMDBQ) 634, 637
DDNAME substitution list (@@FLMDDN) 634, 637
default type (@@FLMSRF) 634, 639
dependencies pointer (@@FLMLIS) 634, 638
destination group (@@FLMGRD) 634, 637
destination group data set name (@@FLMDSD) 634, 637
dynamic includes pointer (@@FLMINC) 634, 638
extended CREF type (@@FLMECR) 634, 637
extended type of source member (@@FLMETP) 635, 637
function invocation date (@@FLMFDT) 635, 637
function invocation time (@@FLMFTM) 635, 637
group found (@@FLMGRF) 635, 637
group found data set name (@@FLMDSF) 635, 637
include (@@FLM$IN) 635, 639
include sets for includes (@@FLM$IS) 635, 639
language (@@FLM) 638
language (@@FLMLAN) 635
language version (@@FLMLVS) 635, 638
member version (@@FLMMVR) 635, 638
number of change codes (@@FLMNCC) 635, 638
number of includes (@@FLMNIN) 635, 638
number of noncomment lines (@@FLMNCL) 635, 638
number of noncomment statements (@@FLMNCS) 635, 638
number of user entries (@@FLMNUE) 635, 638
OUT0 member name (@@FLMOU0) 635, 638
OUT1 member name (@@FLMOU1) 635, 638
OUT2 member name (@@FLMOU2) 635, 638
OUT3 member name (@@FLMOU3) 635, 638
OUT4 member name (@@FLMOU4) 635, 638
OUT5 member name (@@FLMOU5) 635, 638
OUT6 member name (@@FLMOU6) 635, 638
OUT7 member name (@@FLMOU7) 635, 638
OUT8 member name (@@FLMOU8) 635, 638
SCLM variables *(continued)*
OUT9 member name (@@FLMOU9) 635, 638
output member name (@@FLMONM) 635, 638
predecessor date (@@FLMBD4) 635, 636
predecessor date (@@FLMBDT) 635, 636
predecessor time (@@FLMBTM) 635, 636
project (@@FLMPRJ) 635, 638
prolog lines (@@FLMPRL) 635, 639
promote date (@@FLMPD4) 635, 638
promote date (@@FLMPDT) 635, 638
promote time (@@FLMPTM) 635, 639
promote user ID (@@FLMPUS) 636, 639
SCLM internal data pointer (@@FLMINF) 636, 638
SCLM version (@@FLMVER) 636, 639
static pointer (@@FLMSTP) 636, 639
sysprint DDNAME (@@FLMDDO) 636, 637
system user ID (@@FLMUID) 636, 639
target group (@@FLMTOG) 636, 639
target group data set name (@@FLMDST) 636, 637
top CU name (@@FLMCUN) 636, 637
total lines (@@FLMTLL) 636, 639
total statements (@@FLMTLS) 636, 639
translator version (@@FLMTVS) 636, 639
user data entry (@@FLM$UD) 636, 639
SCLMINFO service 455
SCOMPARE command 153
scopes
architecture 182
build 239
promote 247
SCOPY command 153
SCREATE command 154
secondary accounting data set, specifying 27
security 23
security, SCLM 337
selection criteria 367
selection criteria, specifying 181
selection parameters 367
sending to IBM
reader comments xxxi
service
ACCTINFO 380
AUTHCODE 384
BUILD 388
CCEXITS 392
character parameters 367
DBACCT 394
DBUTIL 395
DELETE 400
DELGROUP 402
DSALLOC 407
EDIT 410
END 413
ENDEC 414
EXPORT 416
FLMCMD interface 359
FREE 419
GETBLDMP 420
IMPORT 426
INIT 429
interactive command processing 361
invocation from programs 359
LOCK 430
MIGRATE 434

service *(continued)*
NEXTGRP 438
NOPROM 441
PARSE 442
pointer parameters 367
PROMOTE 444
return code categories 377
RPTARCH 448
SAVE 451
SCLMINFO 455
START 458
STORE 459
UNLOCK 462
VERDEL 464
VERHIST 466
VERINFO 468
VERRECOV 471
service command panels
allocation of output data sets 365
services, FLMCMD 252
Sets the Accounting Status 176
SETSSI command 274
shortcut keys 643
SINC statement
format 283
required 272
skeletons, ISPF 275
SMOVE command 154
SORT command 163
source listing report 187
source type 8
space computations, accounting data set definition 20
SPACE parameter 20
SPROF command 155
SREF statement
format 283
SREF statement, using 547
SREPLACE command 156
SSI field 274
staging
group 141
layer 141
START service 458
static pointer
definition of 368
using 368
variable 636, 639
statistical information
array 370
field descriptions 167
panel 166
record field format 370
STORE service
invoking 459
statistical information 166
subapplication
controlling 274
defining 274
sample 283
subapplication components 274
SUBMIT command 179
subproj_desc
FLMPROJ macro 549
subproject security, SCLM 349
subunit scope
architecture 183
build 239
promote 247
summary of changes xxxiii
supported data 8
synchronization, architecture definition 287
synchronizing data sets 70
syntax diagrams, how to read xxiv
sysprint ddname variable 636

T

tailored data set
definition of 183
format specification 185
options 184
report 185
sample of 185
temporary listing data set
LIST - compiler processed components 273
LMAP - link-edit processed components 274
testing with primary non-key group 6
title
on tailored report 184
title, on tailored report 399
Tivoli Information Management
sample user exit 30
Tivoli Information Management for z/OS 127
top CU name
variable 636, 637
tracking dynamic includes 99
trademarks 650
transfer ownership 174
translator
invocation 274
translators
FLMCSPDB 566
FLMDTLC 570
FLMLPCBL 571
FLMLPFRT 574
FLMLPGEN 578
FLMLRASM 583
FLMLRC2 594
FLMLRC37 597
FLMLRCBL 587
FLMLRCIS 590
FLMLRDTL 601
FLMLRIPF 601
FLMLSS 605
FLMLTWST 608
FLMTBMAP 619
FLMTMJI 621
FLMTMSI 622
FLMTPRE 623
FLMTPST 625
FLMTXFER 627
type
architecture 8
load 8
object 8
source 8
type, definition of 139

U

unconditional mode
  build 239
  promote 247
unit of work
  data set prefix 227
unit of work utility
  hierarchy view 225
  member list panel 228
UNLOCK service 462
UP command 163
UPDATE 23
update authorization code 173
user application data 139
user data entries
  accounting records 166, 169
  array record 371
  variable 636, 639
user exit routine specification
  audit version delete 61
  build 57
  delete 62
  example 64
  promote 57
user interface
  ISPF 643
  TSO/E 643
user-defined line commands 227
user-defined macros 156
user-defined parsers 82
using the database contents utility 266
utilities function
  architecture report 188
  audit and version utility 203
  database contents utility 179
  delete from group utility 213
  export utility 195
  import utility 199
  library utility 157
  migration utility 177
  package backout utility 216
  tailored data set 185
  tailored report 184
utilities function, DBUTIL service 395

V

variable 635
variables
  CLIST 360
  COBOL return code 377
  description of 631
  description of group 640
  field names 633
  FORTRAN 363
  functions 633
  initialize parameter 363, 365
  ISPF, used by SCLM services 372
  list of 632
  Pascal 363
  report 185, 633
  SCLM macros 492
  uses for 632
variables, syntax diagrams xxiv
VERCC, change code verification exit 53, 54
VERCC, change code verification routine 55
VERCOUNT parameter 28
VERDEL service 464
VERHIST service 466
verification
  access key 431
  authorization code 431
  authorization code authorization codes, 177
  build output 431
  bypass 283
  error processing 247
  group 431
  load module 273
  predecessor 431
  promote processing 252
verification change code 55
verify change code, CCVFY 55
VERINFO service 468
VERPDS control option 28
VERPDS data sets 28
VERRECOV service 208, 471
VERS control option 28
VERS2 control option 28
version of SCLM member, storing in a PDS 203
version utility
  compare member versions 209
  external compare option 210
  history of changes 207
  retrieve option 212
  version viewer 208
Versioning and audit tracking 204
versioning partitioned data sets 16, 28
versions of SCLM, running 3
View - Entry panel 147
view function
  description 146
VIO limit 29
VIOUNIT control option 29
VSAM
  accounting data sets 18
  audit control data sets 20
  cluster 17
  data set 18
VSAM data set
  storing audit information 203
VSAM data set, specifying with FLMCNTRL macro 513
VSAM Record Level Sharing 18, 28, 516, 526
VSAMRLS control option
  specifying 28
VSAMRLS parameter 516, 526

W

Where-used 174
work element list 230
workstation build support
  relationship with SCLM 307

Z

ZSAAUTH variable 384

ZSCCODE variable 382, 470
ZSCDAT4 variable 382, 470
ZSCDATE variable 382, 470
ZSCIACTF variable 456
ZSCIAUT variable 456
ZSCIGRP variable 456
ZSCILANG variable 456
ZSCINPAT variable 456
ZSCIPDEF variable 455
ZSCIPROJ variable 455
ZSCISVER variable 456
ZSCITMST variable 456
ZSCITYP variable 456
ZSCTIME variable 382, 470
ZSDNAME variable 382, 470
ZSDTYPE variable 382, 470
ZSIISET variable 382, 470
ZSIMBR variable 382, 470
ZSNGPKEY variable 438
ZSNXTGRP variable 438
ZSUENTRY variable 382, 470
ZSUNUM variable 382, 470
ZSVMBR variable 471

SC19-3625-30