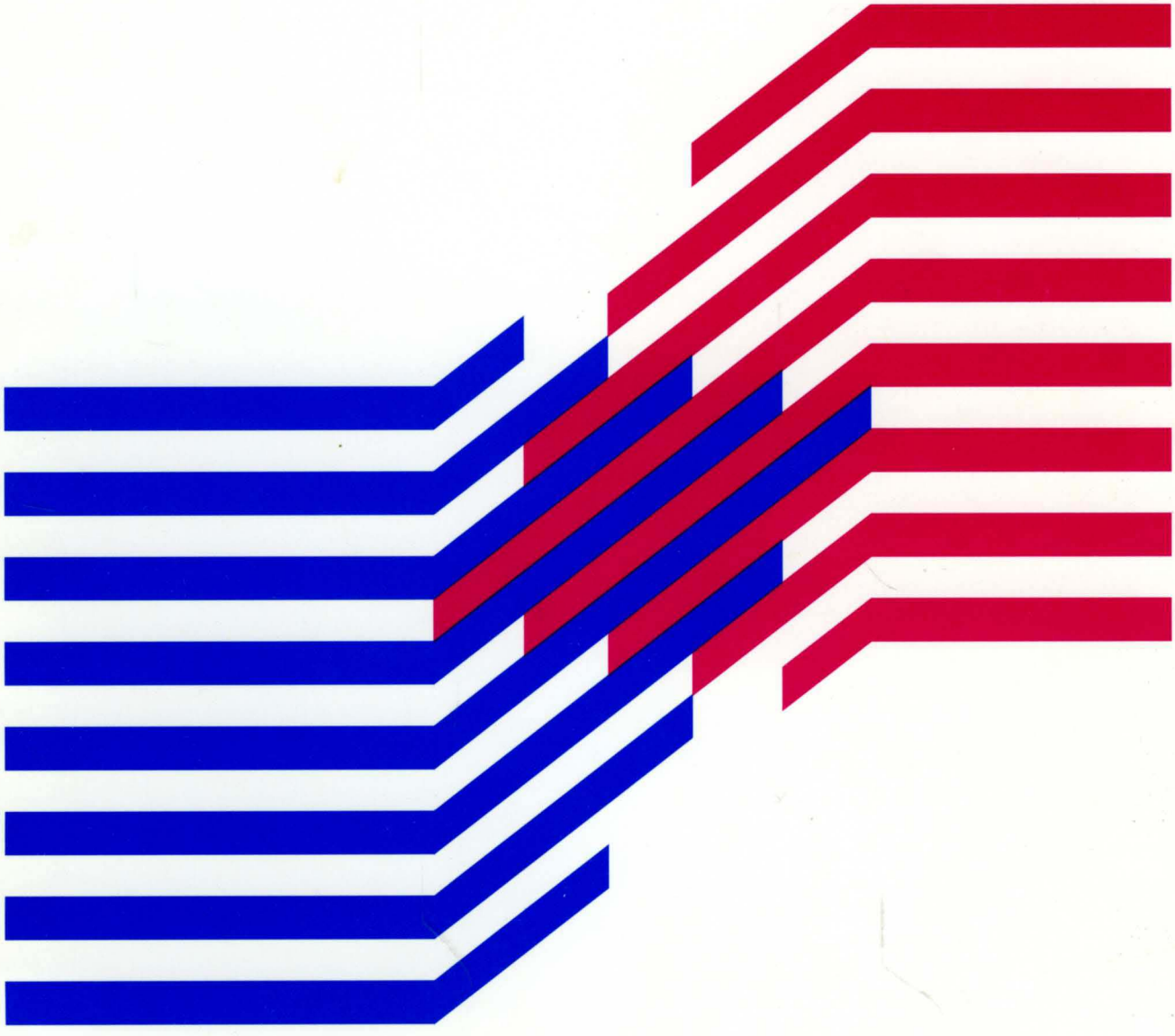
SC34-4254-0

IBM

Interactive
System Productivity Facility/
Program Development Facility (ISPF/PDF)

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

Version 3 Release 2 for MVS



SC34-4254-0

Interactive
System Productivity Facility/
Program Development Facility (ISPF/PDF)

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

Version 3 Release 2 for MVS

First Edition (March 1990)

This publication applies to Version 3 Release 2 of the licensed program Interactive System Productivity Facility/Program Development Facility (ISPF/PDF or PDF) for MVS (5665-402) and to all subsequent releases and modifications until otherwise indicated in new editions of this publication or Technical Newsletters. It is for use with the Interactive System Productivity Facility (ISPF) for MVS (5685-054), Version 3 Release 2, MVS Version 2 Release 2 or later, and TSO/E Version 2 Release 1.

This edition applies to subsequent releases and modifications of this program until otherwise indicated. The licensed programs and related licensed material described herein are provided by IBM under the Agreement for IBM Licensed Programs.

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 MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. **This disclaimer does not apply in the United Kingdom or elsewhere if inconsistent with local law.**

This publication may include inaccuracies or errors. IBM may change this publication and/or the product described herein. Textual changes are indicated by a vertical line to the left of the change.

This book contains technical information that is not supported by IBM, such as examples of code, programs, and samples. Information herein serves as technical reference and guidance only.

IBM may have patents or pending patent applications covering subject matter described herein. This document neither grants nor implies any license or immunity under any IBM or third-party patents, patents applications, trademarks, copyrights, or other similar rights, or any right to refer to IBM in any marketing activities. Other than responsibilities assumed via the Agreement for Purchase of IBM Machines and the Agreement for IBM Licensed Programs, IBM assumes no responsibility for any infringement of third-party rights that may result from use of the subject matter disclosed in this document or from the manufacture, use, lease, or sale of machines or programs described herein.

Licenses under IBM's utility patents are available on reasonable and nondiscriminatory terms. IBM does not grant licenses under its appearance design patents. Direct licensing inquiries in writing to the IBM Director of Commercial Relations, International Business Machines Corporation, Armonk, New York, 10504.

References in this publications to IBM products or services do not imply that they will be available everywhere IBM operates, nor that only IBM's products or services may be used.

Publications are not stocked at the address below. Request IBM publications from your IBM representative or branch office.

A form for comments is provided at the back of this publication. Or you may address comments to: IBM Corporation, Department T45, P.O. Box 60000, Cary, North Carolina 27511. IBM may use and distribute information you supply without obligation to you.

Note to U.S. Government users — Documentation is related to restricted rights; use, duplication, or disclosure is subject to restrictions set forth in the GSA ADP Schedule Contract with IBM Corp.

© Copyright International Business Machines Corporation 1989, 1990. All rights reserved.

Special Notices

Before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, 4300, and 9370 Processors Bibliography*, GC20-0001, *IBM System/370 and 4300 Processors Bibliography of Industry Systems and Application Programs*, GC20-0370, for the editions that are applicable and current.

The following names, used in this publication, are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries:

IBM
MVS
MVS/DFP
Operating System/2
OS/2
Personal System/2
PS/2.

An asterisk (*) is used to identify the first time a trademarked name is used in the text.

This publication contains sample programs. Permission is hereby granted for you to copy and store the sample programs into a single data processing machine and for you to use the stored copies for internal study and instruction only. No permission is granted to use the sample programs for any other purpose.

© Copyright IBM Corp. 1989, 1990

Preface

This book provides reference and usage information, along with conceptual and functional descriptions of the Software Configuration and Library Manager (SCLM).

About This Book

This book is divided into three parts:

- "Programming Reference" contains information on all product functions for programmers, as well as an explanation of the architecture definition functions.
- "Library Administration" contains administrative and diagnostic information for project administrators. It defines the SCLM database, tells how to establish and monitor such a database, and explains the library functions.
- "Messages and Codes" contains a complete listing and description of messages and return codes issued by the SCLM functions.

General-Use Programming Interfaces

Unless specifically stated otherwise, the information in this manual must not be used for programming purposes. However, general-use programming interfaces are provided to allow the customer to write programs that use the services of ISPF/PDF. These interfaces are discussed in the following sections:

- Chapter 3, "SCLM Variables," describes the SCLM system variables.
- Chapter 5, "SCLM Services," describes the SCLM services.
- Chapter 8, "SCLM Macros," describes the SCLM macros.
- Chapter 9, "Advanced Topics," covers the following:
  - "Dynamic Include Tracking" describes how SCLM tracks dynamic includes.
  - "Change Code Verification Routines" describes how to code a change code verification routine.
  - "Build and Promote User Exit Routines" describes how to create a build and promote user exit routine.
- Chapter 12, "Messages and Codes" describes FLMCMD and Translator return codes.

Who Should Use This Book

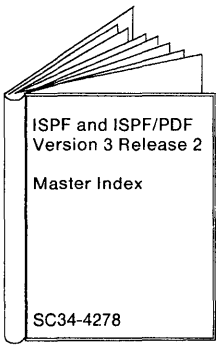
This book is for:

- Programmers whose projects are controlled by SCLM
- Project administrators who use SCLM to manage the software development process.

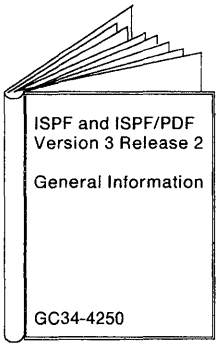
© Copyright IBM Corp. 1989, 1990

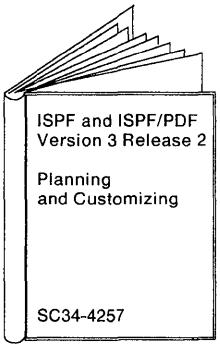
The ISPF and ISPF/PDF Library for MVS

General

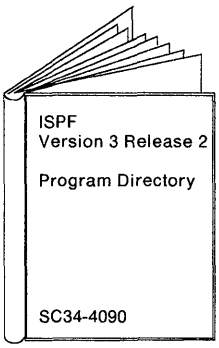


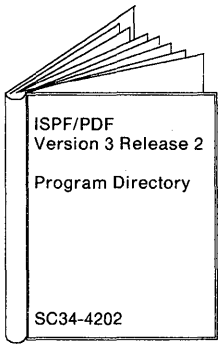
Evaluation and Planning



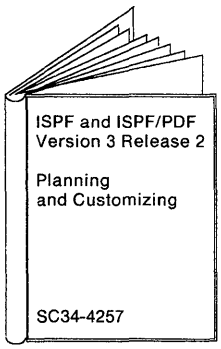


Installation and Migration

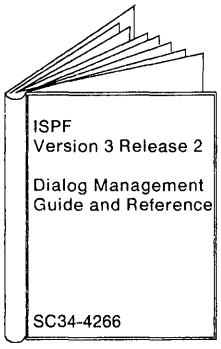


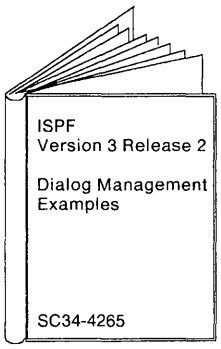


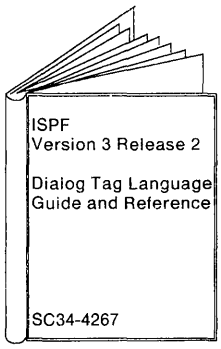
Customization

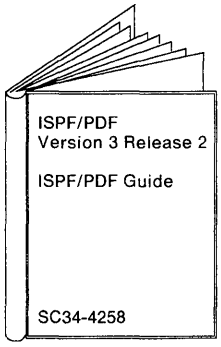


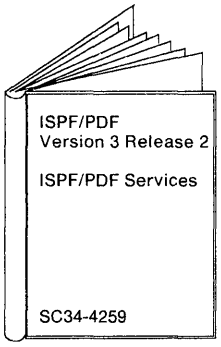
Programming

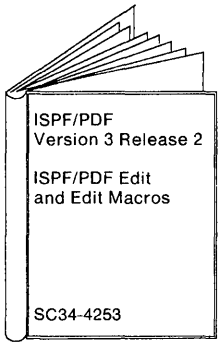


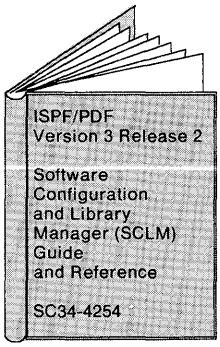


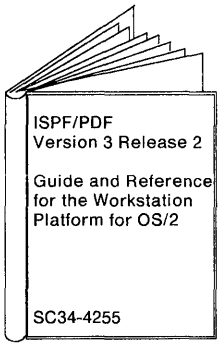


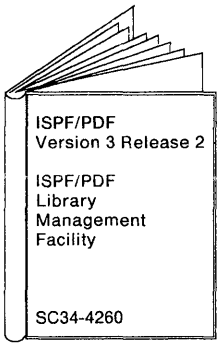




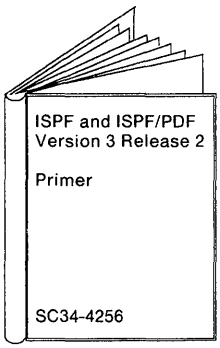


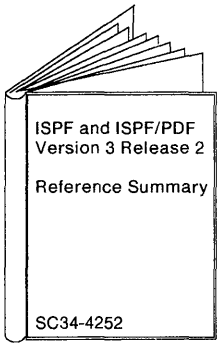






End Use





Related Publications

- *OS/VS2 MVS JCL*, GC28-0692
- *MVS Resource Access Control Facility (RACF) Command Language Reference*, SC28-0733
- *OS/VS2 MVS Utilities*, GC26-3902
- *TSO Extensions Version 2 Command Language Reference*, SC28-1881
- *MVS/XA Supervisor Services and Macro Instructions*, GC28-1154
- *OS Assembler H Language*, GC26-3771.
- *TSO Extensions Version 2 CLISTs*, SC28-1876

Contents

© Copyright IBM Corp. 1989, 1990

Figures

Tables

Summary of Changes

This Summary of Changes lists the major changes and enhancements for ISPF and ISPF/PDF Version 3 Release 2 (Version 3.2) for MVS*.

Functional Changes for ISPF Version 3 Release 2 for MVS

The following functional changes have been made for ISPF Version 3.2.

Improvements to Dialog Tag Language and Conversion Utility

Extensions to the Dialog Tag Language (DTL) have been made to support additional tags and tag attributes. The extensions include support for the following:

- Defining horizontal layout of panel elements as provided by the DIR=HORIZ attribute of the REGION tag. This enhancement allows multiple fields on the same line and the ability to define interactive fields on one side of the panel and an explanatory information area on the other side of the panel and other combinations of panel layout.
- Using the HELP=help-panel-name attribute on ABC and PDC tags. This provides field-level help capability for action bar and pull-down choices.
- Specifying cursor position on the PANEL tag. The developer can position the cursor at panel development time instead of using the cursor parameters of the DISPLAY and TBDISPL service.
- Specifying group headings for columns of ISPF table data. The new LSTGRP tag defines headings (in addition to the individual list column headings) for a single column or multiple columns.
- Using the HELP=help-panel-name attribute for LSTCOL tag. This provides field-level help for columns of ISPF table data.
- Using XLATL FORMAT=UPPER to allow translations of user input to uppercase prior to validation check.
- Specifying AUTOTAB attribute for data fields and list columns.
- Imbedding tag files into the DTL source through ENTITY definitions.
- Concatenating multiple DTL source files for a single execution of the conversion utility.

Enhanced DTL Compatibility with OS/2 Version 1.2 Dialog Manager

Changes have been made to DTL and the conversion utility to provide a greater level of compatibility with the DTL supported by the Dialog Manager component of Operating System/2* (OS/2*) Version 1 Release 2. All DTL supported by the OS/2 Dialog Manager is checked for syntax, and a warning message is issued for all DTL that is not supported by ISPF. While changes have been made to the syntax of some tags supported by ISPF, the conversion utility continues to support the syntax documented for the prior release and issues a warning message advising that the DTL source file be updated to the new syntax level (especially if the DTL source will be ported to OS/2 for use with the OS/2 Dialog Manager).

---

* See "Special Notices" on page iii for a complete list of the trademarks used in this book.

© Copyright IBM Corp. 1989, 1990

The ISPF Version 3.2 DTL conversion utility formats SBCS and DBCS panel text according to a more precise set of Asian formatting rules.

Significant improvements have been made to the *ISPF Dialog Tag Language Guide and Reference*. These improvements include the addition of a guide to using DTL and additional wording for many tags that better defines the formatting that occurs.

Adjustments have been made to ISPF run-time support for the ISPF Version 3.2 DTL enhancements.

Starting the DTL compiler has been simplified with the addition of an invocation panel that contains the input fields required by the DTL compiler.

Additional Help Support

ISPF Help support has been expanded to include the following:

- Support for field-level (contextual) help on action bar choices, pull-down choices, and list columns.
- Support for extended help after field-level help and message help.
- Support for keys help. The application developer can define a help panel that can provide the application user with a brief description of each key defined for a panel.

In addition, the handling of the display of help panels has been improved to ensure that the full help panel is displayed. Help panels always appear in a pop-up window if they are defined using DTL or if you specify the WINDOW keyword on the)BODY statement of the ISPF panel language.

The width and depth values specified on the HELP tag or on the WINDOW keyword must be valid for the device on which these help panels are displayed. As these values were not always referenced in ISPF Version 3.1, you may need to update existing ISPF Version 3.1 help panels with valid depth and width values before displaying them under ISPF Version 3.2.

Dialog Test Facility

The Dialog Test facility is now included in ISPF. Previously this was part of ISPF/PDF.

Message Text Definitions Increased

The long message text field of ISPF message definitions can now be up to 512 characters. This allows developers to define clearer, more descriptive messages.

You should update existing dialogs in which the system variable ZERRLM is defined to 78 characters. Using VDEFINE, set this variable to 512 characters.

Miscellaneous Enhancements

Three new system variables (ZCURFLD, ZCURPOS, and ZCURINX) provide the dialog application with information on the position of the cursor when the user submits a panel.

Functional Changes for ISPF/PDF Version 3 Release 2 for MVS

The following functional changes have been made to ISPF/PDF Version 3.2.

Partitioned Data Set Extended (PDSE)

ISPF/PDF Version 3.2 offers support for the partitioned data set extended (PDSE), a new data set type introduced in Data Facility Product (MVS.DFP*) Version 3.2. ISPF/PDF can allocate a PDSE through the ISPF/PDF data set utility option (Option 3.2). Unlike a partitioned data set (PDS), a PDSE automatically reuses space created when members are updated. PDSEs can use all current ISPF/PDF functions, such as Edit and Browse.

C/370 Language Support

ISPF/PDF Version 3.2 offers support for the C/370 language, including C/370 language models. These models help you define dialog elements while you are editing C/370 language files. In addition, ISPF/PDF Version 3.2 provides an interface into the foreground and batch compile dialogs supplied with the C/370 compiler.

Workstation Platform for OS/2

ISPF/PDF Version 3.2 provides a Workstation Platform for OS/2, a Personal System/2* (PS/2*) interface into SCLM. By using the Workstation Platform for OS/2 you can obtain a list of SCLM-controlled projects, check out members from these projects, perform work against the members, and check them back in.

- The Library List is an application that serves as a programmable workstation (PWS) front end to a development system using SCLM. It allows SCLM-controlled members to be downloaded to the PWS by using member lists of SCLM-controlled libraries.
- The Library List allows you to install PWS tools to manipulate SCLM-controlled library members. It also allows you to organize tools under generic "verbs" (called actions) which are sensitive to the type of the members selected.
- The Library List allows you to keep multiple member lists open at the same time. The information displayed in the member list can be customized by each user, and all information in the member list is saved across invocations.
- The Workstation Platform for OS/2 provides an Application Programming Interface (API) to a subset of the host SCLM functions.
- The Workstation Log serves as a central location for recording significant events (such as the invocation of a command or an error condition encountered during the processing of a command) that occur during application processing. An API to the log is provided to allow tools to add entries into the log along with an end-user interface to allow review of some or all entries in the log.

---

* See "Special Notices" on page iii for a complete list of the trademarks used in this book.

Additional Enhancements

- The browse interface service (BRIF) provides support for temporary end-of-file and dialog-specific primary commands.
- The Library Member List service (LMMLIST) now allows a dialog to specify that the entire list of members generated by the LMMLIST service is to be written to either the ISPF List data set or a sequential data set.
- The Data List Services (LMDINIT, LMDFREE, and LMDLIST) allow your dialog to manipulate data sets in a manner similar to ISPF/PDF Option 3.4. The services are processed similarly to the LMINIT, LMMLIST, and LMFREE services, creating an internal list of data sets and passing one data set name back on each LMDLIST request.
- The deletion of migrated data sets from the Data Set List utility (Option 3.4) no longer causes a recall of the data set. The installation can specify the volume name for the migrated data sets and a command (such as HDELETE) to be run against those data sets.
- There is now one more "Additional Input Library" field on each of the Foreground and Batch compiler interface panels.
- A new type of line number has been added to the Edit COPY command. This allows the end user to determine whether COPY should be sensitive to ISPF/PDF statistics mode and use the proper portion of the line number.
- The LMF problem resulting when a needed part is locked and the owner is out sick, on vacation, or away from the office has been corrected. Any authorized project administrator can now promote or free the locked part on behalf of the user.
- ISPF/PDF Version 3.2 supports more edit models for SCLM project definition macros and for architecture definitions.
- The ISPF/PDF Logo panel now includes copyright information.

ISPF and ISPF/PDF Version 3 Release 2 for MVS Library

- Two new books have been added:
  - *ISPF and ISPF/PDF Master Index*
  - *ISPF/PDF User's Guide and Reference for the Workstation Platform for OS/2.*
- Titles of three previous books have been changed and some restructuring done to each:
  - *ISPF Dialog Management Services and Examples* has been renamed to *ISPF Dialog Management Examples.*

    This restructured manual contains only ISPF examples. ISPF services are now in *ISPF Dialog Management Guide and Reference.*
  - *ISPF Dialog Management Guide* has been renamed to *ISPF Dialog Management Guide and Reference.*

    This manual contains the information from *ISPF Dialog Management Guide*, as well as information about the ISPF services.
  - *ISPF Conversion Utility User's Guide and Reference* has been renamed to *ISPF Dialog Tag Language Guide and Reference.*

This manual has been expanded for Version 3.2 to include additional information on using the Dialog Tag Language (DTL).

See Table 1 for a comparison of the complete Version 2.3, Version 3, and Version 3.2 libraries.

Table 1 (Page 1 of 2). ISPF and ISPF/PDF Library for MVS

ISPF and ISPF/PDF Version 2 Release 3	**ISPF and ISPF/PDF Version 3**	**ISPF and ISPF/PDF Version 3 Release 2**
ISPF and ISPF/PDF General Information GC34-4116	ISPF and ISPF/PDF General Information GC34-4133	ISPF and ISPF/PDF General Information GC34-4250
ISPF and ISPF/PDF Installation and Customization SC34-4117	ISPF and ISPF/PDF Planning and Customizing SC34-4134	ISPF and ISPF/PDF Planning and Customizing SC34-4257
ISPF and ISPF/PDF Primer SC34-4122	ISPF and ISPF/PDF Primer SC34-4139	ISPF and ISPF/PDF Primer SC34-4256
What's New in ISPF and ISPF/PDF GC34-2172-3	What's New in ISPF and ISPF/PDF GC34-2172-4	Not available for Version 3.2 library. See "Summary of Changes" in this book.
	ISPF and ISPF/PDF Directory of Programming Interfaces for Customers GC34-4128	Not available for Version 3.2 library. Information integrated in Version 3.2 library.
ISPF Licensed Program Specifications GC34-4114	ISPF Licensed Program Specifications GC34-4212	ISPF Licensed Program Specifications GC34-4262
ISPF Dialog Management Guide SC34-4112	ISPF Dialog Management Guide SC34-4213	ISPF Dialog Management Guide and Reference SC34-4266
ISPF Dialog Management Services and Examples SC34-4113	ISPF Dialog Management Services and Examples SC34-4215	ISPF Dialog Management Examples SC34-4265
	ISPF Conversion Utility User's Guide and Reference SC34-4216	ISPF Dialog Tag Language Guide and Reference SC34-4267
ISPF/PDF Licensed Program Specifications GC34-4115	ISPF/PDF Licensed Program Specifications GC34-4185	ISPF/PDF Licensed Program Specifications GC34-4251
ISPF/PDF Guide SC34-4118	ISPF/PDF Guide SC34-4135	ISPF/PDF Guide SC34-4258
ISPF/PDF Services SC34-4119	ISPF/PDF Services SC34-4136	ISPF/PDF Services SC34-4259
ISPF/PDF Library Management SC34-4120	ISPF/PDF Library Management Facility SC34-4137	ISPF/PDF Library Management Facility SC34-4260
ISPF/PDF Edit and Edit Macros SC34-4121	ISPF/PDF Edit and Edit Macros SC34-4138	ISPF/PDF Edit and Edit Macros SC34-4253
		ISPF/PDF User's Guide and Reference for the Workstation Platform for OS/2 SC34-4255

Table 1 (Page 2 of 2). ISPF and ISPF/PDF Library for MVS

ISPF and ISPF/PDF Version 2 Release 3	ISPF and ISPF/PDF Version 3	ISPF and ISPF/PDF Version 3 Release 2
		ISPF and ISPF/PDF Master Index SC34-4278
	ISPF/PDF Software Configuration and Library Manager (SCLM) Guide and Reference SC34-4235	ISPF/PDF Software Configuration and Library Manager (SCLM) Guide and Reference SC34-4254
ISPF Summary Card, SC34-4124 ISPF/PDF Summary Card, SC34-4125 ISPF/PDF Edit and Edit Macros Summary, SC34-4126	ISPF and ISPF/PDF Reference Summary SC34-4214	ISPF and ISPF/PDF Reference Summary SC34-4252
Bill of Forms Number SBOF-0420 includes: ISPF Dialog Management Guide SC34-4112 ISPF Dialog Management Services and Examples SC34-4113 ISPF Summary Card, SC34-4124 ISPF Binder, SX66-0209 ISPF Cover Inserts, SX66-0210 Bill of Forms Number SBOF-0419 includes: ISPF/PDF Guide, SC34-4118 ISPF/PDF Services, SC34-4119 ISPF/PDF Summary Card, SC34-4125 ISPF/PDF Binder, SX66-0209 ISPF/PDF Cover Inserts, SX66-0211 Bill of Forms Number SBOF-0361 includes: ISPF/PDF Edit and Edit Macros, SC34-4121 ISPF/PDF Edit and Edit Macros Command Summary Card, SC34-4126 ISPF/PDF Edit Macros Binder, SX66-0213 ISPF/PDF Edit Macros Cover Inserts, SX66-0212	Bill of Forms Number SBOF-1032-0 orders all of the ISPF and ISPF/PDF for MVS library books. No new binders or cover inserts are available.	Bill of Forms Number SBOF-1196-0 orders all of the ISPF and ISPF/PDF for MVS library books. No new binders or cover inserts are available.

Ordering Information

You can order the ISPF and ISPF/PDF Version 3.2 for MVS books separately or use Bill of Forms number **SBOF-1196-0** to order the complete set.

The ISPF and ISPF/PDF Version 3.2 library will be available when the Version 3.2 products are generally available.

Part 1. Programming Reference

© Copyright IBM Corp. 1989, 1990

Chapter 1. SCLM Concepts and Terminology

The Software Configuration and Library Manager (SCLM) allows you to define a project database. It has functions for building, manipulating, and tracking data stored in the database.

How to Use This Manual

This manual is part of the ISPF/PDF library and assumes that you are familiar with the operation of ISPF/PDF in the MVS environment.

Chapter 1, "SCLM Concepts and Terminology" provides an overview of SCLM. All SCLM users should read this chapter first. The rest of the chapters in the manual assume that you have read and understood Chapter 1. The first part of the chapter describes the library structure and gives you an overview of the SCLM functions. In particular, one section explains the SCLM database structure. The rest of the chapter describes basic SCLM functions and discusses the capabilities and uses of each function. After reading Chapter 1, project administrators can go directly to Chapter 7, "Defining the Project." Developers and project managers should continue with Chapter 2.

Chapter 2, "Architecture Definition," describes how to use architecture members (individual software component definitions). 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.

Chapter 3, "SCLM Variables," lists the SCLM variables by field name and identifies each function you can use them with.

Chapter 4, "SCLM Dialog Interface," describes how to use the ISPF dialog interface, select SCLM functions to retrieve or process certain information, and generate reports on the information stored in project databases. It also describes information stored in accounting, cross-reference, and intermediate records for members in the project databases.

Chapter 5, "SCLM Services," introduces and describes the services that you use to retrieve and process certain information that you store in the project databases. It lists the general categories of SCLM service return codes and provides command and call invocation formats, return codes, and parameters for each service. It also explains the notation conventions used to document the services.

Chapter 6, "A Sample Program Using SCLM Services," provides a sample program in Pascal that allows you to invoke SCLM services.

Chapter 7, "Defining the Project," describes how to generate a project definition by discussing the steps you use to customize the database for a specific project. It explains the steps that enable you to create the database that best meets the needs of your project.

Chapter 8, "SCLM Macros," introduces and describes the macros that you use to create project definitions in SCLM. It also explains the notation conventions used to document the macros.

© Copyright IBM Corp. 1989, 1990

Chapter 9, "Advanced Topics," describes advanced topics that aid you in managing complex configurations.

Chapter 10, "Language Restrictions," describes restrictions that apply to the support SCLM provides for languages.

Chapter 11, "IBM Ada Setup," describes the language definitions that you must use and the setup operations you must perform to use the IBM Ada compiler.

Chapter 12, "Messages and Codes," explains the messages that you receive using SCLM. The chapter shows programmer responses, project administrator responses, and an explanation for each code. It also lists the FLMCMD command processor and SCLM translator return codes.

Library Structures and Naming Conventions

A *project database* in SCLM is a set of logically ordered MVS partitioned data sets (known as libraries) under a single high-level qualifier. Database organization is flexible enough to accommodate both small and large projects. Data can reside in a single data set or can be distributed among a series of data sets. SCLM does not control or limit the number of data sets you can maintain. Your ability to access data anywhere in the database eliminates the need for duplicating data.

SCLM tracks all updates to the database, thus allowing you to concentrate on developing programs rather than on locating data in data sets.

Projects, Groups, and Types

Data set names in SCLM must follow a standard naming convention consisting of three levels of qualification, for example, `PROJECT1.USER1.SOURCE`. You can allocate data set names using any attribute as long as the data sets are partitioned data sets. All data sets belonging to a specific project have the same high-level qualifier, known as the *project identifier*.

Related project data sets must be organized with a common middle-level qualifier to form *groups*. Each group consists of a set of data sets that contain the different kinds of data maintained.

Groups can contain a variety of project data. The low-level qualifier of the SCLM-controlled data sets, such as SOURCE, OBJECT, and LOAD, identifies the kinds of data maintained in a specific group (source code, object code, load modules that can be processed), which are known as *types*.

A group is made up of a set of types. You can store source code for programs in one type and object code in another type. However, you do not have to limit similar kinds of data to one type. In a project you can have source code distributed among multiple types or all source code residing in one type. You must declare the same types for all the groups in a project. For example, if the development group has SOURCE, OBJECT, LOAD, and LISTING types, all the other groups in the project must have those same types.

Members

Each component in a partitioned data set controlled by SCLM is a *member*. Libraries are composed of a series of members representing different units of data. A member can contain any kind of data. For example, a load module or an included unit for a program can be a member. SCLM stores units of data in the database as members of partitioned data sets. Therefore, members are the *discrete elements* of an SCLM database.

Figure 1 depicts a typical database organization. In the sample project there are nine different groups — the release group, the test group, the integration group, three staging groups, and three development groups. Groups USER1, USER2, and USER3 are the development layer, while STAGE1, STAGE2, and STAGE3 are the staging layer. Similarly, groups INT, TEST, and RELEASE are called the integration, test, and release layers, respectively.

Note: Libraries that make up a development group are called private libraries. A *private library* is a partitioned data set (PDS) or partitioned data set extended (PDSE) belonging to a group in the development layer of the hierarchy. A development layer contains groups that do not allow other groups to promote into them.

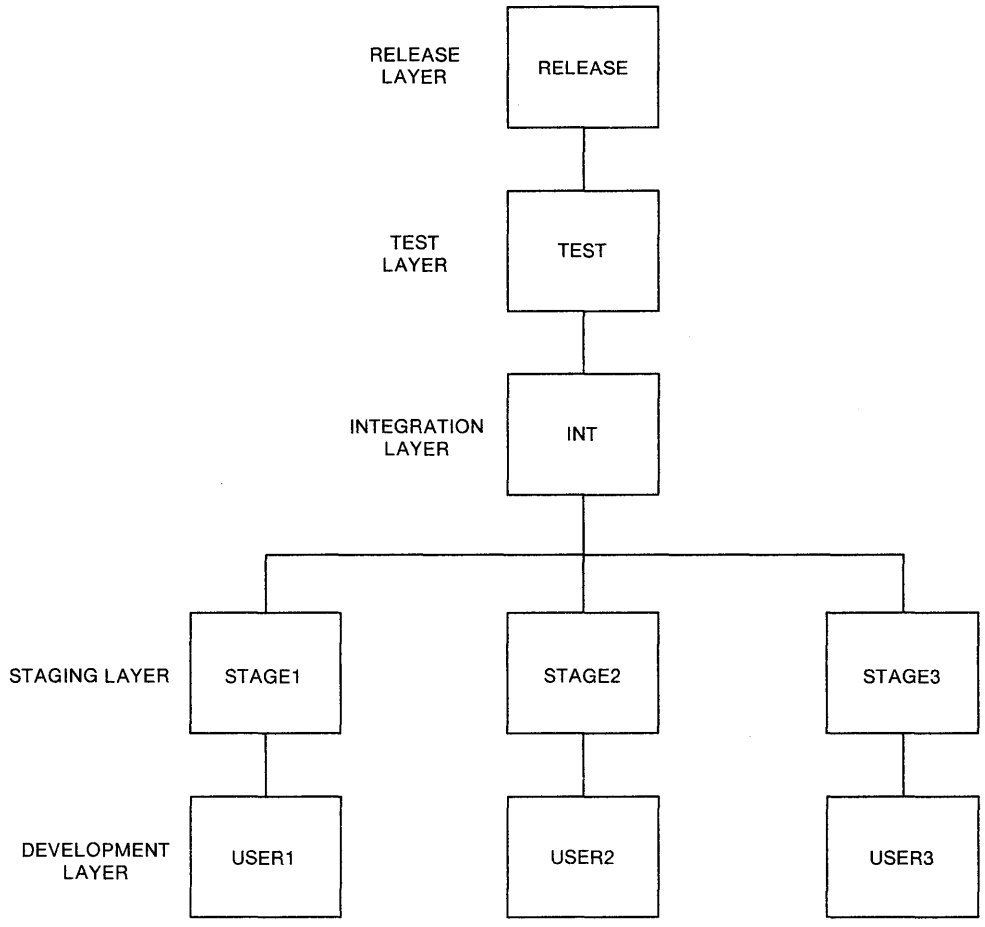


Figure 1. Typical Project Database Organization

Hierarchies

The project database illustrated above is organized into groups, each group being subordinate to the one above it. This form of database organization is known as a *hierarchy*. The concept of a multiple group hierarchy allows you to concatenate groups to form a complete project. A concatenation of groups is called a *hierarchical view*.

Hierarchies are always allocated from bottom to top. Thus, when you reference data, the members at lower positions in the hierarchy take precedence over members at higher positions.

Hierarchies allocated from different layers can represent different versions of the project. See Figure 2 for an illustration of two hierarchical views.

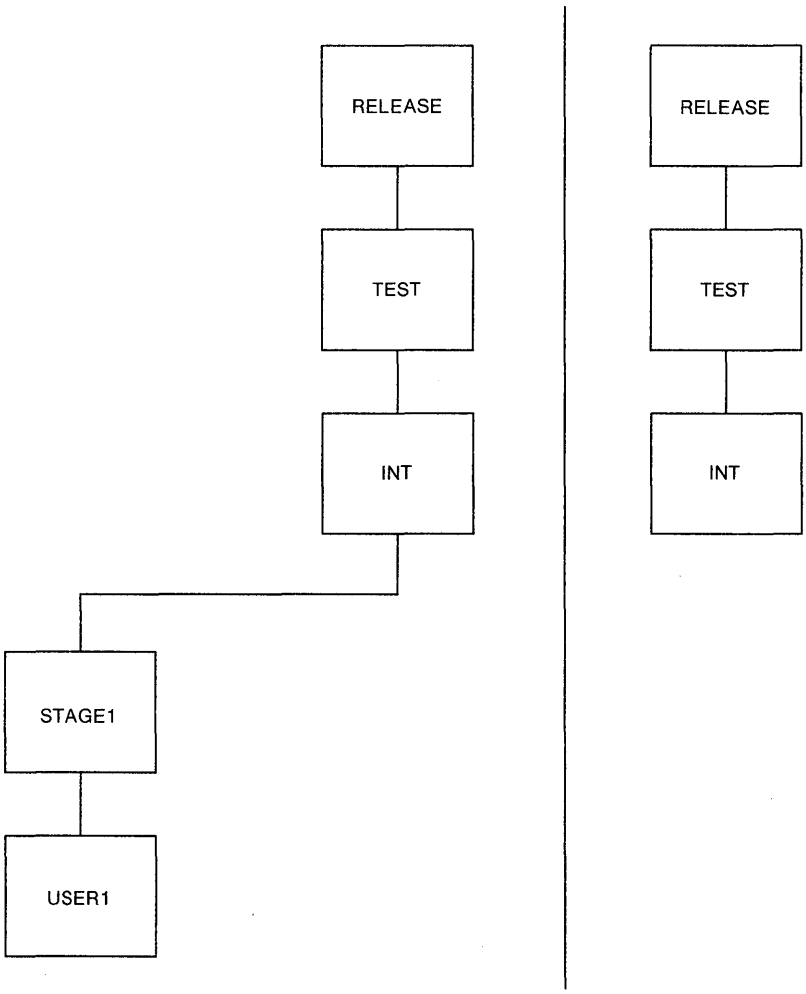


Figure 2. Two Hierarchical Views of the Same Database Organization

Key/Non-Key Groups

You can further distinguish groups in the project database as *key* groups and *non-key* groups. A maximum of 16 groups in any hierarchy must contain all the software components of the application under development. These 16 groups are key because of this special significance. A project can have as many key groups as you want as long as any hierarchical view has no more than 16.

SCLM allows a project to specify up to 16 transition groups between key groups. These groups are known as non-key groups. When you move data up 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.

Moving Data Through the Hierarchy

When you move data from group to group, the following rules apply.

- Copy units from key groups to non-key groups
- Move units from non-key groups to non-key groups
- Move units from key groups to key groups
- Move units from non-key groups to key groups and purge from the previous key group.

In this manner, the combination of all key groups represents all the software components of the project.

In general, when SCLM accesses a hierarchy from a particular group, it allocates only the necessary groups. If the lowest level in the hierarchy to be accessed is non-key, SCLM allocates it, and all the non-key groups above it, up to the next key group. From there, SCLM allocates only the key groups. If the starting group in the hierarchy to be accessed is key, then SCLM allocates only it and the key groups above it. The number of allocated groups cannot exceed 16.

The one exception to this allocation 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 allocate them in a hierarchy. Groups that must be allocated 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. Any hierarchy can have a combined maximum of 16 primary groups.

Guidelines for Defining Groups

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. For exceptions, see "Primary Non-Key Group Testing Techniques" on page 258.

SCLM Data Set Naming Conventions

SCLM limits data set names to three levels of qualification. See "Projects, Groups, and Types" on page 6 for a more detailed description of the qualifiers.

The first level of qualification corresponds to the project name. All data sets controlled by SCLM for a specific project must have the same high-level qualifier. The middle-level and low-level qualifiers correspond to the group and type, respectively. Therefore, form partitioned data set names in the following manner:

```
project_name.group_name.type_name
```

SCLM Functions

SCLM functions allow you to browse, create, update, delete, compile, link, promote up the hierarchy, and report on data stored in a project's database. You can generate reports with the build, promote, and utilities functions.

You can call SCLM functions in a variety of environments. In addition to the SCLM dialog interface, you can call SCLM functions independently with a command line processor or a program service interface except for the browse, edit, SCLM library utility, and migration utility functions. See Chapter 5, "SCLM Services," for more information.

This part of the chapter describes the basic SCLM functions and discusses the capabilities and uses of each. The five basic functions are:

- Browse
- Edit
- Utilities
- Build
- Promote.

Chapter 4, "SCLM Dialog Interface," describes how to call these functions.

Browse Function

The browse function uses ISPF/PDF Browse to allow you to display data in a project database. For more information on browse, see "Browse (Option 1)" on page 45.

Edit Function

The edit function is an interface to the ISPF/PDF editor. The SCLM editor ensures that editing occurs only in private libraries. It automatically locks the member when you begin the edit session. When you end an edit session, it parses and stores edited members and their accounting information.

The editor uses the LOCK, PARSE, and STORE services to identify and control members. This process verifies user authorization and prohibits simultaneous updates of members. The following pages describe these services.

LOCK: The LOCK service ensures that even though two versions of the same member may exist concurrently in parallel private libraries, only one copy can be promoted to a higher group in the hierarchy.

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 private library, LOCK prohibits others from editing the same member in their private 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 may exist in two private libraries at the same time. See "Defining Authorization Codes for a Group" on page 192 for more information, or see the project administrator for the project.

The LOCK service provides the following capabilities:

- *Verifying a group*

  SCLM locks members only when you copy them into a private library. SCLM copies a member to your private library when you edit the member. Group verification allows SCLM to control all source modifications to the higher levels 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 "Defining Authorization Codes for a Group" on page 192 for more information.

- *Verifying predecessors*

  The LOCK service guarantees that the member to be locked in the private 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 object files. (These members are referred to as "non-editable" elsewhere in this book.)

- *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.

PARSE: SCLM gathers statistical and dependency information by parsing each member it controls according to the syntax rules of the language of the member. Parsers supplied with SCLM gather the following statistical information for each member:

- Number of comment statements
- Number of non-comment statements
- Total number of statements
- Number of comment lines
- Number of non-comment lines
- Number of blank lines
- Total number of source lines
- Names of members referenced by an include construct
- Names of JOVIAL compools referenced

- Names of compilation units and their dependencies (Ada-type language only). A *compilation unit* is the smallest Ada language unit that compiles separately.

SCLM provides parsers for a variety of languages. But you can define project-specific parsers to use instead. See "Invoking User-Defined Parsers" on page 239 for more information. If you define a project-specific parser, you can gather the following additional statistical information:

- Prolog lines
- Control statements
- Assignment statements.

See "Statistics" on page 57 for a description of each of these fields.

SCLM supplies parsers for the following languages:

- Ada
- Assembler
- BookMaster
- CLIST
- COBOL
- EDL
- FORTRAN
- JOVIAL
- Pascal
- PL/I
- SCRIPT/VS.

Note: A packed member will not parse correctly if you use an SCLM-supplied parser. Before you promote the member, make sure the profile for the member you are editing has the ISPF/PDF pack mode off. See "Edit (Option 2)" on page 46 for more information.

STORE: For every member it processes, SCLM stores statistical, dependency, and historical information in the project's database. SCLM gathers this information from a variety of sources: dependency and statistical information from the PARSE service, historical information from partitioned data set (PDS) or partitioned data set extended (PDSE) directory information and user input, and change code information from user input to 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.

Change code information relates problem report (PR) and change request (CR) numbers 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. See "Change Code Verification Routines" on page 261 for more information.

Use the STORE service to enter data in the accounting information for a member. To add user data to the accounting information, you must design a software configuration management system, taking advantage of the SCLM services, or design a user-defined parser. But you cannot add user data through the ISPF/PDF dialog directly.

Because SCLM treats each compilation unit as an independent entity, many of the rules defined for members apply to compilation units as well. For example, you lock compilation units the way you lock members. However, compilation units are not locked until SCLM stores the member containing the compilation unit.

The STORE service verifies that any compilation units to be stored with the member are not present outside the hierarchy. The STORE service also verifies that the compilation unit does not reside in a different member within the hierarchy. It uses the authorization code of the member for the compilation unit verification.

You can retrieve accounting information using the database contents and the library utilities or the SCLM services.

Utilities Function

The SCLM utilities function allows you to browse accounting information, members, or build maps for a project. The utilities also allow you to extract accounting information for a project for purposes of reporting, generating command data sets, or creating input for other tools.

The SCLM utilities consist of the library, Ada sublibrary management, migration, database contents, and architecture report utilities.

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 records and to update authorization codes.

Ada Sublibrary Management Utility

The Ada sublibrary management utility allows you to browse or delete intermediate records and forms for compilation units.

Migration Utility

The migration utility allows you to verify authorization codes, to prohibit simultaneous updates of members, and to 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.

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 edit function.

Database Contents Utility

The database contents utility allows you to create an input stream containing variables associated with SCLM accounting data. This accounting data can then be extracted for members in the database that meet selection criteria you specify. You also control the order and format of the data extracted. The utility generates a report that lists the members that match your selection criteria.

Unlike other SCLM functions, the database contents utility does not verify the group, type, or member parameters you specify. This feature allows you to report on accounting information that is no longer defined in the project definition. For example, if the project administrator removes a development group from the hierarchy, the utility can still retrieve accounting information for that group.

Architecture Report

This report examines the requested *architecture* and all of its references, and then constructs a report of the architecture. In this book, architecture refers simply to the organization of software components to form integrated applications.

The architecture report is divided into three parts: 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 given application. This part of the report can help you eliminate unnecessary code. The cross-reference information indicates where a given software component is imbedded in the architecture of the application.

Examples of the architecture report appear in Figure 39 on page 85, Figure 40 on page 87, and Figure 41 on page 89.

Build Function

The build function does the following:

- Ensures total project integrity by verifying that all components defined to the architecture being built are present and complete
- Performs necessary (or requested) compiles and links
- Conditionally saves compiler and linkage editor output in the database.

Build compiles, links, and integrates software components according to the architecture. For any group in the database, 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.

Build uses internal data, such as accounting records and build maps, to determine when components have changed. With this information, the build function selectively builds components in the database to conserve machine resources. At the completion of the build, SCLM produces a report identifying which components were built and which components were out of date.

Build Input

Input to the build can be either a source member or an *architecture member*. An 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.

For a complex application, you can use architecture members to specify how the individual components of an application relate and how they are processed. You can create architecture members and treat them as source, but you must register them with SCLM (using the SCLM editor or migration utility) before the build. Chapter 2, "Architecture Definition," discusses the contents and uses of architecture members in greater detail.

Build Maps

SCLM creates *build maps* to identify how the build changed the database. Build maps contain a complete analysis of the database at the time of the build; that is, they include the names of all referenced members and the last change date and version number of each member. Additionally, build maps list those source members in the build that are *include structures* of other members in the build.

An include structure is a generic term for code that you insert when the source member is compiling. The syntax of an include statement in a program is language-dependent and is defined by language syntax rules.

SCLM stores the build maps with the accounting records. Once generated, the build maps are used by subsequent builds to determine whether changes have occurred since the last build.

The promote function also uses build maps to determine which members can be promoted.

Build Processing

Build processing consists of verification and dependency processing.

- Verification

  First, SCLM determines which architecture and source members will take part in the build. Then it verifies that all necessary architecture and source members have correct accounting information. SCLM also verifies that the build scope you specify is consistent with, and not less than, the scope defined in the language definition. If SCLM detects an error during this scope verification, it does not call the next phase, regardless of the build mode. See "Build (Option 4)" on page 91 for more information.

- Dependency processing

  Dependency processing consists of generating a build map, calling appropriate translators, and generating a report.

  - Generating a build map

    This task creates a build map for each software component taking part in the build. SCLM uses the build map to determine whether the translators are to be called. If an error occurs while generating a build map, processing for this component ends. However, SCLM can continue processing the remaining unbuilt members depending on the build mode you specify.

  - Calling translators

    Build calls appropriate translators if the build maps created contain out-of-date components.

    SCLM compares return codes from the translators to the return codes defined in the language definition to determine whether the translation was successful.

- Generating a report

  The final step in dependency processing is generating a report. The build report lists the results of the build.

Build Scopes

SCLM provides four scopes of build processing to accommodate the special processing required for compilation unit dependencies (Ada-type languages). Build scopes include the following: limited, normal, subunit, and extended.

Build processes two types of compilation unit dependencies: upward dependency and downward dependency.

- An upward dependency member is processed *before* a given member.
- A downward dependency member is processed *after* a given member.

For more information, see "Build (Option 4)" on page 91.

Build Modes

SCLM provides four modes of build processing: conditional, unconditional, forced, and report only.

You can use the modes to do the following:

- Check for unacceptable compile or link return codes (conditional)
- Generate translator listings for all components processed (conditional)
- Process software components despite translator errors (unconditional)
- Force build to compile and link all requested components again regardless of the previous status of the modules (forced)
- Generate a complete build report without performing the actual build (report only).

For more information, see "Build (Option 4)" on page 91.

Build Report

The build report provides a synopsis of the build. It includes:

- The date and time of the build
- The scope and the mode used
- The name of the component that was requested to be built
- The component's last change date and time
- The project definition used.

The build report also lists the software components that were rebuilt and saved in the database, that is, those components that passed the compilation or linkage edit phase. The report also shows the build maps that required regeneration, along with a list of out-of-date software components that caused the regeneration.

Figure 43 on page 93 shows an example of a build report.

Build Messages

The build function issues processing messages to allow you to monitor the build progress during the verification and the dependency processing.

During verification, the error messages identify only those components that do not have correct accounting information or that do not exist. During dependency processing, the messages identify those components being compiled and linked.

Build also issues return codes from all translators called. If any errors occur during processing, it generates appropriate informational messages.

Build Listings

The build function generates compiler and linkage editor listings in a listings data set. You can specify in the project definition, individual architecture members, or both, to have your listings saved in the database.

You can choose to receive all compiler and linkage editor listings or only error listings in the listings data set. If you request error listings, build only produces the listings that resulted from compiles or links with unacceptable return codes.

Automatic Ordering

The build function orders compiles and links to provide complete application integrity. Build compiles programs in the correct sequence to ensure that all dependencies are resolved. SCLM provides link ordering when load modules include other load modules and processes all load modules in the correct order.

Promote Function

The promote function does the following:

- Determines which components are eligible for promotion
- Verifies that the database is complete and current
- Promotes the components that are at the current group and scope
- Potentially purges the components from the current group (and possibly lower key groups)
- Generates a promote report.

Promote gives you an easy and efficient method to move data through a database. 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.

At the completion of the promote, the promote function generates a report identifying the components promoted.

Promote Processing

Promote processing consists of verification, copy, purge, and report generation.

Processing occurs sequentially. Depending on the promote mode you select, one or more of these phases may not occur. The following paragraphs discuss each phase.

- Verification

  This phase ensures that all software components to be promoted are available, have correct accounting information, and are current. To be current, the application to be promoted must not have been modified since its software

components were successfully built into an application. Promote compares each software component to be promoted to the database to verify that it has not been modified. A verification error results when SCLM finds an out-of-date software component. Verification always occurs.

- Copy

  This phase copies members to a group in the next layer of the hierarchy. The promote occurs alphabetically by type, one type at a time. Promote performs enqueuing during promotion to ensure that the database is secure. Therefore, multiple promotions may occur concurrently with no loss of database integrity. Verification always occurs.

  Copy errors occur when the target group does not have enough space or directory blocks. If a copy error occurs, promote again after you correct the problem. The second promote does not repeat work completed during the first promote.

- Purge

  This phase deletes members from the current and possibly the lower groups after they are successfully copied to the next higher group.

  Depending upon the classification of the "from" group and the target group, SCLM may or may not purge members from the lower group after promotion. SCLM may not delete a member to ensure that its most recent version is in a key group (even though it may also be in a non-key group). For more information, see "Key/Non-Key Groups" on page 8.

  The purge process uses the following strategy for groups that promote from:

  - ***Key group to key group:*** SCLM deletes members of the "from" group after a successful copy.
  - ***Key group to non-key group:*** SCLM does not delete members of the "from" group after the copy.
  - ***Non-key group to non-key group:*** SCLM deletes members of the "from" group after a successful copy.
  - ***Non-key group to key group:*** SCLM compares copied members to the corresponding members at the next lower key group.
    - If you can edit the copied member and it exactly matches the lower key group, SCLM deletes the member from both groups.
    - If the member is an output of the build and exactly matches or is a more recent version of the member at the lower key group, SCLM deletes the member from both groups.

- Report Generation

  SCLM generates a report no matter what the results were from the previous phases. If verification fails, the report reflects which software components are eligible for promotion. Otherwise, the report indicates the results of the promote.

Promote Scopes

The promote function provides three processing scopes. They are normal, subunit, and extended. SCLM provides the subunit and extended scopes for promoting members with compilation units.

Promote processes components and members the same way build processes these scopes. See "Promote (Option 5)" on page 94 for more information.

Promote Modes

Promote has three processing modes: conditional, unconditional, and report only. Each mode is available to all SCLM users.

You can use the modes to do the following:

- Bypass the copy and purge phases if promote discovers a verification error (conditional)
- Perform copy and purge processing despite verification errors, but promote only those members with correct accounting information (unconditional)
- Promote software components for incomplete or partial applications (unconditional)
- Perform verification and report generation processing (report only).

For more information, see "Promote (Option 5)" on page 94.

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. In report-only mode, the report displays a list of members eligible for promotion.

Figure 45 on page 97 shows an example of a promote report.

Promote Messages

The promote function issues processing messages to allow you to monitor the promote progress during verification, copy, and purge processing. If any errors occur during processing, promote generates appropriate informational messages.

During verification, the error messages identify only those components that do not have correct accounting information or that do not exist.

Chapter 2. Architecture Definition

If you are responsible for a given software component, you must define how SCLM should process that component. SCLM allows you to define the architecture for an application that determines how individual software components are to be tracked and maintained.

To define the architecture, you need to describe all subcomponents that make up the component, all data types of the component, and any special options for the component. You also need to select where all data types of a component will reside. You can provide this information in one architecture member for each software component.

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 database. 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 members that you can use to generate an architecture definition for an application. They are compilation control (CC), linkedit control (LEC), high-level (HL), and generic.

Each kind of architecture member controls a different kind of component that SCLM processes. Table 2 categorizes the use of each kind of architecture member.

Table 2. 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.

Each of these uses is described in the following pages. See "Sample Application Using Architecture Definitions" on page 31 for an example of an application consisting of architecture members.

© Copyright IBM Corp. 1989, 1990

Defining Compiler Processed Components

Standard compilers produce object modules as output. SCLM creates object modules if you specify either a compilable source member or a compilation control architecture member as input to the build function. The following discusses both methods for producing object modules; however, specifying source members simplifies the architecture definition.

Compilation Control Architecture Members

One method of creating object modules is through an architecture definition with Compilation Control (CC) architecture members.

CC architecture members contain all the information necessary to produce and track software components with object module output. Use CC architecture members to provide the following:

- The source member or members to be compiled
- Information concerning the target type names for the compiler outputs, such as object, listings, and compools
- Compiler options.

CC architecture members must have at least one SINC statement. See "Architecture Statements" on page 27 for more information. CC architecture members cannot reference other types of architecture members; therefore, you can only reference source to be compiled in the CC architecture member.

Because SCLM tracks included members, you only need to reference main source members. SCLM extracts the name of the compiler to be called from the language assigned to the referenced source member. If more than one source member is referenced, SCLM uses the language of the first referenced member. SCLM passes source members to the compiler in the order of reference; thus SCLM supports manual ordering of compiler input. One CC architecture member generates one call to a compiler regardless of the number of source members referenced.

You can override default compiler options by using the PARM statement. Use the statement as many times as necessary to specify all options you want. For source members being processed directly, you can only pass the default parameters defined in the language definition for the processor.

You can pass parameters directly to the translator by using the PARM*x* statement.

You can pass compile directives directly to the compiler using the CMD statement. You can insert the statements along with the source by careful positioning in the CC architecture member. In this manner, you can control compiler processing without modifying the source member. Use this feature to force titles on listings or to control compiler listing flow.

SCLM provides special statements for creating CC architecture members for JOVIAL programs. Use the COMP statement to identify database targets for generated JOVIAL compools. Use the statement the same way as the OBJ and LIST statements.

A *compool reference* is a reference to a JOVIAL data mapping structure that SCLM must compile before it can compile the current member. Compool references are specific to the JOVIAL languages. (JOVIAL programs that reference compools must use the CREF statement to indicate which type SCLM is to extract the referenced compools from.) If SCLM finds more than one CREF statement in a CC architecture member, it only uses information from the last one. For JOVIAL programs with no compool references, SCLM ignores the information.

Note: You must order compiles for JOVIAL programs with compool references. SCLM compiles the programs in the correct sequence to ensure that all dependencies are resolved if you create architecture members that reference all dependencies.

You must also order compiles for programs that have upward and downward dependencies, such as Ada. SCLM processes an Ada program *after* processing the upward dependencies, but *before* processing the downward dependencies. Thus SCLM compiles Ada programs in the correct sequence if the architecture member references all dependencies.

SCLM allows you to track and maintain all forms of generated data. Often, due to space limitations, you do not want to save it all. SCLM gives you the option of saving listings in the database or discarding them. Therefore, the architecture member statement LIST is optional. Nonetheless, SCLM generates listings to temporary listing data sets for your viewing during the build.

Specifying Source Members

The alternate method of creating object modules is to specify a source member to the build function. The source member's language definition in the project definition identifies which translators SCLM calls and where it saves output in the database. The language definition also specifies which compiler SCLM will call.

SCLM does not require that you use architecture members. You only need to reference the compilable source member because SCLM automatically tracks included members. See "Defining a Software Component using the FLMALLOC Macro" on page 216 for implementation details. This technique only works for source members.

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:

- The load module name and the type you want it saved in
- The linkage editor listing name and the type you want it saved in
- All object and other load modules the load module is to contain.

You can also specify linkedit control statements and linkage editor options. LEC architecture members must have at least one INCL or INCLD statement.

Construct a load module architecture member by creating an LEC architecture member that references source members, CC architecture members, LEC architecture members, or a combination. If the LEC architecture member references a CC architecture member or a source member, SCLM includes the

object module that results from a build of the member. If the LEC architecture member references another LEC architecture member, SCLM includes the load module produced during a build.

An LEC architecture member can have any number of CC and LEC architecture member references. However, LEC architecture members cannot reference high-level or generic architecture members. During processing, SCLM passes object and load modules to the linkage editor in the order of reference. Thus if linkage editor dependencies exist, carefully organize the CC and LEC architecture member references to resolve any problems.

You can use two methods to include other load modules in the load module to be generated. As previously discussed, a reference to the LEC that generated the other load module automatically includes it in the load module being generated. Alternately, you can link other load modules into the load module being generated by using the LINK statement. Although each method produces the same results, each allows you a different amount of control.

If you reference the LEC that generated the other load module, SCLM verifies that the other load module is also up-to-date. If you link the other load module, SCLM bypasses verification and includes the other load module even if it is not current.

Note: SCLM provides link ordering when load modules include other load modules. In other words, SCLM processes all load modules in the correct order.

Because the load module referenced by the LINK statement is not part of the build, the architecture member must reference this LEC to allow link ordering. SCLM does not order linked load modules unless specifically part of the build.

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 IBM S/370 linkage editor for all linkedits. To override the default linkage editor, use the LKED statement. You must define other linkage editors to be invoked in the project definition for the project. See "Step 5: Allocate Project Data Sets" on page 193 for more information.

You can specify that SCLM pass linkage edit control statements directly to the standard S/370 linkage editor in the LEC by using the CMD statement. Insert the control statements along with the object and load modules by careful positioning in the LEC architecture member.

Due to space limitations, you may not want online linkage editor listings. SCLM allows you to save listings in the database or discard them. Therefore, the architecture member statement LMAP is optional. Nonetheless, SCLM generates listings to temporary listing data sets for your viewing during the build.

You can use SCLM variables in LEC architecture members by using PARM and PARM*x* statements. SCLM substitutes the variables with the appropriate values before calling translators.

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.

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 which 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. A reference to this HL architecture member results in a reference to every software component in the application. Therefore, you can control the entire application through one HL architecture member. 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/PDF 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, you can control them 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.

If you want dialog-specific implied dependencies, you can reference the software components in other architecture members. As with other members referenced by the PROM statement, the date_check parameter allows SCLM to ignore implied dependencies. Otherwise, if you change the member, SCLM recompiles, relinks, or reprocesses the other software components referenced by the architecture member.

Careful use of the PROM statement in this manner can eliminate unnecessary SCLM processing and improve processing efficiency.

Defining Specially Processed Components

Generic outputs are produced by processors other than standard compilers and linkage editors. The SCRIPT/VS processor is an example of a nonstandard processor. You can create generic outputs by specifying the generic architecture member or source member as input to the build function. The following sections discuss both methods. However, keep in mind that referencing source members simplifies the architecture definition.

Generic Architecture Members

One method to create generic outputs is through a definition using generic architecture members. You can reference generic architecture members with HL architecture members.

Generic architecture members identify the source member or groups of source members to be processed by a processor other than a standard compiler or linkage editor. You should save information concerning the target types for the processor outputs in the generic architecture member. Generic architecture members must have at least one SINC statement.

Generic architecture members are equivalent to CC architecture members and, therefore, cannot reference other architecture members. Use them to invoke processors that do not produce standard object or load modules.

Generic architecture members allow you to use nonstandard processors. SCLM extracts the identification of the processors to be invoked from the language assigned to the source member last referenced. You must provide the definition of processor invocations in the project definition. See Chapter 7, "Defining the Project," for more information. You can insert processor directives along with the processor inputs by conscientious positioning of the CMD statements between processor input references.

You can specify parameters for the processor in the generic architecture member by using the PARM*x* and PARM statements. SCLM concatenates parameters to the default parameters defined for the processor in the language definition and passes them as specified to the processor. See Chapter 7, "Defining the Project." SCLM separates concatenated parameters by a comma and removes extraneous blanks. For a source member being processed directly, SCLM only passes the default parameters defined for the processor in the language definition.

Often, due to space limitations, you do not want to save all forms of data. SCLM gives you the option of saving listings in the database or discarding them. Therefore, the architecture member statement LIST is optional. Nonetheless, SCLM generates listings to temporary listing data sets for your viewing during the build.

Specifying Source Members

The alternate method of creating generic output is to specify a source member as input to the build function, or reference it with an HL architecture member. The source member's language definition in the project definition identifies where the compiler outputs are to be saved in the database.

You only need to reference main source members due to SCLM's automatic include tracking capability. Processors can generate up to 10 outputs to be saved in the database. You must identify each output to be saved by using a different OUT*x* statement. Indicate the processor to be invoked with the language identifier of the referenced member. See the DFLTTYP parameter description in "FLMALLOC Macro" on page 213.

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 statements. Architecture statements 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 lower case. You can write all statements, except for CMD, 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.

Member and type names must follow MVS naming conventions. SCLM does not check parameters and control statements for validity. They may continue through column 72.

Statement Uses

SCLM distinguishes architecture members from one another by their contents. For example, it assumes that members containing compilation information are CC architecture members and members containing linkedit information are LEC architecture members.

You use architecture statements to provide information about the design of applications in the project database.

Table 3 shows valid statements for each type of member.

Table 3. Valid Keywords for Architecture Member Statements

HL	LEC	CC	Generic
*	*	*	*
COPY	ALIAS	CMD	CMD
INCL	CMD	COMP	COMP
INCLD	COPY	COPY	COPY
PROM	INCL	CREF	CREF
	INCLD	LIST	LIST
	LINK	OBJ	OUT*x*
	LKED	OUT*x*	PARM
	LMAP	PARM	PARM*x*
	LOAD	PARM*x*	PROM
	OUT*x*	PROM	SINC
	PARM	SINC	SREF
	PARM*x*	SREF	
	PROM		
	SREF		

Each architecture statement is comprised of a keyword followed by one or more operands. The following list shows the valid statements, their usage, and their format:

***** Identifies an architecture comment statement.

```
* <comment>
```

ALIAS Identifies load module aliases to be generated. Use it only in LEC architecture members.

```
ALIAS <member_name> <type_name> <optional_comment>
```

CMD Identifies command statements to be included with inputs to the compiler, linkage editor, or other processors. The statement is positional; therefore, all blanks following this statement starting after the first blank are significant. Do not use the optional_comment with the CMD statement because it can cause unpredictable results. The CMD statement is not valid in HL architecture members.

```
CMD <control_statements>
```

COMP Identifies the name of the compool to be created (for JOVIAL programs only) and the type in which it is to reside. Use it only in CC and generic architecture members.

```
COMP <member_name> <type_name> <optional_comment>
```

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, isolate identical statements into a separate member and reference the member using the COPY statement. Referenced members must follow all formatting rules for architecture members.

The COPY directive 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>
```

CREF Identifies the type from which JOVIAL compools are to be resolved. This statement allows you to specify from which type SCLM accesses all referenced JOVIAL compools. Use it only in CC and generic architecture members.

```
CREF <type_name> <optional_comment>
```

INCL Identifies another architecture member that this architecture member references. It is not valid in generic or CC architecture members. You cannot use INCL to reference source members.

```
INCL <member_name> <type_name> <optional_comment>
```

INCLD Identifies a source member that this architecture member references. It is not valid in generic or CC architecture members.

```
INCLD <member_name> <type_name> <optional_comment>
```

LINK Identifies a load module to be linked into the load module being created. The referenced load module must be the product of another LEC. The build function does not verify the contents of a load module referenced by LINK. You can substitute the INCL statement to perform this verification. Use the LINK statement only in LEC architecture members.

```
LINK <member_name> <type_name> <optional_comment>
```

LIST Identifies the member and type in which the compiler listing is to reside. Use it only in CC and generic architecture members.

```
LIST <member_name> <type_name> <optional_comment>
```

LKED Identifies the linkage editor to be invoked. Use it only in LEC architecture members.

Language_id is an eight-character language identifier for a translator. The language ID specified must correspond to a valid language identifier defined in the project definition. See Chapter 7, "Defining the Project," for more information.

```
LKED <language_id> <optional_comment>
```

LMAP Identifies the member 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 name of the load module to be created and the type in which it is to reside. Use it only in LEC architecture members.

```
LOAD <member_name> <type_name> <optional_comment>
```

OBJ Identifies the name of the object module to be created and the type in which it is to reside. Use it only in CC architecture members.

```
OBJ <member_name> <type_name> <optional_comment>
```

OUT*x* Identifies the name of the output member to be created and the type in which it is to 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 translators of a compiler, linkage editor, or other processor. Use it in generic, CC, or LEC architecture members.

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. See Chapter 3, "SCLM Variables," for more information.

Do not use the optional_comment with the PARM statement because it can cause unpredictable results.

```
PARM <parameters>
```

PARM*x* Identifies parameters (options) to be passed to specific 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 the optional_comment with the PARM*x* statement because it can cause unpredictable results.

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>
```

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 (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, 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.

The date_check parameter for the PROM statement can alleviate implied dependencies. Specify with the date_check parameter that SCLM should disregard the last change date on the module. Thus SCLM ignores all implied dependencies. SCLM tracks members with the architecture member, but does not check them to verify that they are up-to-date.

Do not use the optional_comment with the PROM statement because it can cause unpredictable results.

```
PROM <member_name> <type_name> <date_check>
```

SINC Identifies the source member or group of source members to be processed. Use it only in generic and CC architecture members.

```
SINC <member_name> <type_name> <optional_comment>
```

SREF Identifies the type to be referenced during processing. Use it in generic, CC, and LEC architecture members.

```
SREF <type_name> <optional_comment>
```

Sample Application Using Architecture Definitions

The following application is composed of two subapplications. Each subapplication consists of two load modules, which are composed of a series of object modules. Load module LMOD1 and LMOD2 contain one object module each, while LMOD3 and LMOD4 contain multiple object modules. Figure 3 shows a diagram of the design of this application.

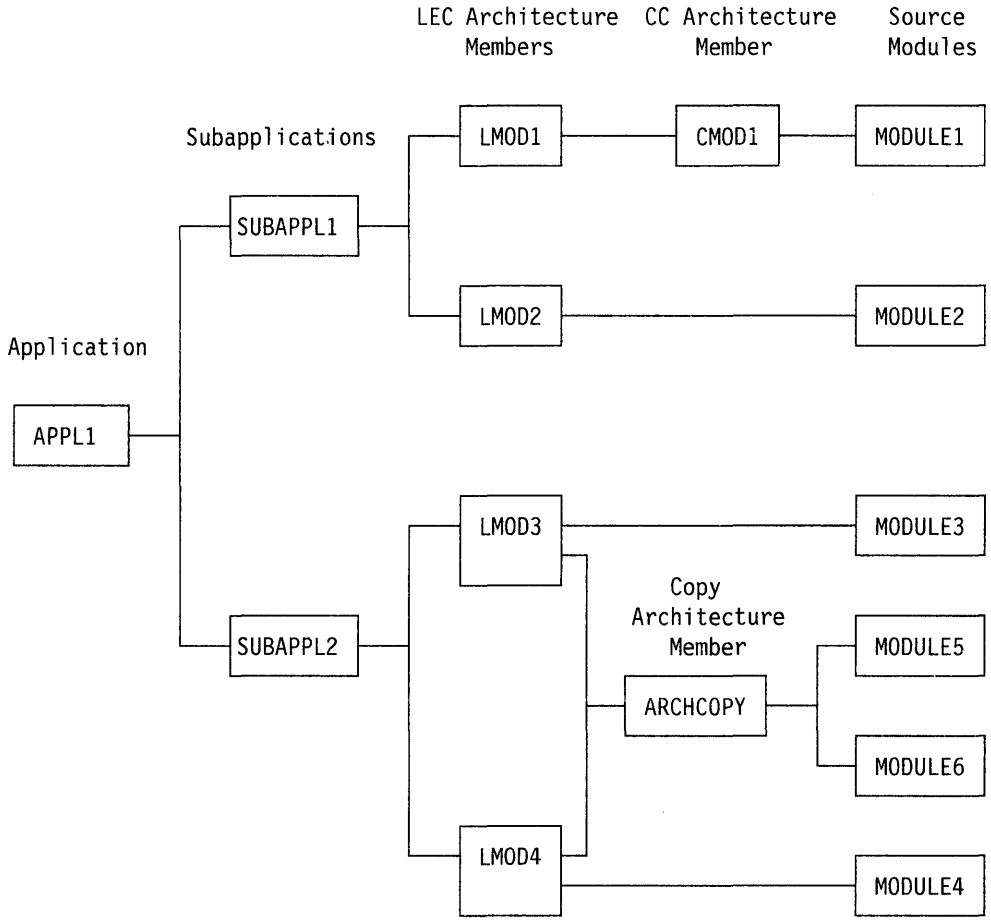


Figure 3. Application APPL1

Figure 4 on page 32 shows the architecture members for the APPL1 application.

High-Level Architecture Members

APPL1

```
*
*   Application APPL1
*
INCL SUBAPPL1 ARCHDEF
INCL SUBAPPL2 ARCHDEF
```

SUBAPPL1

```
*
*  Subapplication 1
*
INCL  LMOD1    ARCHDEF
INCL  LMOD2    ARCHDEF
```

SUBAPPL2

```
*
*  Subapplication 2
*
INCL  LMOD3    ARCHDEF
INCL  LMOD4    ARCHDEF
```

Linkedit Control Architecture Members

LMOD1

```
*
*    Load Module LMOD1
*
LOAD  LMOD1    LOAD
LMAP  LMOD1    LMAP
INCL  CMOD1    ARCHDEF
PARM  MAP,NCAL
PARM  LET
CMD    ALIAS MAIN1
```

LMOD2

```
*
*    Load Module LMOD2
*
LOAD  LMOD2    LOAD
LMAP  LMOD2    LMAP
INCLD MODULE2  SOURCE
PARM  MAP,NCAL,LET
CMD    ALIAS MAIN2
```

LMOD3

```
*
*    Load Module LMOD3
*
LOAD  LMOD3    LOAD
LMAP  LMOD3    LMAP
COPY  ARCHCOPY ARCHDEF
INCLD MODULE3  SOURCE
```

LMOD4

```
*
*    Load Module LMOD4
*
LOAD  LMOD4    LOAD
LMAP  LMOD4    LMAP
COPY  ARCHCOPY ARCHDEF
INCLD MODULE4  SOURCE
```

Figure 4 (Part 1 of 2). Architecture Members For Application Sample

Compilation Control Architecture Members

CMOD1

```
*
*    Object Module 1
*
OBJ   MODULE1   OBJ
LIST  MODULE1   LIST
CMD   %CHECK ON
SINC  MODULE1   SOURCE
CMD   %CHECK OFF
PARM  NOXREF,LC(75)
```

Copy Architecture Members

ARCHCOPY

```
*
* COPY ARCHITECTURE
*
INCLD MODULE5  SOURCE
INCLD MODULE6  SOURCE
PARM  MAP
```

Figure 4 (Part 2 of 2). Architecture Members For Application Sample

The HL architecture member in Figure 4 on page 32 includes references to two subapplications (SUBAPPL1 and SUBAPPL2). 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. CMD statements in LMOD1 and LMOD2 pass linkedit control statements to the linkage editor. Leading blanks are significant on CMD statements. (Note that inserting an extra blank before the statement satisfies linkage editor requirements for control statements.) Two PARM statements override the default linkage editor options.

Load modules LMOD3 and LMOD4 contain COPY statements. These statements identify the LEC architecture member ARCHCOPY, which references two source modules for SCLM to insert into the LMOD3 and LMOD4 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 ARCHCOPY affect LMOD3 and LMOD4 and all the other architecture members that might reference ARCHCOPY.

Because the CC architecture member is one of the lowest levels of architecture members, it references the actual source to be compiled or processed rather than other architecture members; that is, it only references main source members.

The control statements, designated by the CMD statement in the CC architecture member CMOD1, cause special compiler processing for the program. In addition, CMOD1 specifies compiler options with the PARM statement to override the default compiler options. The SINC statement references a source member rather than another architecture member.

See Figure 39 on page 85 for an architecture report of the APPL1 application.

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 sychronizes the resulting object and listing with the source. If you build an architecture definition, SCLM sychronizes 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, then those source, object, and load modules may no longer be synchronized.

In the example below, 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 used by many applications. 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). APPL1 and APPL2 are LEC architecture definitions that describe how to link edit LOAD1 and LOAD2, respectively. Finally, TOPARCH is a high-level architecture definition that includes APPL1 and APPL2.

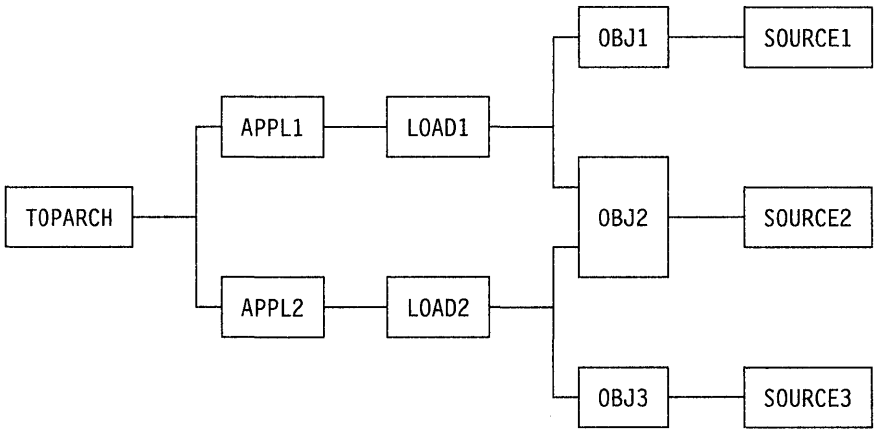


Figure 5. Example of Synchronization

In this example, all of the modules shown in the diagram exist only in the production level of your SCLM-controlled hierarchy and all source, object and load modules are synchronized. 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 timestamps) 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 scenario below 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 APPL1 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 APPL1 on the BUILD panel. LOAD2 will **not** be rebuilt, even though OBJ2 changed, because LOAD2 is outside of the scope of architecture definition APPL1. Herein lies the problem. When you promote APPL1, SCLM checks that everything that needs to be rebuilt (within the scope of APPL1) has been rebuilt. Unfortunately, modules outside the scope of APPL1 should be rebuilt as well.

When complete, all modules within the scope of APPL1 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 (in REPORT mode) as part of your testing. The build in REPORT mode indicates any out-of-sync modules by listing those modules that need to be rebuilt.

Chapter 3. SCLM Variables

General-Use Programming Interface

The SCLM variables are general-use programming interfaces, which you may use for programming purposes.

SCLM variables are character strings that SCLM replaces with a value. SCLM replaces these variables with eight-character values except for the following:

- @@FLM$XN variable has a value with a maximum length of 110.
- @@FLM$UD variable has a value with a maximum length of 128.
- @@FLM$LIS variable contains an address in decimal character format.
- @@FLM$STP variable contains an address in decimal character format.

Many of the variables can be used for certain SCLM functions only. Table 4 lists the SCLM variables in alphabetical order by field name and indicates which SCLM functions they can be used for. Table 5 on page 40 lists the SCLM variables in alphabetic order.

Table 4 (Page 1 of 3). SCLM Field Name Variables and their SCLM Functions

SCLM Field Name	Variable	Parse	Build	Promote	Utilities
Access Key	@@FLMACK				X
Accounting Group	@@FLMGRP	X	X	X	X
Accounting Member	@@FLMMBR	X	X	X	X
Accounting Record Type	@@FLMATP				X
Accounting Status	@@FLMSTA	X	X		X
Accounting Type	@@FLMTYP	X	X	X	X
Alternate Project Definition	@@FLMALT			X	X
Assignment Statements	@@FLMASG				X
Authorization Code	@@FLMACD				X
Authorization Code Change	@@FLMACC				X
Blank Lines	@@FLMBLL				X
Buffer Size in Bytes	@@FLMSIZ	X			
Build Map Date	@@FLMMDT				X
Build Map Name	@@FLMMNM				X
Build Map Time	@@FLMMTM				X
Build Map Type	@@FLMMSC				X
Change Code	@@FLM$CC				X

© Copyright IBM Corp. 1989, 1990

Table 4 (Page 2 of 3). SCLM Field Name Variables and their SCLM Functions					
SCLM Field Name	**Variable**	**Parse**	**Build**	**Promote**	**Utilities**
Change Code Date	@@FLM$CD				X
Change Code Time	@@FLM$CT				X
Change Date	@@FLMCDT				X
Change Group	@@FLMCLV				X
Change Time	@@FLMCTM				X
Change User ID	@@FLMCUS				X
Comment Lines	@@FLMCML				X
Comment Statements	@@FLMCMS				X
Compilation Unit Name	@@FLM$XN				X
Compilation Unit Type	@@FLM$XT				X
Compool	@@FLM$CM				X
Control Statements	@@FLMCNS				X
Creation Date	@@FLMIDT				X
Creation Time	@@FLMITM				X
Database Qualifier	@@FLMDBQ		X		X
ddname Substitution List	@@FLMDDN	X		X	X
Default Type	@@FLMSRF		X		
Dependencies Pointer	@@FLMLIS	X			
Dynamic Includes Pointer	@@FLMINC		X		
Include	@@FLM$IN				X
Language	@@FLMLAN				X
Language Version	@@FLMLVS				X
Member Version	@@FLMMVR				X
Next Group	@@FLMTOG			X	
Number of Change Codes	@@FLMNCC				X
Number of Compilation Units	@@FLMNCU			X	X
Number of Compools	@@FLMNCM				X
Number of Includes	@@FLMNIN				X

Table 4 (Page 3 of 3). SCLM Field Name Variables and their SCLM Functions

SCLM Field Name	Variable	Parse	Build	Promote	Utilities
Number of Noncomment Lines	@@FLMNCL				X
Number of Noncomment Statements	@@FLMNCS				X
Number of User Entries	@@FLMNUE				X
Predecessor Date	@@FLMBDT				X
Predecessor Time	@@FLMBTM				X
Project	@@FLMPRJ	X	X	X	X
Prolog Lines	@@FLMPRL				X
Promote Date	@@FLMPDT				X
Promote Time	@@FLMPTM				X
Promote User ID	@@FLMPUS				X
SCLM Internal Data Pointer	@@FLMINF		X		
SCLM Version	@@FLMVER				X
Static Pointer	@@FLMSTP	X			
System User ID	@@FLMUID		X		
Top CU Name	@@FLMCUN		X		
Total Lines	@@FLMTLL				X
Total Statements	@@FLMTLS				X
Translator Version	@@FLMTVS				X
User Data Entry	@@FLM$UD				X

Note: The build function does not support the use of SCLM variables on the FLMCPYLB macro.

Table 5 lists the SCLM variables in alphabetic order.

Table 5 (Page 1 of 3). SCLM Variables and their SCLM Functions

Variable	SCLM Field Name	Parse	Build	Promote	Utilities
@@FLMACC	Authorization Code Change				X
@@FLMACD	Authorization Code				X
@@FLMACK	Access Key				X
@@FLMALT	Alternate Project Definition			X	X
@@FLMASG	Assignment Statements				X
@@FLMATP	Accounting Record Type				X
@@FLMBDT	Predecessor Date				X
@@FLMBLL	Blank Lines				X
@@FLMBTM	Predecessor Time				X
@@FLMCDT	Change Date				X
@@FLMCLV	Change Group				X
@@FLMCML	Comment Lines				X
@@FLMCMS	Comment Statements				X
@@FLMCNS	Control Statements				X
@@FLMCTM	Change Time				X
@@FLMCUN	Top CU Name		X		
@@FLMCUS	Change User ID				X
@@FLMDBQ	Database Qualifier		X		X
@@FLMDDN	ddname Substitution List	X		X	X
@@FLMGRP	Accounting Group	X	X	X	X
@@FLMIDT	Creation Date				X
@@FLMINC	Dynamic Includes Pointer		X		
@@FLMINF	SCLM Internal Data Pointer		X		
@@FLMITM	Creation Time				X
@@FLMLAN	Language				X
@@FLMLIS	Dependencies Pointer	X			
@@FLMLVS	Language Version				X

Table 5 (Page 2 of 3). SCLM Variables and their SCLM Functions

Variable	SCLM Field Name	Parse	Build	Promote	Utilities
@@FLMMBR	Accounting Member	X	X	X	X
@@FLMMDT	Build Map Date				X
@@FLMMNM	Build Map Name				X
@@FLMMSC	Build Map Type				X
@@FLMMTM	Build Map Time				X
@@FLMMVR	Member Version				X
@@FLMNCC	Number of Change Codes				X
@@FLMNCL	Number of Noncomment Lines				X
@@FLMNCM	Number of Compools				X
@@FLMNCS	Number of Noncomment Statements				X
@@FLMNCU	Number of Compilation Units			X	X
@@FLMNIN	Number of Includes				X
@@FLMNUE	Number of User Entries				X
@@FLMPDT	Promote Date				X
@@FLMPRJ	Project	X	X	X	X
@@FLMPRL	Prolog Lines				X
@@FLMPTM	Promote Time				X
@@FLMPUS	Promote User ID				X
@@FLMSIZ	Buffer Size in Bytes	X			
@@FLMSRF	Default Type		X		
@@FLMSTA	Accounting Status	X	X		X
@@FLMSTP	Static Pointer	X			
@@FLMTLL	Total Lines				X
@@FLMTLS	Total Statements				X
@@FLMTOG	Next Group			X	
@@FLMTVS	Translator Version				X
@@FLMTYP	Accounting Type	X	X	X	X
@@FLMUID	System User ID		X		
@@FLMVER	SCLM Version				X
@@FLM$CC	Change Code				X

Table 5 (Page 3 of 3). SCLM Variables and their SCLM Functions					
Variable	**SCLM Field Name**	**Parse**	**Build**	**Promote**	**Utilities**
@@FLM$CD	Change Code Date				X
@@FLM$CM	Compool				X
@@FLM$CT	Change Code Time				X
@@FLM$IN	Include				X
@@FLM$UD	User Data Entry				X
@@FLM$XN	Compilation Unit Name				X
@@FLM$XT	Compilation Unit Type				X
Note: The build function does not support the use of SCLM variables on the FLMCPYLB macro.					

Chapter 4, "SCLM Dialog Interface," defines and lists the SCLM fields (as they are displayed in the dialog) for each record that is stored in the project database.

You can use the variables with the following:

- The FLMTRNSL OPTIONS parameter
- The PARM and PARM*x* architecture member keywords
- The COPYLIB parameter
- The line format parameter of the database contents utility
- Build and promote user exits.

Chapter 4. SCLM Dialog Interface

This chapter describes the panels you use to access the SCLM functions and the various options you can select from each panel. It also describes the panels that allow you to generate reports and provides several examples of the reports.

This chapter also compares SCLM to ISPF/PDF and notes the differences in the edit commands and the similarities of the utilities.

You can access all SCLM functions interactively through a set of panels under ISPF/PDF dialog management by selecting Option 10 from the ISPF/PDF Primary Option Menu.

Note: A virtual region size of 4096K is recommended when you use the SCLM dialog. Increase the virtual region size if you encounter GETMAIN problems.

SCLM Primary Option Menu

Select the six SCLM primary functions from the SCLM Primary Option Menu shown in Figure 6.

```
------------------------ SCLM PRIMARY OPTION MENU -------------------------
OPTION  ===>

   1  BROWSE      - ISPF/PDF Browse
   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 up SCLM hierarchy
   X  EXIT        - Terminate SCLM

SPECIFY SCLM PROJECT CONTROL INFORMATION:
   PROJECT    ===> PROJ1    (Project high-level qualifier)
   ALTERNATE  ===>          (Project definition: defaults to project)
   DEV GROUP  ===> USER1    (Development group: defaults to user ID)
```

Figure 6. SCLM Primary Option Menu

© Copyright IBM Corp. 1989, 1990

When you select one of these options and press the Enter key, SCLM displays another panel that is determined by the option you selected. Figure 6 on page 43 shows the options that this chapter describes. You can use the options to:

Option	Description
1 Browse	Display data without changing it and see large data sets, such as compiler listings. You can scroll browse displays up, down, left, or right. Browse commands, entered on the COMMAND line, allow you to do tasks like finding a character string. See "Browse (Option 1)" on page 45 for more information.
2 Edit	Create or change source data, such as program code and documentation. SCLM uses the ISPF/PDF editor, which is a full-screen editor. Unlike Browse, Edit allows you to type over the data displayed on your screen. You can scroll the data up, down, left, or right. You can change the data by using the edit *line commands*, which are entered directly on the line number of the line or lines to be affected, and by using *primary commands*, which are entered on the COMMAND line. See "Edit (Option 2)" on page 46 for more information.
3 Utilities	Carry out library and data set maintenance tasks, such as browsing or deleting members, accounting records, build maps, and intermediate records and forms; updating member authorization codes; migrating project databases to SCLM; and creating database contents and architecture reports. See "Utilities (Option 3)" on page 52 for more information.
4 Build	Build data set members or components of an application, automatically compiling and linking modules that require processing. See "Build (Option 4)" on page 91 for more information.
5 Promote	Promote data set members or components of an application. See "Promote (Option 5)" on page 94 for more information.

The fields on the SCLM Primary Option Menu are:

PROJECT
The common identifier for all ISPF libraries belonging to the same programming project. This field is required to access any SCLM function.

ALTERNATE
You can enter an alternate project definition. Leaving this field blank results in the project definition being the same as the project high level qualifier. For more information, see "Primary Non-Key Group Testing Techniques" on page 258 for alternate project definition.

DEV GROUP
A group at the bottom of the SCLM hierarchy. Your private library is in this group. This field defaults to your TSO PREFIX or to your user ID if no TSO PREFIX has been created.

Browse (Option 1)

The Browse option allows you to display data in a project database. The SCLM browse interface analyzes the database 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 DEV GROUP specified on the Primary Option Menu.

SCLM browse is functionally equivalent to ISPF/PDF browse. (Refer to *ISPF/PDF Guide* 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 browse a non-SCLM data set, a partitioned data set (PDS), or a partitioned data set extended (PDSE). Figure 7 shows the panel SCLM displays when you select Option 1 BROWSE from the SCLM Primary Option Menu.

```
------------------------  SCLM BROWSE - ENTRY PANEL  ------------------------
COMMAND ===>

ISPF LIBRARY:
   PROJECT ===> PROJ1
   GROUP   ===> USER1     ===> INT       ===> TEST      ===> RELEASE
   TYPE    ===> SOURCE
   MEMBER  ===>                (Blank or pattern for member selection list)

OTHER PARTITIONED OR SEQUENTIAL DATA SET:
   DATA SET NAME  ===>
   VOLUME SERIAL  ===>          (If not cataloged)

DATA SET PASSWORD ===>          (If password protected)

MIXED MODE        ===> NO       (Specify YES or NO)

FORMAT NAME       ===>
```

Figure 7. SCLM Browse - Entry Panel

The fields on the SCLM Browse - Entry panel are:

PROJECT
The project that you specified on the SCLM Primary Option Menu.

GROUP
The development group that you specified in the DEV GROUP field on the SCLM Primary Option Menu. The default is your user ID. This group is followed by the next key group in the hierarchy for up to four groups.

TYPE
The identifier for the type of information in the ISPF library, such as PL/I, SCRIPT, and PANELS.

MEMBER
The name of an ISPF/PDF library or other partitioned data set member. Leaving this field blank or typing a character string followed by an asterisk causes SCLM to display a member list.

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 quotes. However, if you omit the TSO user prefix and single quotes, your TSO user prefix is automatically added to the beginning of the data set name.

VOLUME SERIAL
A real DASD volume or a virtual volume residing on an IBM 3850 Mass Storage System. To access 3850 virtual volumes, you must also have MOUNT authority, which is acquired through the TSO ACCOUNT command. ISPF/PDF does not allow the use of data sets that contain more than one volume. SCLM does not use the system catalog when you specify a volume serial.

DATA SET PASSWORD
The password for OS password-protected data sets. This is not your TSO user ID password.

MIXED MODE
You can browse unformatted mixed data that contains both EBCDIC (one-byte) characters and Double Byte Character Set (DBCS or two-byte) characters. To do this, you must specify mixed mode. Valid values for this field are:

YES Indicates mixed data

NO Indicates no mixed data.

If your terminal does not support DBCS, ISPF/PDF ignores the operation mode.

FORMAT NAME
The name of a format definition or blank if no format is to be 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, ISPF/PDF ignores the operation mode.

Edit (Option 2)

The Edit option of SCLM is provided by the ISPF/PDF editor. Within SCLM, the editor automatically locks the member when you begin the edit session and parses and stores edited members and their accounting records when you end the edit session.

When you select the Edit option, the SCLM editor analyzes the database structure for the specified project and displays the concatenation sequence of the groups in your library concatenation. It presents the four lowest key groups for the project previously specified in the project definition. This preprocessing, coupled with the ISPF/PDF "drawdown" 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 private 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.

Figure 8 on page 47 shows the panel SCLM displays when you select Option 2 EDIT from the SCLM Primary Option Menu.

```
------------------------- SCLM EDIT - ENTRY PANEL --------------------------
COMMAND ===>

ISPF LIBRARY:
   PROJECT ===> PROJ1
   GROUP   ===> USER1     ===> INT       ===> TEST      ===> RELEASE
   TYPE    ===> SOURCE
   MEMBER  ===> MODULE5         (Blank or pattern for member selection list)

PROFILE NAME       ===>         (If blank, defaults to data set type)

INITIAL MACRO      ===>
MIXED MODE         ===> NO        (YES or NO)

CHANGE CODE        ===> 2

AUTHORIZATION CODE ===>             (If blank, default auth code used)
PARSER VOLUME      ===>             (If blank, default volume used)
```

Figure 8. SCLM Edit - Entry Panel

The fields on the SCLM Edit - Entry panel are:

PROJECT
The project that you specified on the SCLM Primary Option Menu.

GROUP
The development group that you specified in the DEV GROUP field on the SCLM Primary Option Menu. The default is your user ID. 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 private libraries by not allowing you to modify this field. SCLM guarantees that the group is a valid private library by verifying it against the specified project definition. (All other displayed groups are in unprotected fields and you can alter them.)

Also, if you specify an incorrect order for the drawdown of a given member (that is, the concatenation sequence of groups does not match the order of the key groups in the library concatenation), SCLM does not allow the edit session. SCLM then displays a panel indicating all groups that comprise the complete hierarchy.

TYPE
The identifier for the type of information in the ISPF library, such as PL/I, SCRIPT, and PANELS.

MEMBER
The name of an ISPF library or other partitioned data set member. Leaving this field blank or typing a pattern as a member name causes SCLM to display a member list.

PROFILE NAME
The name of an edit profile, which you can use to override the default edit profile. Refer to *ISPF/PDF Edit and Edit Macros* for more information.

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. Refer to *ISPF/PDF Edit and Edit Macros* for more information.

MIXED MODE
You can edit unformatted mixed data that contains both EBCDIC (one-byte) characters and Double Byte Character Set (DBCS or two-byte) characters. To do this, you must specify mixed mode. Valid values for this field are:

YES Specifies that the editor is to look for shift-out and shift-in delimiters surrounding DBCS data.

NO No mixed data.

If your terminal does not support DBCS, SCLM ignores the operation mode.

CHANGE CODE
Specify a change code to indicate why you updated the member.

AUTHORIZATION CODE
Specify the current authorization code for the member.

PARSER VOLUME
The specific volume ID in which SCLM stores output from the SCLM parser. This field is not required.

The SCLM editor provides all of the functions of the ISPF/PDF editor. For example, you can specify a profile name and an initial macro before editing a member. Enhancements now allow you to lock a member; to parse, create, or update an accounting record; and to specify change codes.

The parser supplied with SCLM does not recognize ISPF/PDF packed data. If the ISPF/PDF 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.

The following paragraphs describe how additional features of the SCLM editor differ from the ISPF/PDF editor.

SAVE

The SCLM SAVE command is similar to the ISPF/PDF SAVE command except that the member is automatically parsed and the member's accounting record is created or updated.

The first time you save a member, SCLM displays the SCLM Edit Profile panel (see Figure 9 on page 50) for you to specify a change code and the member's language.

The SCLM editor supports two modes of operation, UPARSE and USUBDD, that allow you to force save an Ada language member. Each of these modes allows an Ada language member to be parsed or drawn down, or both, even when it contains a compilation unit that already exists in another member at a higher group in the hierarchy.

If you specify the UPARSE mode, SCLM parses the member and stores the accounting information for that member.

If you specify the USUBDD mode, SCLM allows the compilation unit to be drawn down to your group. You can specify either one or both modes.

Note: Be careful when you use these options to save an Ada member because doing so can cause SCLM to track a compilation unit defined in two different members. A forced save for a non-Ada language member has the same effect as a save.

Command Format

```
SAVE      [UPARSE] [USUBDD]
```

SCREATE

The SCLM SCREATE command is similar to the ISPF/PDF CREATE command except that the SCLM editor automatically parses, locks out, and creates an accounting record for the created member.

If you do not enter a change code on the SCLM Edit - Entry panel (and it is required), SCLM displays the SCLM Edit Profile panel shown in Figure 9 on page 50. 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. SCLM displays the SCLM Edit Profile panel so you can specify another language. Otherwise, the newly-created member has the same member attributes as the current member.

Note: If the member to be created already exists in your library, SCLM automatically defaults to the SREPLACE command.

The SCLM SCREATE command does not offer an extended panel for creating a member outside the hierarchy.

Command Format

```
SCREATE   member-name  [line-range]

SCRE
```

SMOVE

The SCLM SMOVE command is similar to the ISPF/PDF MOVE command except that the SCLM editor deletes the accounting information of the member being moved if the member is moved from a development library.

The SCLM SMOVE command does not offer an extended panel for moving a member from outside the hierarchy.

Command Format

```
SMOVE     member-name  [AFTER label]
                       [BEFORE label]
```

SPROF

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 9, when you end the edit session (if you did not enter a change code on the SCLM Edit - Entry panel and it is required) or whenever you enter the SPROF command. SCLM also displays the SCLM Edit Profile panel to specify a language for a new member.

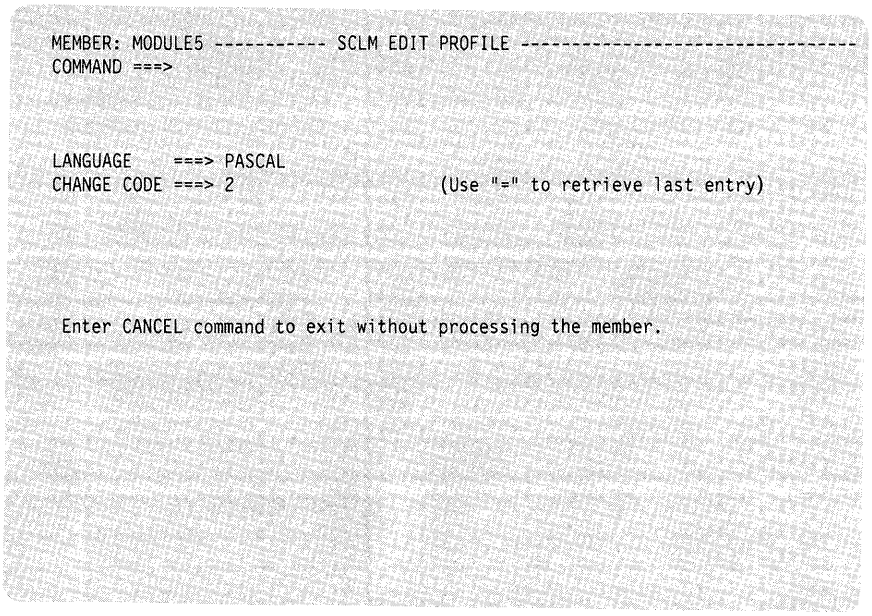


Figure 9. SCLM Edit Profile

The fields on the SCLM Edit Profile panel are:

LANGUAGE
The language definition name to be used to process the member. This field is required.

CHANGE CODE
Specify a change code to indicate why you updated the member. This field is required if your project has a change code verification routine. See "Change Code Verification Routines" on page 261 for more details.

You can change the information on this panel at any time during the edit session. If you alter the LANGUAGE field or modify the member, or both, SCLM parses and creates or updates the member's accounting record while saving the member.

SCLM processes the member and saves it in your private library if you alter the language or change code and if the member does not exist in your private library. If you alter the change code but do not modify the member and it exists in the private library, SCLM regenerates only the accounting information.

When you enter SCLM edit profile information, SCLM maintains it across SCLM edit sessions. 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

The SCLM SREPLACE command is similar to the ISPF/PDF 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 library.

If you do not enter a change code on the SCLM Edit - Entry panel (and it is required), SCLM displays the SCLM Edit Profile panel shown in Figure 9 on page 50. Also, the replaced member will have the same member attributes as the current member.

The SCLM SREPLACE command does not offer an extended panel for replacing a member outside the hierarchy.

Command Format

```
SREPLACE  member-name  [line-range]

SREPL
```

Because the SCLM editor uses ISPF/PDF edit macros to perform its functions, do not override SCLM command macro definitions, especially the END, SAVE, CANCEL, and RETURN macros. If you need a user-defined end macro, define an alternate command name such as QUIT. At the end of this alternate end macro, you must enter the END, RETURN, SAVE, or CANCEL command to start the SCLM end routines.

If you override an SCLM macro by using the DEFINE command, the macro is not redefined until you begin a new edit session.

You can also override SCLM edit macros by entering the ISPF/PDF BUILTIN command (for example, BUILTIN SAVE).

Note: Be careful if you override SCLM command macros. If you call SCREATE or SREPLACE as BUILTIN, for instance, SCLM does not automatically parse, lock, and update accounting records for the created member.

Utilities (Option 3)

Figure 10 shows the panel SCLM displays when you select Option 3 UTILITIES from the SCLM Primary Option Menu.

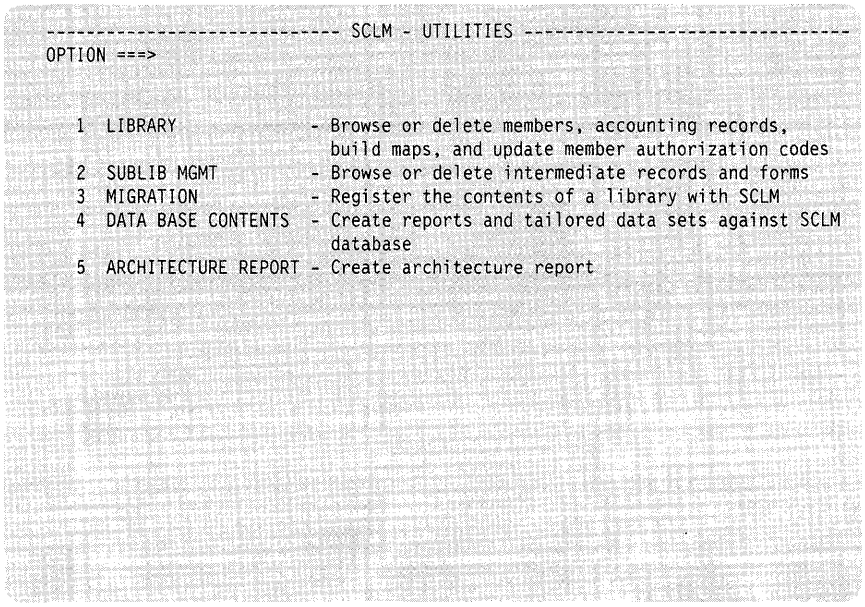


Figure 10. SCLM Utilities

When you select one of these options and press the Enter key, SCLM displays another panel that is determined by the option you selected. Figure 10 shows the following options that you can use to:

Option	Description
1	Browse or delete source members and their accounting records and build maps. You can also update authorization codes and browse statistical information, such as the number of change codes, includes, compools, compilation units, and user entries for an accounting record.
2	Browse or delete intermediate records or forms for Ada members.
3	Migrate a large number of members into a project database.
4	Create reports and tailored data sets on the contents of a project hierarchy.
5	Create reports that show the architecture of an application or a subapplication.

Library Utility

The library utility is completely interactive and parallels the ISPF/PDF library utility.

Figure 11 on page 53 shows the panel SCLM displays when you select Option 1 LIBRARY from the SCLM Utilities panel.

```
------------------------- SCLM - LIBRARY UTILITY --------------------------
COMMAND ===>

   A - Browse accounting record
   B - Browse member
   D - Delete member, accounting record, build map, and cross reference records
   M - Browse build map
   U - Update accounting record authorization code
   blank - Display member list

SCLM LIBRARY:
   PROJECT ===> PROJ1
   GROUP   ===> USER1
   TYPE    ===> SOURCE
   MEMBER  ===>
```

Figure 11. SCLM Library Utility

The fields on the SCLM Library Utility panel are:

PROJECT
The project that you specified on the SCLM Primary Option Menu.

GROUP
The development group that you specified in the DEV GROUP field on the SCLM Primary Option Menu.

TYPE
The identifier for the type of information in the ISPF/PDF library. If you specify an invalid SCLM type, enter the HELP command (the default is PF1) to display all valid types for the current project definition.

MEMBER
The name of an ISPF/PDF library or other partitioned data set member. Leaving this field and the COMMAND field blank causes SCLM to display a member list.

Library Utility Commands

Enter your selection in the COMMAND field.

- If you enter A, B, or M, SCLM displays the specified member or record if it is present.
- If you enter D, SCLM deletes all portions of the member such as text, accounting, and build map records.

  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 need to delete the member from the non-key group.
- If you enter U, SCLM displays an input panel and updates the authorization code according to your input. (To delete or update any data, you must have UPDATE authority to the specified data set.)
- To delete, browse, or update several members, use the member selection list.

Member Selection List

You can delete, browse, or update members by making selections from a member selection list. To display a member selection list, do the following:

1. Leave the COMMAND field blank.
2. Enter the project, group, and type information in the appropriate fields.
3. Leave the MEMBER field blank.

Use the scroll commands or the LOCATE command to scroll the list.

Figure 12 shows the panel SCLM displays when you select the member selection list.

```
DATA SET -- PROJ1.USER1.SOURCE ---------------------------------  ROW 1 OF 6
COMMAND ===> _                                                  SCROLL ===> CSR

  Line Commands:
    Browse :  A - Accounting   B - Text   M - Build map
    Alter  :  D - Delete text, accounting, build map   U - Update auth code

    Member     Status    Text     Accounting   Build Map
    ------     ------    ----     ----------   ---------
    MODULE1                X          X           X
    MODULE2                X          X
    MODULE3                X          X           X
    MODULE4                X          X           X
A   MODULE5                X          X           X
    MODULE6                X          X           X
******************************* BOTTOM OF DATA ********************************
```

Figure 12. Member Selection List

The fields for the panel shown in Figure 12 are:

MEMBER
The names of the members in the project, group, and type you specified on the SCLM Library Utility panel.

STATUS
SCLM displays the status of the member according to the line command you select. Enter a line command to do the following:

A Display an accounting record
B Browse a member
D Delete a member (see Note)
M Display a build map record
U Update an authorization code.

To delete the accounting records for a member that you deleted outside the SCLM dialog, enter the name of the member in the member selection list. Then enter the D line command.

Note: SCLM can only delete accounting records and build maps from libraries that you can allocate; that is, the libraries these records are from must exist.

TEXT
An X in this field indicates that the member exists.

ACCOUNTING
An X in this field indicates that the accounting information for the associated member exists.

BUILD MAP
An X in this field indicates that the build map record for the associated member exists.

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 13.

```
DATA SET -- PROJ1.USER1.SOURCE(MODULE5) - ACCOUNTING RECORD ------------------
COMMAND ===>

GENERAL DATA:
  Accounting Status  : EDITABLE        Change Group          : USER1
  Change User ID     : VEND107         Authorization Code    : REL
  Member Version     : 3               Auth. Code Change     :
  Language           : PASCAL          Translator Version    :
  Creation Date      : 01/31/89        Change Date           : 02/14/89
  Creation Time      : 12:45:33        Change Time           : 17:03:00
  Promote User ID    :                 Access Key            :
  Promote Date       : 00/00/00        Build Map Name        :
  Promote Time       : 00:00:00        Build Map Type        :
  Predecessor Date   : 02/01/89        Build Map Date        : 02/14/89
  Predecessor Time   : 14:21:00        Build Map Time        : 17:03:00

   _   Display Statistics                          (Enter "S" to select)
   _   Number of Change Codes      : 5             (Enter "S" to select)
   _   Number of Includes          : 1             (Enter "S" to select)
   _   Number of Compools          : 0             (Enter "S" to select)
   _   Number of Compilation Units : 0             (Enter "S" to select)
   _   Number of User Entries      : 2             (Enter "S" to select)
```

Figure 13. Accounting Record

The fields on the Accounting Record panel are:

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 you could edit if they were not locked out.
INITIAL	Members that you could edit if SCLM were not verifying whether they are locked out.

CHANGE USER ID
The user ID of the person who made the last update to the member.

MEMBER VERSION
The number of times the member was drawn down. (A version of 1 is used for new members.)

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 temporary authorization code used during verification. When set, it represents the "change to" authorization code.

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 that indicates who has exclusive access to the member.

BUILD MAP NAME
The name of the map that created the member.

BUILD MAP TYPE
The name of the type containing the map.

BUILD MAP DATE
The date the map created the member.

BUILD MAP TIME
The time the map created the member.

DISPLAY STATISTICS
SCLM displays the Accouting Record Statistics panel, shown in Figure 14 on page 57, if you enter S in this field.

NUMBER OF CHANGE CODES
The number of change codes entered against the member.

NUMBER OF INCLUDES
The number of include references in the source member.

NUMBER OF COMPOOLS
The number of JOVIAL compool references in the member.

NUMBER OF COMPILATION UNITS
The number of compilation units in the member.

NUMBER OF USER ENTRIES
The number of user data entry records associated with the member.

Type S in the appropriate input fields and press the Enter key to display additional panels. You can browse the statistics or lists of change codes, includes, JOVIAL compools, compilation units, or user entries referenced by a member. You can also scroll the lists.

Figure 14 through Figure 18 show the panels SCLM displays when you select each of the items in the Accounting Record panel.

Statistics

SCLM displays statistical information, as shown in Figure 14, when you enter S in the DISPLAY STATISTICS field on the Accounting Record panel.

```
DATA SET -- PROJ1.USER1.SOURCE(MODULE5) - ACCOUNTING RECORD ------------------
COMMAND ===>


STATISTICS:
  Total Lines       : 13            Total Statements       : 4
  Comment Lines     : 2             Comment Statements     : 2
  Noncomment Lines  : 5             Control Statements     : 0
  Blank Lines       : 6             Assignment Statements  : 0
  Prolog Lines      : 0             Noncomment Statements  : 2
```

Figure 14. Accounting Record Statistics

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.

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.

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.

Change Code List: Figure 15 is an example of the information SCLM displays when you enter S in the NUMBER OF CHANGE CODES field on the Accounting Record panel.

```
DATA SET -- PROJ1.USER1.SOURCE(MODULE5) - CHANGE CODE LIST -------  ROW 1 OF 5
COMMAND ===>                                                  SCROLL ===> CSR


      Line Command:      D - Delete change code

Delete  Status   Change Number    Change Date    Change Time
------  ------   -------------    -----------    -----------
        *SELECT  31               02/14/89       17:03:00
                 2                02/14/89       17:00:00
                 PR3573           02/01/89       14:21:00
                 CR3582           02/01/89       11:34:00
                 PR3456           02/01/89       11:31:00
****************************** BOTTOM OF DATA ********************************
```

Figure 15. Change Code List

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 NUMBER
A change code assigned to indicate why a member was updated.

CHANGE DATE
The last date a developer modified the member for the associated change number. 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.

Include List: Figure 16 is an example of the information SCLM displays when you enter S in the NUMBER OF INCLUDES field on the Accounting Record panel.

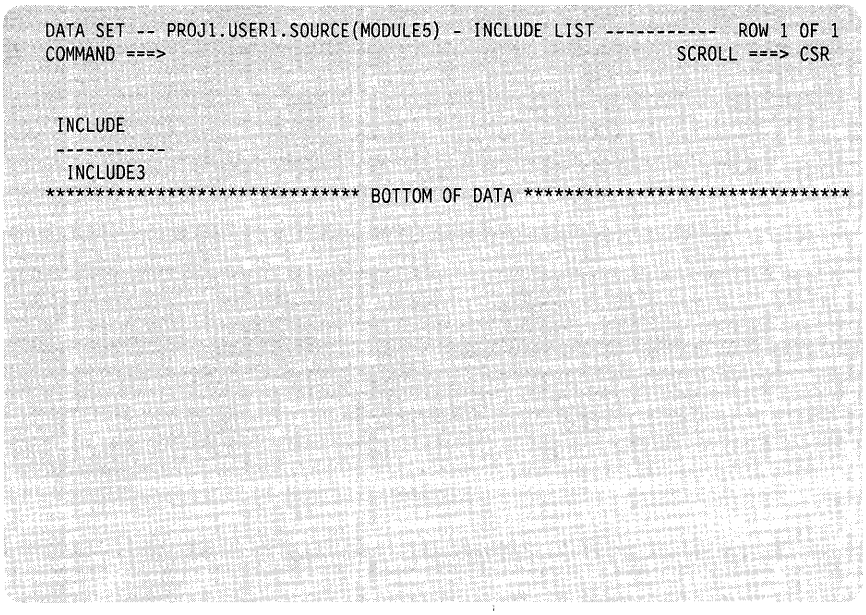


Figure 16. Include List

The field on the Include List panel is:

INCLUDE
The name of an include reference in the source member. An include reference is a generic term for code that you insert when SCLM compiles the source member. The syntax of an include statement in a program is language-dependent and is defined by language syntax rules.

Compool List: Figure 17 is an example of the information SCLM displays when you enter S in the NUMBER OF COMPOOLS field on the Accounting Record panel.

```
DATA SET -- PROJ1.USER1.SOURCE(MODULE4) - COMPOOL LIST ------------------------
COMMAND ===>                                                  SCROLL ===> CSR


 COMPOOL
 -----------
 COMP1
 COMP2
****************************** BOTTOM OF DATA ********************************
```

Figure 17. Compool List

The field on the Compool List panel is:

COMPOOL
The name of a compool reference in the source member. A compool reference is a reference to a JOVIAL data mapping structure that SCLM must compile before it compiles the current member. Compool references are specific to the JOVIAL languages.

Compilation Units: Figure 18 is an example of the information SCLM displays when you enter S in the NUMBER OF COMPILATION UNITS field on the Accounting Record panel.

```
DATA SET -- PROJ1.USER1.SOURCE(MODULE3) - COMPILATION UNITS ------------------
COMMAND ===>                                                  SCROLL ===> CSR


      Line Command:     S - Select cross reference record for review


        Compilation
Select      Type       Compilation Unit Name
------      ----       -------------------------------------------------------
            BODY       XPKG2
            SPEC       XPKG2
****************************** BOTTOM OF DATA ********************************
```

Figure 18. Compilation Units

The fields on the Compilation Units panel are:

SELECT
SCLM displays the contents of the cross-reference record for the selected compilation unit when you enter S in this field.

COMPILATION TYPE
The type of the compilation unit:

SPEC Specification of the compilation unit
BODY Body of the compilation unit
XREF A record SCLM creates for dependency tracking of main procedures.

COMPILATION UNIT NAME
The name of the compilation unit. A *compilation unit* is an Ada language entity that compiles separately.

SCLM considers each compilation unit contained in an Ada source member to be an entity. The PARSE service obtains dependency information for each compilation unit. The compilation unit names are not necessarily member names.

Cross-Reference Record: Figure 19 is an example of the information SCLM displays when you enter S in the SEL field on the Compilation Units panel.

```
DATA SET -- () - CROSS-REFERENCE RECORD -------------------------------------
COMMAND ===>                                                  SCROLL ===>

 Compilation Unit : XPKG2

 Compilation Type   : BODY        Authorization Code : TEST
 CU Qualifier       : ADACODE     Generic Flag       : GENERIC
 Accounting Member  : MODULE1     Change Date        : 11/22/88
 Accounting Type    : SOURCE      Change Time        : 11:44:32

                     Dependency Information
 Depend-
 ency   Compilation
 Type   Type   Dependency Name
 ----   ----   ----------------------------------------
  UP    BODY   XYZPKG
 DOWN   SPEC   ABCPKG
```

Figure 19. Cross-Reference Record

The fields on the Cross-Reference Record panel are:

COMPILATION UNIT
The name of the compilation unit.

COMPILATION TYPE
The type of the compilation unit.

CU QUALIFIER
The name of the compilation unit (CU) qualifier specified in the language definition.

ACCOUNTING MEMBER
The member that generated this cross-reference record.

ACCOUNTING TYPE
The type containing the source that generated this cross-reference record.

AUTHORIZATION CODE
The current authorization code for the cross-reference record.

GENERIC FLAG
A flag indicating whether this compilation unit contains an Ada generic or an inline construct.

CHANGE DATE
The last date a developer modified the cross-reference record.

CHANGE TIME
The last time a developer modified the cross-reference record.

DEPENDENCY TYPE
The type of dependency the current compilation unit has: UP for upward dependency and DOWN for downward dependency.

- Upward dependency

  A compilation unit has an upward dependency on the units it references using the WITH and IS SEPARATE language structures. It is a package or procedure body that has an upward dependency on its specification. An upward dependency member is processed *before* a given member.

- Downward dependency

  A compilation unit has a downward dependency on the units it references with the IS SEPARATE language structure. It is a package or procedure specification that has a downward dependency on its body. A downward dependency member is processed *after* a given member.

COMPILATION TYPE
The type of the compilation unit:

SPEC Specification of the compilation unit
BODY Body of the compilation unit
XREF A record SCLM creates for dependency tracking of main procedures.

DEPENDENCY NAME
The name of a compilation unit on which this compilation unit has a dependency.

User Data Entries: Figure 20 is an example of the information SCLM displays when you enter S in the NUMBER OF USER ENTRIES field on the Accounting Record panel.

```
DATA SET -- PROJ1.USER1.SOURCE(MODULE5) - USER DATA ENTRIES ------------------
COMMAND ===>                                                 SCROLL ===> CSR

      Line Command:      D - Delete User Data Entry

Del Stat Rec# User Data Entry
--- ---- ---- ------------------------------------------------------------
         1    This record is very long to prove that two lines can be shown in
              one record.
         2    This record is short.
****************************** BOTTOM OF DATA ********************************
```

Figure 20. User Data Entries

The fields on the User Data Entries panel are:

DEL
You specify that you want to delete the user data entry record when you enter 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. 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 21, displays the fixed build map information SCLM records for a member.

```
DATA SET -- PROJ1.USER1.SOURCE(MODULE5) - BUILD MAP RECORD --------------------
COMMAND ===>

GENERAL DATA:
  Change User ID  : VEND107            Change Group     : USER1
  Member Version  : 5                  Change Date      : 02/14/89
  Language        : CCMAP              Change Time      : 17:10:57
  Creation Date   : 01/31/89           Promote Date     : 00/00/00
  Creation Time   : 14:57:16           Promote Time     : 00:00:00
                                       Promote User ID  :

  Translator Version   :               Build Map Date   : 02/14/89
  Language Version     : 1.0           Build Map Time   : 17:10:57
  Build Map Name       : MODULE5
  Build Map Type       : SOURCE

_ Review Build Map Contents            (Enter "S" to select)
```

Figure 21. Build Map Record

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 the member was drawn down. (A version of 1 is used for new members.)

LANGUAGE
The language of the member.

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 a developer modified the member.

CHANGE TIME
The last time a developer modified the member.

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 map that created the member.

BUILD MAP TYPE
The name of the type containing the map.

BUILD MAP DATE
The date the map created the member.

BUILD MAP TIME
The time the map created the member.

REVIEW BUILD MAP CONTENTS
SCLM displays the Build Map Contents panel, shown in Figure 22 on page 67, when you enter S in this field.

Build Map Contents

SCLM displays the build map contents in a browse data set. Figure 22 shows the contents of a build map record for an architecture defined in a CC architecture member.

```
BROWSE -- PROJ1.USER1.MODULE5 -------------------- LINE 00000000 COL 001 080
COMMAND ===>                                              SCROLL ===> CSR

******************************** TOP OF DATA ********************************
                              BUILD MAP CONTENTS
                              ------------------

Keyword  Member   Type     Last Time Modified Version
-------  -------  ----     ------------------ -------
OBJ      MODULE5  OBJ      02/14/89  17:10:57 5
LIST     MODULE5  LIST     02/14/89  17:10:57 5
I1*      INCLUDE3 SOURCE2  02/14/89  16:50:00 2
SINC     MODULE5  SOURCE   02/14/89  17:03:00 3

* INTERNAL KEYWORDS
  I#     - INCLUDED MEMBER REFERENCED BY SINC MEMBER, # = IMBEDDED GROUP
******************************** BOTTOM OF DATA ********************************
```

Figure 22. Build Map Contents

The fields on the Build Map Contents panel are:

KEYWORD
You can use certain keywords to identify architecture information. See "Architecture Statements" on page 27 for more details.

The architecture member example contains three keywords: OBJ, LIST, and SINC. The actual parameters from the architecture member (prior to substitution) are kept for PARM and PARM*x* keywords. Keywords denoted with an asterisk (*) are include references found in source member MODULE5.

MEMBER
The name of the source member referenced in the architecture member.

TYPE
The name of the type containing the source member.

LAST TIME MODIFIED
The last time SCLM parsed and stored the specified member. For SCLM-generated code, that is, OBJ and LIST, it is the last time SCLM generated the member.

VERSION
The number of times SCLM parsed and stored the member. SCLM has parsed and stored source member MODULE5 only once but has generated its corresponding object module and listing two times.

INTERNAL KEYWORDS
Keywords that SCLM uses to track references. The internal keyword I# indicates the group in which the members were first referenced.

Authorization Code Update

Enter U on the Library Utility panel or the member selection list to display the Authorization Code Update panel. Figure 23 shows the panel SCLM displays for you to update the authorization code for a member.

```
DATA SET -- PROJ1.USER1.SOURCE(MODULE5) - AUTH CODE UPDATE -------------------
COMMAND ===>


 Member to be updated: PROJ1.USER1.SOURCE(MODULE5)

 OLD AUTHORIZATION CODE ===> REL
 NEW AUTHORIZATION CODE ===>

Instructions:
 Press ENTER key to confirm the update request.

 Enter END command to cancel the update request.
```

Figure 23. Authorization Code Update

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 the END command to cancel the update request.

Ada Sublibrary Management Utility

Use the Ada sublibrary management utility to delete or browse Ada intermediate records and intermediate forms for compilation units. Ada intermediate records are accounting records that SCLM tracks for the Ada intermediate form of compilation units. The build function creates these records after a successful compile.

To delete intermediate records and forms for compilation units, you must have UPDATE authority to the specified source member data set. SCLM deletes the intermediate form by starting Ada compiler utility programs. The utility is completely interactive.

Figure 24 on page 69 shows the panel SCLM displays when you select Option 2 SUBLIB MGMT from the SCLM Utilities panel.

```
------------------------ SUBLIBRARY MANAGEMENT UTILITY -----------------------
COMMAND ===>

ADA DATABASE:
   CU (compilation unit) QUALIFIER  ===>


ISPF LIBRARY:
   PROJECT  ===> PROJ1
   GROUP    ===> USER1
   TYPE     ===>            ("*" for all types )


  Press ENTER key to browse or delete Ada intermediate records and forms.
```

Figure 24. Sublibrary Management Utility

The fields on the Sublibrary Management Utility panel are:

CU QUALIFIER
The name of the compilation unit qualifier specified in the language definition. SCLM uses it to distinguish between different Ada languages when searching for compilation unit dependencies.

PROJECT
The project that you specified on the SCLM Primary Option Menu.

GROUP
The private library that you specified on the SCLM Primary Option Menu. The default is your user ID.

TYPE
The name of the type you want processed.

Member Selection List

You can delete or browse intermediate records and forms for compilation units by making selections from a member selection list. To display a list of the Ada intermediate records, enter the following:

1. The Ada database in the CU QUALIFIER field.
2. The appropriate group in the GROUP field.
3. The appropriate type in the TYPE field or an asterisk (*) for all types in a given group.

SCLM displays the name of the compilation unit and its type on each line of the member selection list, as shown in Figure 25 on page 70.

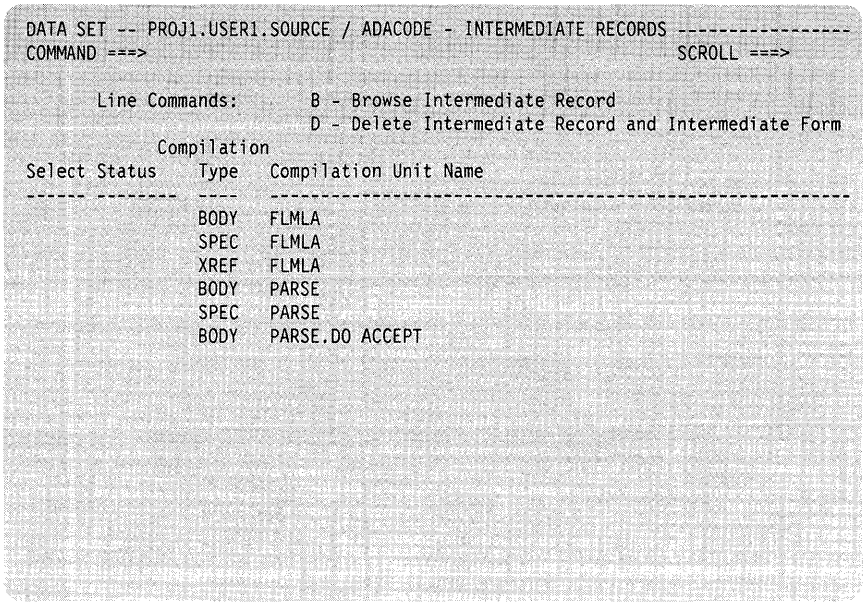


Figure 25. Member Selection List

The fields on the Member Selection List panel are:

SELECT
SCLM selects one or more Ada intermediate records and forms for processing when you enter line commands D (for delete) or B (for browse) in this field for the compilation units you want.

Figure 26 on page 71 shows the panel SCLM displays when you enter the B line command.

STATUS
SCLM displays the delete selection status in this field if you entered D in the SELECT field:

*DELETED Indicates the compilation unit you want to delete.

Enter the END command to confirm the delete request.

*ERROR SCLM cannot delete the selected compilation unit intermediate record or form because an error occurred.

SCLM records detailed error information in a temporary data set. Enter the HELP command (the default is the PF1 key) to obtain the name of this data set.

COMPILATION TYPE
The type of the compilation unit:

SPEC Specification of the compilation unit
BODY Body of the compilation unit
XREF A record SCLM creates for dependency tracking of main procedures.

COMPILATION UNIT NAME
The name of the compilation unit.

Intermediate Record

SCLM displays the contents of the intermediate record for the selected compilation unit, shown in Figure 26, when you enter B in the SELECT field on the Intermediate Record Member Selection List panel. SCLM stores the accounting information for the compilation unit in an Ada sublibrary.

```
DATA SET -- PROJ1.USER1.SOURCE / ADACODE - INTERMEDIATE RECORDS ----------
COMMAND ===>

 Compilation Unit  : ADA@2

 Compilation Type  : BODY

HISTORY
 Change User ID : USER1
 Creation Date  : 01/12/77              Change Date        : 12/22/87
 Creation Time  : 12:44:50              Change Time        : 11:22:30

GENERAL INFORMATION
 Member Version       : 1
 Language             : ADA
 Translator Version   : 3
 Change Group         : USER1
 Map Name             : ADAMAP          Accounting Member  : ADA1
 Map Type             : SOURCE          Accounting Type    : SOURCE
```

Figure 26. Intermediate Records

The fields on the Intermediate Records panel are:

COMPILATION UNIT
The name of the compilation unit.

COMPILATION TYPE
The type of the compilation unit:

SPEC Specification of the compilation unit
BODY Body of the compilation unit
XREF A record created for dependency tracking of main procedures.

CHANGE USER ID
The user ID of the person who made the last update to the member.

CREATION DATE
The date a developer first registered the intermediate form with SCLM.

CREATION TIME
The time a developer first registered the intermediate form with SCLM.

CHANGE DATE
The last date a developer modified the intermediate form.

CHANGE TIME
The last time a developer modified the intermediate form.

MEMBER VERSION
The number of times the member was drawn down. (A version of 1 is used for new members.)

LANGUAGE
The language of the member.

TRANSLATOR VERSION
The version of the translator.

CHANGE GROUP
The name of the group in which the member was last updated.

MAP NAME
The name of the map that created the member.

MAP TYPE
The name of the type containing the map.

ACCOUNTING MEMBER
The member that generated this compilation unit.

ACCOUNTING TYPE
The type that generated this compilation unit.

Migration Utility

In addition to the SCLM editor, the migration utility allows you to indicate the members you want tracked. Use this utility to enter a large number of members into a project's database, such as during a conversion to SCLM. You can also use it to lock, parse, and create accounting records for members that were edited without using the SCLM edit function.

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 project's database. For a complete description of the lock, parse, and store process, see "Edit Function" on page 10.

Figure 27 shows the panel SCLM displays when you select Option 3 MIGRATION from the Utilities panel.

```
------------------------- SCLM MIGRATION UTILITY ----------------------------
COMMAND ===> EX                                          (EXECUTE or SUBMIT)

SELECTION CRITERIA:
  PROJECT ===> PROJ1
  GROUP   ===> USER1
  TYPE    ===> SOURCE
  MEMBER  ===> MODULE*                  (Pattern may be used)

MEMBER INFORMATION:
  AUTHORIZATION CODE     ===> REL
  CHANGE CODE            ===> 2
  LANGUAGE               ===> PASCAL

OUTPUT CONTROL:
  MESSAGES      ===> TERMINAL           (TERMINAL, PRINTER, DATASET, or NONE)
  LISTINGS      ===> NONE
  PRINTER       ===> H                  (Printer output class)
  VOLUME        ===>                    (If blank, the default volume is used)
 JOB STATEMENT INFORMATION:
  ===> //JOBNAME$ JOB (ACCOUNT,DEPT,BIN),'TSOUSERNAME',
  ===> // MSGCLASS=A,CLASS=A,NOTIFY=JOBNAME,
  ===> // USER=,GROUP=????????,PASSWORD=????????
  ===> //*
```

Figure 27. SCLM Migration Utility

To migrate a set of SCLM members, you must enter information for each field. The fields for the Migration Utility panel are:

PROJECT
The project that you specified on the SCLM Primary Option Menu.

GROUP
The private library that you specified on the SCLM Primary Option Menu. The default is your user ID.

TYPE
The name of the type you want processed.

MEMBER
The name of the member you want processed. You can use patterns for the member name. See "Specifying Selection Criteria" on page 75 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.

CHANGE CODE
The current change code entered against the member. To enter a different change code for the member, type over the displayed change code. SCLM verifies the code you entered before it processes the member. See "STORE" on page 12 for more information.

LANGUAGE
The language of the member. See "PARSE" on page 11 for a list of languages that SCLM supplies parsers for.

OUTPUT CONTROL
Specify destinations, such as TERMINAL, PRINTER, or DATASET, for the outputs. Also specify which printer output class you want to use and the volume on which SCLM should save data sets.

JOB STATEMENT INFORMATION
You can call the processing part of the migration utility from the interactive or batch environment. Enter the EXECUTE command if you want interactive processing, or enter the SUBMIT command if you want batch processing.

See Figure 46 on page 100 for a sample of the job statement information you must provide if you select batch processing.

Database Contents Utility

You can use the SCLM database contents utility to generate reports on the contents of a project hierarchy. You can define the format of the report, or you can use the default format. Database contents utility reports can contain build map or accounting information, or both, from a project database.

Figure 28 shows the panel SCLM displays when you select Option 4 DATABASE CONTENTS from the Utilities panel.

```
---------------------- SCLM DATABASE CONTENTS UTILITY -------------------------
COMMAND ===> EX                                          (EXECUTE or SUBMIT)

SELECTION CRITERIA:                     (Patterns may be used)
  PROJECT ===> PROJ1
  GROUP   ===> USER1      ===> INT       ===>           ===>
          ===>            ===>
  TYPE    ===> SOURC*
  MEMBER  ===> *

  CHANGE ADDITIONAL SELECTION CRITERIA  ===> YES            (YES or NO)

OUTPUT CONTROL:
  MESSAGES          ===> TERMINAL      (TERMINAL, PRINTER, DATASET, or NONE)
  REPORT            ===> DATASET
  TAILORED OUTPUT   ===> DATASET
  PRINTER           ===> H             (Printer output class)
  VOLUME            ===>               (If blank, the default volume is used)

 JOB STATEMENT INFORMATION:
  ===> //JOBNAME$ JOB (ACCOUNT,DEPT,BIN),'TSOUSERNAME',
  ===> // MSGCLASS=A,CLASS=A,NOTIFY=JOBNAME,
  ===> // USER=,GROUP=????????,PASSWORD=????????
  ===> //*
```

Figure 28. SCLM Database Contents Utility

You can use patterns for each of the SELECTION CRITERIA fields. The fields on the Database Contents Utility panel are:

PROJECT
The project that you specified on the SCLM Primary Option Menu.

GROUP
The groups that are to be reported.

TYPE
The name of the type you want processed.

MEMBER
The name of the member you want processed.

CHANGE ADDITIONAL SELECTION CRITERIA
Enter YES if you want to change the additional selection criteria; otherwise, enter NO. The panel shown in Figure 29 on page 76 appears if you enter YES.

OUTPUT CONTROL
Specify destinations for the outputs. If you enter TERMINAL, PRINTER, or DATASET in the TAILORED OUTPUT field, the panel shown in Figure 31 on page 79 appears.

Also specify which printer output class you want to use and the volume SCLM should save data sets on.

JOB STATEMENT INFORMATION

Enter the EXECUTE command if you want interactive processing, or enter the SUBMIT command if you want batch processing.

See Figure 46 on page 100 for a sample of the job statement information you must provide if you select batch processing.

Specifying Selection Criteria

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.

Note: Do not use an equal sign (=) as the first character in a pattern because it is a special character in ISPF/PDF.

Use the patterns shown in Table 6 to select accounting information.

Table 6. Pattern Examples

Pattern	Match
AB*Z	ABZ,ABCZ,ABCZYZ
¬AB*Z	ABC,XABZ,ABZX
*AB*Z	ABZ,XABZ,ABCABZ
USER1=	USER1,USER2,USER3
STAGE3=	STAGE1,STAGE2,STAGE3

Note: See Figure 1 on page 7 for an illustration of the hierarchy represented in the last two rows.

The portion of the project database that SCLM displays is determined by the parameters you specify.

The panel in Figure 29 on page 76 appears if you enter YES in the CHANGE ADDITIONAL SELECTION CRITERIA field on the Database Contents Utility panel.

If you enter NO, SCLM does not display the panel and the reports are generated with the values that already exist on the Additional Selection Criteria panel.

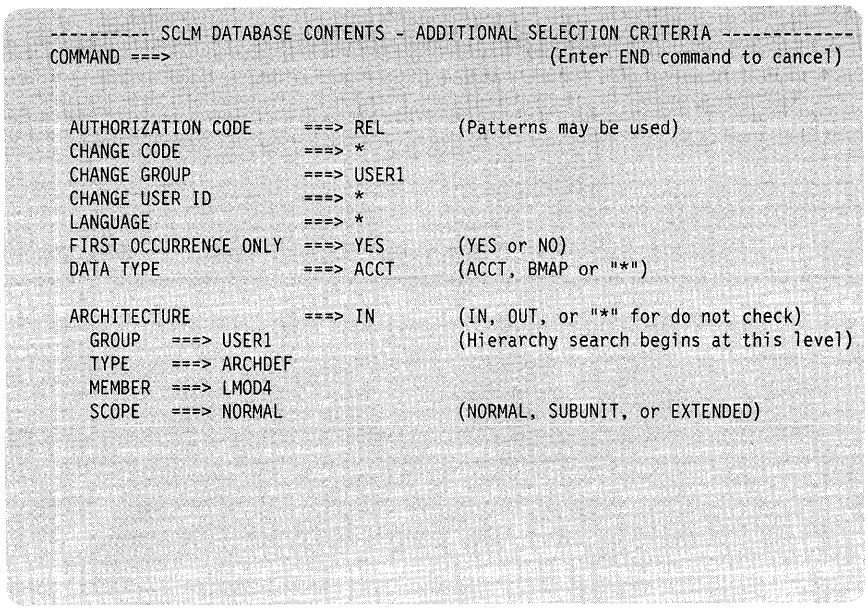


Figure 29. SCLM Database Contents - Additional Selection Criteria

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.

Use a blank value for the authorization code to select build outputs. Use a logical NOT symbol (¬) to select all members that can be edited. Build map information always contains a blank authorization code.

CHANGE CODE
Members that can be edited that were assigned a change code matching the change code pattern.

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. If a member has more than one change code, only one of the change codes must match the pattern for the member to be selected.

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.

FIRST OCCURRENCE ONLY
If you specify YES, 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.

This capability is particularly useful if you specify the groups in a hierarchy for the group patterns. The result is a member list for the hierarchy.

If you specify NO, SCLM selects all versions of all members.

DATA TYPE
Specify the following:

ACCT To report exclusively on accounting information.
BMAP To report exclusively on build map information.
* To report on build map and accounting information.

DATA TYPE is always required, but if it is left blank it defaults to **ACCT**.

Architecture Definition Fields

You can also 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 definition field. If you specify an architecture that has never been built, none of the members are 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
Specify the following:

IN To select members controlled by the architecture definition.

OUT To select members not controlled by the architecture definition.

* To indicate that an architecture definition is not used to identify selected members.

The following fields are required if you enter IN or OUT in the ARCHITECTURE field:

GROUP
The group identifying the lowest level in the hierarchy where SCLM should find the architecture definition.

TYPE
The type containing the architecture definition that controls the selected members.

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|EXTENDED
To select members that do have compilation unit dependencies.

See "Architecture Definition Fields" on page 77 for information on how scopes affect the architecture definition.

Note: You cannot use patterns in the GROUP, TYPE, and MEMBER fields on the Additional Selection Criteria panel.

The database contents report contains a list of all members that you select from the selection criteria. If you request a tailored data set, SCLM generates the data set from this list of accounting and build map information.

Figure 30 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   : LMOD4
SCOPE    : NORMAL

  DATE: 02/23/89      TIME: 11:26:18
```

Figure 30 (Part 1 of 2). Database Contents Utility Report

```
DATABASE CONTENTS REPORT                                        PAGE       2
---------------------------- TYPE: SOURCE -----------------------------------
MEMBER     GROUP1     GROUP2     GROUP3     GROUP4     GROUP5     GROUP6
--------   --------   --------   --------   --------   --------   --------
MODULE4    USER1
MODULE5               INT
MODULE6               INT
---------------------------- TYPE: SOURCE2 ----------------------------------
INCLUDE3              INT
```

Figure 30 (Part 2 of 2). Database Contents Utility Report

Note: An asterisk (*) next to the group name on the report indicates that the member represents build map information.

Tailored Output

If you want to tailor the database contents output, enter TERMINAL, PRINTER, or DATASET in the TAILORED OUTPUT field on the Database Contents Utility panel. SCLM displays the Customization Parameters panel, shown in Figure 31, which you use to generate the tailored report.

```
------------ SCLM DATABASE CONTENTS - CUSTOMIZATION PARAMETERS --------------
COMMAND ===>                                      (Enter END command to cancel)

  PAGE HEADERS  ===> YES              (YES or NO)
  SHOW TOTALS   ===> YES              (YES or NO)
  REPORT NAME   ===> SAMPLE REPORT

  REPORT LINE FORMAT:
  ===> @@FLMALT @@FLMGRP @@FLMTYP @@FLMMBR
```

Figure 31. SCLM Database Contents - Customization Parameters

The fields on the Customization Parameters panel are:

PAGE HEADERS
Enter YES 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.

SHOW TOTALS
Enter YES 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.

REPORT NAME
The title of the report in the tailored output. The maximum length is 35 characters.

REPORT LINE FORMAT
The format of a line of data in the tailored output. The line format can be up to 160 characters long.

If you use the SCLM @@FLM$XN or @@FLM$UD variables, keep in mind that their values can exceed eight characters. Place these variables at the end of the report line to ensure that the columns in the report line up evenly.

Press Enter to confirm these requests or enter the END command to cancel them.

Figure 32 shows an example of a tailored output. The title of the report is TESTREP. The report line format, specified as @@FLMMBR @@FLM$IN, causes the utility to generate output consisting of the members reported in the database contents report and their associated included members.

```
*TESTREP

*@@FLMMBR @@FLM$IN
*-------------------------------------------------------------------
 B0FREE   B0FREE1
          B0FREE2
 B0FREHIR B0FRE1
 B0SAVOUT B0SAV1
          B0SAV2
          B0SAV3
 B0HEX    B0HE1
 B0SAVE
 B0SAVOUT B0SAVOU1
```

Figure 32. Database Contents Tailored Data Set, Page 1

Tailored Output Examples

The report that appears in Figure 33 on page 81 is a formatted representation of the accounting and build map information you specified for the database contents report. The tailored output format specification consists of report variables and constant values. The report displays the report variables as headers over the lines of variable values. If multiple lines are output, it does not repeat constant values such as the member name.

Chapter 3, "SCLM Variables," provides a list of report variables.

```
                    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    : LMOD4
SCOPE     : NORMAL

     CUSTOMIZATION PARAMETERS
PAGE HEADERS : YES
SHOW TOTALS  : YES
REPORT NAME  : SAMPLE REPORT

  DATE: 02/15/89      TIME: 09:52:17
```

Figure 33 (Part 1 of 2). Database Contents Utility Tailored Report

```
                                                                    PAGE       2
SAMPLE REPORT

@@FLMALT @@FLMGRP @@FLMTYP @@FLMMBR
--------------------------------------------------------------------------------
PROJ1    USER1    SOURCE   MODULE4
PROJ1    INT      SOURCE   MODULE5
PROJ1    INT      SOURCE   MODULE6
PROJ1    INT      SOURCE2  INCLUDE3
--------------------------------------------------------------------------------
PROJ1                              4
```

Figure 33 (Part 2 of 2). Database Contents Utility Tailored Report

The reports in Figure 34 on page 82 through Figure 37 on page 83 show examples of a change code, accounting statistics, source listing, and cleanup report.

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; and the report name is CHANGE CODE REPORT. Figure 34 on page 82 shows the tailored output.

```
                                                                    PAGE       2
CHANGE CODE REPORT

@@FLMGRP @@FLMTYP @@FLMMBR @@FLM$CD @@FLM$CC
--------------------------------------------------------------------------------
USER1    SOURCE   MODULE4
INT      SOURCE   MODULE5  02/14/89 2
                           02/01/89 PR3573

                           02/01/89 CR3582

                           02/01/89 PR3456

INT      SOURCE   MODULE6  02/14/89 2
                           02/01/89 PR3573

INT      SOURCE2  INCLUDE3 02/14/89 2
```

Figure 34. Change Code Report, Page 2

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; and the report name is ACCOUNTING STATISTICS REPORT. Figure 35 shows the tailored output.

```
                                                                    PAGE       2
ACCOUNTING STATISTICS REPORT

@@FLMMBR @@FLMLAN @@FLMTLL @@FLMCML @@FLMNCL @@FLMBLL @@FLMTLS @@FLMCMS
--------------------------------------------------------------------------------
MODULE4  PASCAL          8        0        4        4        2        0
MODULE5  PASCAL         13        2        5        6        4        2
MODULE6  PASCAL          8        0        4        4        2        0
INCLUDE3 PASCAL          5        5        0        0        5        5
--------------------------------------------------------------------------------
       4                34        7       13       14       13        7
```

Figure 35. Accounting Statistics Report, Page 2

Source Listing Report: This example shows a generated script data set that the SCRIPT/VS processor can process. However, the data resulting from this formatted input begins in column 2. The SCRIPT/VS processor cannot process the generated data set correctly until you edit the data set so that all command lines begin in the first column. The tailored report uses column 1 for carriage returns.

The report line format input for this example is: .IM @@FLMMBR.

The report does not have page headers, totals, or a name. Figure 36 shows the tailored output.

```
.IM MODULE4
.IM MODULE5
.IM MODULE6
.IM INCLUDE3
```

Figure 36. Source Listing Report, Page 2

Cleanup Report: The cleanup data set is a command data set that can be passed as input to the SCLM command processor. See "Using the FLMCMD File Format" on page 104 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 37 shows the sample tailored report.

```
DELETE,PROJ1   ,PROJ1   ,USER1   ,SOURCE  ,MODULE4
DELETE,PROJ1   ,PROJ1   ,INT     ,SOURCE  ,MODULE5
DELETE,PROJ1   ,PROJ1   ,INT     ,SOURCE  ,MODULE6
DELETE,PROJ1   ,PROJ1   ,INT     ,SOURCE2 ,INCLUDE3
```

Figure 37. Cleanup Report, Page 2

Architecture Report

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 an indented report of the architecture. The report lists software components in each type referenced by the architecture to help you 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 and cross-reference information.

- 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 of various architecture types processed.

- Cross-reference

  Lists all the members, by type, that were listed in the first part of the architecture report. Use this information to determine the origin of a particular member.

  An example of the architecture report appears in Figure 39 on page 85.

SCLM displays the panel in Figure 38 on page 84 when you select Option 5 ARCHITECTURE REPORT on the Utilities panel.

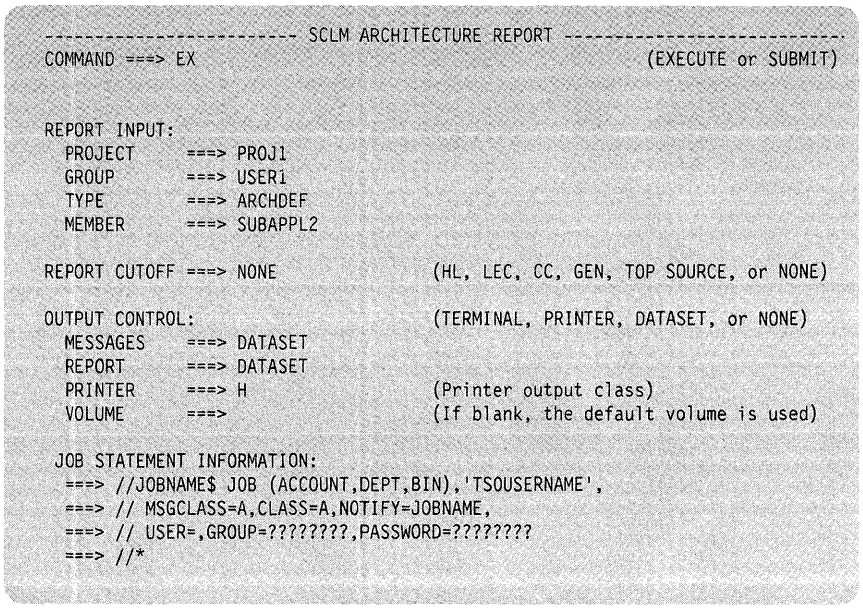


Figure 38. SCLM Architecture Report

The fields on the SCLM Architecture Report panel are:

PROJECT
The project that you specified on the SCLM Primary Option Menu.

GROUP
The group used to identify the lowest level 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 print only the HL architecture members in the application represented by the architecture member you specified in the MEMBER field.

LEC (Linkedit control)
To print 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 print all of the HL, LEC, and CC architecture members in the application represented by the architecture member you specified in the MEMBER field.

GEN (Generic)
To print 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 print all of the HL, LEC, CC, and generic architecture members and the top source members in the application represented by the architecture member you specified in the MEMBER field.

NONE
To print 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.

OUTPUT CONTROL
Specify destinations, such as TERMINAL, PRINTER, or DATASET for the outputs. Also specify which printer output class you want to use and the volume on which SCLM should save data sets.

JOB STATEMENT INFORMATION
Enter the EXECUTE command if you want interactive processing, or enter the SUBMIT command if you want batch processing.

See Figure 46 on page 100 for a sample of the job statement information you must provide if you select batch processing.

Figure 39 and Figure 40 on page 87 show an example of the architecture report with a report cutoff of NONE. Figure 41 on page 89 shows an example of the architecture report with a report cutoff of LEC.

Architecture Report Example

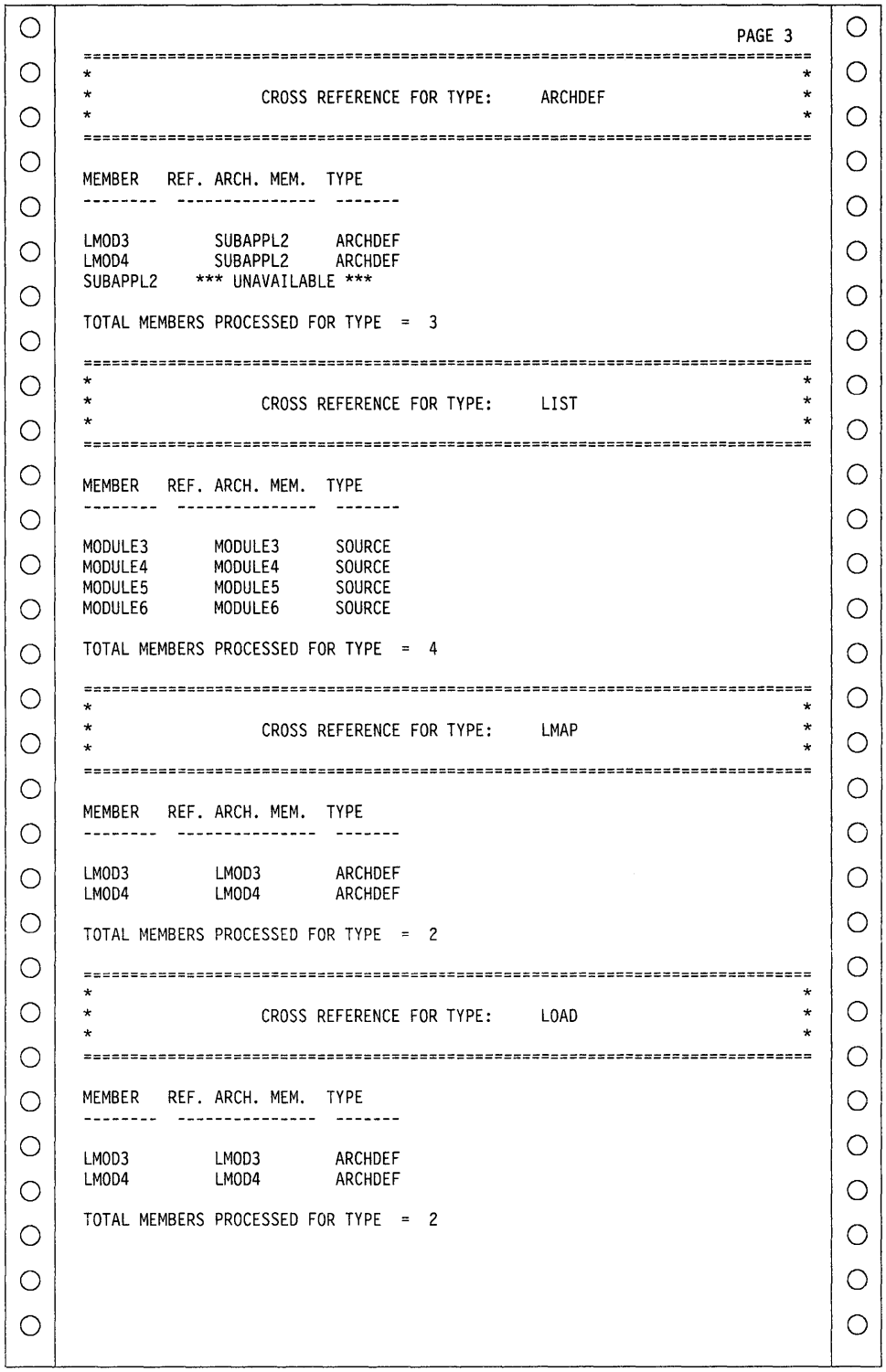
This report provides listings of all the components in a given application. The title page identifies the date and time the report was generated, the architecture member requested, and the report cutoff. It also identifies the alternate project definition, if specified.

```
SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)

ARCHITECTURE REPORT

 02/15/89   09:22:48


 PROJECT:   PROJ1
 GROUP:     USER1
 TYPE:      ARCHDEF
 MEMBER:    SUBAPPL2
 CUTOFF:    NONE
```

Figure 39 (Part 1 of 2). Architecture Report, Part I — Architecture Information

```
                                                                        PAGE 2
 ================================================================================
 *                                                                              *
 *                            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:  SUBAPPL2

 ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---

H   SUBAPPL2
L     LMOD3
D       MODULE5
T         MODULE5
I           INCLUDE3
D       MODULE6
T         MODULE6
D       MODULE3
T         MODULE3
I           INCLUDE2
L     LMOD4
D       MODULE5
T         MODULE5
I           INCLUDE3
D       MODULE6
T         MODULE6
D       MODULE4
T         MODULE4


 NUMBER OF HIGH GROUP MEMBERS PROCESSED           =    1
 NUMBER OF LINK EDIT CONTROL MEMBERS PROCESSED    =    2
 NUMBER OF COMPILATION CONTROL MEMBERS PROCESSED  =    0
 NUMBER OF GENERIC MEMBERS PROCESSED              =    0
 NUMBER OF DEFAULT MEMBERS PROCESSED              =    4
 NUMBER OF TOP MEMBERS PROCESSED                  =    4
 NUMBER OF INCLUDED MEMBERS PROCESSED             =    2
 NUMBER OF ERROR MEMBERS FOUND                    =    0
```

Figure 39 (Part 2 of 2). Architecture Report, Part I — Architecture Information



```
                                                                        PAGE 3
================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     ARCHDEF                     *
*                                                                              *
================================================================================

MEMBER   REF. ARCH. MEM.  TYPE
--------  --------------  -------

LMOD3          SUBAPPL2     ARCHDEF
LMOD4          SUBAPPL2     ARCHDEF
SUBAPPL2    *** UNAVAILABLE ***

TOTAL MEMBERS PROCESSED FOR TYPE  =  3

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     LIST                        *
*                                                                              *
================================================================================

MEMBER   REF. ARCH. MEM.  TYPE
--------  --------------  -------

MODULE3        MODULE3      SOURCE
MODULE4        MODULE4      SOURCE
MODULE5        MODULE5      SOURCE
MODULE6        MODULE6      SOURCE

TOTAL MEMBERS PROCESSED FOR TYPE  =  4

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     LMAP                        *
*                                                                              *
================================================================================

MEMBER   REF. ARCH. MEM.  TYPE
--------  --------------  -------

LMOD3          LMOD3        ARCHDEF
LMOD4          LMOD4        ARCHDEF

TOTAL MEMBERS PROCESSED FOR TYPE  =  2

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     LOAD                        *
*                                                                              *
================================================================================

MEMBER   REF. ARCH. MEM.  TYPE
--------  --------------  -------

LMOD3          LMOD3        ARCHDEF
LMOD4          LMOD4        ARCHDEF

TOTAL MEMBERS PROCESSED FOR TYPE  =  2
```

Figure 40 (Part 1 of 2). Architecture Report, Part II — Cross-Reference Information

```
                                                                        PAGE 4
================================================================================
*                                                                              *
*                 CROSS REFERENCE FOR TYPE:     OBJ                          *
*                                                                              *
================================================================================

MEMBER   REF. ARCH. MEM.  TYPE
--------  --------------  -------

MODULE3       MODULE3      SOURCE
MODULE4       MODULE4      SOURCE
MODULE5       MODULE5      SOURCE
MODULE6       MODULE6      SOURCE

TOTAL MEMBERS PROCESSED FOR TYPE  =  4

================================================================================
*                                                                              *
*                 CROSS REFERENCE FOR TYPE:     SOURCE                       *
*                                                                              *
================================================================================

MEMBER   REF. ARCH. MEM.  TYPE
--------  --------------  -------

INCLUDE2      MODULE3      SOURCE
INCLUDE3      MODULE5      SOURCE
MODULE3       MODULE3      SOURCE
              LMOD3        ARCHDEF
MODULE4       MODULE4      SOURCE
              LMOD4        ARCHDEF
MODULE5       LMOD4        ARCHDEF
              MODULE5      SOURCE
              LMOD3        ARCHDEF
MODULE6       LMOD4        ARCHDEF
              MODULE6      SOURCE
              LMOD3        ARCHDEF

TOTAL MEMBERS PROCESSED FOR TYPE  =  12
```

Figure 40 (Part 2 of 2). Architecture Report, Part II — Cross-Reference Information

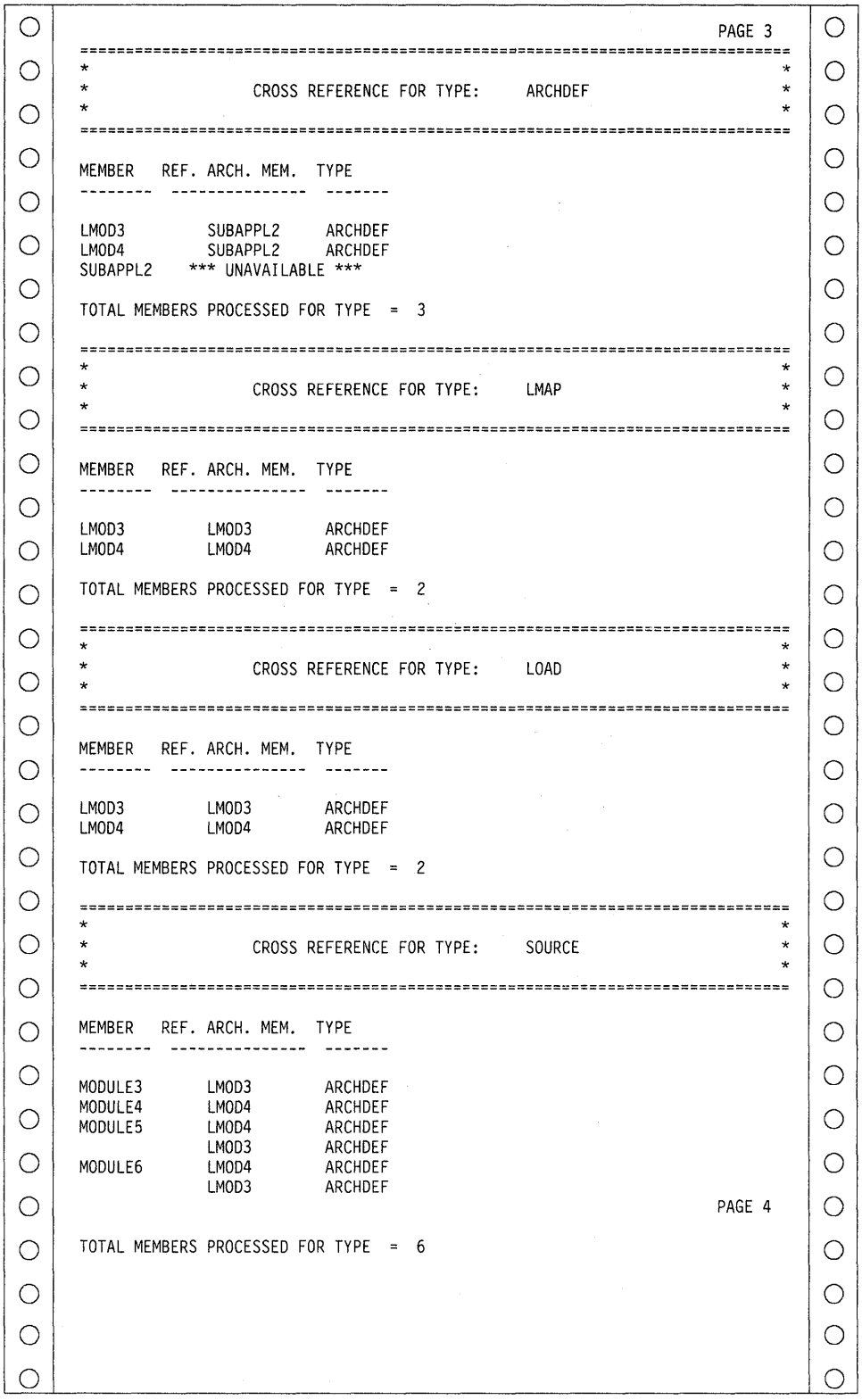
Figure 41 shows an example of the architecture report with an LEC report cutoff.

```
                SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)

                              ARCHITECTURE REPORT

                               02/15/89   09:25:04



                               PROJECT:   PROJ1
                               GROUP:     USER1
                               TYPE:      ARCHDEF
                               MEMBER:    SUBAPPL2
                               CUTOFF:    LINK EDIT CONTROL
```

Figure 41 (Part 1 of 3). Architecture Report, LEC Report Cutoff

```
                                                                          PAGE 2
 ============================================================================
 *                                                                          *
 *                            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:  SUBAPPL2

 ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+---

H   SUBAPPL2
L     LMOD3
L     LMOD4


 NUMBER OF HIGH GROUP MEMBERS PROCESSED          =    1
 NUMBER OF LINK EDIT CONTROL MEMBERS PROCESSED   =    2
 NUMBER OF ERROR MEMBERS FOUND                   =    0
```

Figure 41 (Part 2 of 3). Architecture Report, LEC Report Cutoff



```
                                                                        PAGE 3
================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     ARCHDEF                     *
*                                                                              *
================================================================================

MEMBER   REF. ARCH. MEM.  TYPE
--------  --------------  -------

LMOD3          SUBAPPL2     ARCHDEF
LMOD4          SUBAPPL2     ARCHDEF
SUBAPPL2    *** UNAVAILABLE ***

TOTAL MEMBERS PROCESSED FOR TYPE  =  3

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     LMAP                        *
*                                                                              *
================================================================================

MEMBER   REF. ARCH. MEM.  TYPE
--------  --------------  -------

LMOD3          LMOD3        ARCHDEF
LMOD4          LMOD4        ARCHDEF

TOTAL MEMBERS PROCESSED FOR TYPE  =  2

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     LOAD                        *
*                                                                              *
================================================================================

MEMBER   REF. ARCH. MEM.  TYPE
--------  --------------  -------

LMOD3          LMOD3        ARCHDEF
LMOD4          LMOD4        ARCHDEF

TOTAL MEMBERS PROCESSED FOR TYPE  =  2

================================================================================
*                                                                              *
*                  CROSS REFERENCE FOR TYPE:     SOURCE                      *
*                                                                              *
================================================================================

MEMBER   REF. ARCH. MEM.  TYPE
--------  --------------  -------

MODULE3        LMOD3        ARCHDEF
MODULE4        LMOD4        ARCHDEF
MODULE5        LMOD4        ARCHDEF
               LMOD3        ARCHDEF
MODULE6        LMOD4        ARCHDEF
               LMOD3        ARCHDEF
                                                                        PAGE 4

TOTAL MEMBERS PROCESSED FOR TYPE  =  6
```

Figure 41 (Part 3 of 3). Architecture Report, LEC Report Cutoff

Build (Option 4)

The build processor automatically compiles and links modules requiring processing. The panel shown in Figure 42 appears when you select Option 4 BUILD from the SCLM Primary Option Menu.

```
------------------------------ SCLM - BUILD ---------------------------------
COMMAND ===> EX                                          (EXECUTE or SUBMIT)

BUILD INPUT:
  PROJECT     ===> PROJ1
  GROUP       ===> USER1
  TYPE        ===> ARCHDEF
  MEMBER      ===> LMOD3

BUILD SCOPE   ===> NORMAL          (LIMITED, NORMAL, SUBUNIT, or EXTENDED)
BUILD MODE    ===> FORCED          (CONDITIONAL, UNCONDITIONAL, FORCED
                                    or REPORT)
OUTPUT CONTROL:                    (TERMINAL, PRINTER, DATASET, or  NONE)
  MESSAGES    ===> TERMINAL
  REPORT      ===> DATASET
  LISTINGS    ===> DATASET         ERROR LISTINGS ONLY ===> YES
  PRINTER     ===> H               (Printer output class)
  VOLUME      ===>                 (If blank, the default volume is used)

 JOB STATEMENT INFORMATION:
  ===> //JOBNAME$ JOB (ACCOUNT,DEPT,BIN),'TSOUSERNAME',
  ===> // MSGCLASS=A,CLASS=A,NOTIFY=JOBNAME,
  ===> // USER=,GROUP=????????,PASSWORD=????????
  ===> //*
```

Figure 42. SCLM Build

The fields for the SCLM Build panel are:

PROJECT
The project that you specified on the SCLM Primary Option Menu.

GROUP
The group in which the build is to occur.

TYPE
The type of the member.

MEMBER
The name of the member to be built.

BUILD SCOPE
Select one of the following:

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.

Note: Do not specify a scope other than NORMAL unless each source member you want translated has compilation unit dependencies. Otherwise, specify a scope equal to or greater than the scope specified with the SCOPE keyword in the FLMLANGL macro.

BUILD MODE
Select one of the following:

Conditional
To check for unacceptable compile or link return codes. Processing stops immediately if build detects any unacceptable codes.

SCLM saves build maps and translator output only for compiles and links that complete successfully. SCLM generates translator listings for all components processed, and the build report reflects the final results of the build.

Unconditional
To continue processing despite translation errors.

Use this mode when you need to update complete applications or large subapplications. You can also use this mode initially to detect compile and link errors in several components.

Forced
To force all requested components to be compiled and linked again regardless of the previous status of the modules.

Use this mode to create a listing for a current component whose listing is not tracked by SCLM.

Report only
To generate a complete build report without performing an actual build. The report reflects the potential results of an unconditional build.

OUTPUT CONTROL
Specify destinations, such as TERMINAL, PRINTER, or DATASET, for these outputs. Also specify which printer output class you want to use and the volume on which SCLM should save data sets.

JOB STATEMENT INFORMATION
Enter the EXECUTE command if you want interactive processing, or enter the SUBMIT command if you want batch processing.

See Figure 46 on page 100 for a sample of the job statement information you must provide if you select batch processing.

Build Report Example

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 Primary Option Menu.

The report lists the components that were rebuilt 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 regeneration, along with a list of software components that caused the regeneration.

If you enter REPORT ONLY in the BUILD MODE field, the report indicates what would be rebuilt if you requested an unconditional build.

Figure 43 shows an example of a build report.

```
                SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)

                          B U I L D    R E P O R T

                             02/15/89   09:28:35

                           PROJECT:       PROJ1
                           GROUP:         USER1
                           TYPE:          ARCHDEF
                           MEMBER:        LMOD3
                           ALTERNATE:     PROJ1
                           SCOPE:         NORMAL
                           MODE:          FORCED
```

```
          ********** B U I L D   O U T P U T S **********  Page 1

MEMBER     TYPE     VERSION       KEYWORD
------     ----     -------       -------

MODULE3    OBJ              6     OBJ
MODULE5    OBJ              6
MODULE6    OBJ              6

MODULE3    LIST             6     LIST
MODULE5    LIST             6
MODULE6    LIST             6

LMOD3      LOAD             6     LOAD

LMOD3      LMAP             6     LMAP
```

Figure 43 (Part 1 of 2). Build Report

```
******* 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
------     ----       -------     -------    ----

LMOD3      ARCHDEF          6     ***  FORCE MODE  ***
MODULE3    SOURCE           6     ***  FORCE MODE  ***
MODULE5    SOURCE           6     ***  FORCE MODE  ***
MODULE6    SOURCE           6     ***  FORCE MODE  ***
```

Figure 43 (Part 2 of 2). Build Report

Promote (Option 5)

The promote function moves members from any group to the next higher group. The panel shown in Figure 44 appears when you select Option 5 PROMOTE from the SCLM Primary Option Menu.

```
------------------------------ SCLM - PROMOTE -------------------------------
COMMAND ===> EX                                          (EXECUTE or SUBMIT)

PROMOTE INPUT:
  PROJECT     ===> PROJ1
  FROM GROUP  ===> STAGE1
  TYPE        ===> ARCHDEF
  MEMBER      ===> LMOD3

PROMOTE SCOPE ===> NORMAL          (NORMAL, SUBUNIT, or EXTENDED)
PROMOTE MODE  ===> CONDITIONAL     (CONDITIONAL, UNCONDITIONAL, or REPORT)

OUTPUT CONTROL:                    (TERMINAL, PRINTER, DATASET, or  NONE)
  MESSAGES    ===> TERMINAL
  REPORT      ===> DATASET
  PRINTER     ===> H               (Printer output class)
  VOLUME      ===>                 (If blank, the default volume is used)

 JOB STATEMENT INFORMATION:
  ===> //JOBNAME$ JOB (ACCOUNT,DEPT,BIN),'TSOUSERNAME',
  ===> // MSGCLASS=A,CLASS=A,NOTIFY=JOBNAME,
  ===> // USER=,GROUP=????????,PASSWORD=????????
  ===> //*
```

Figure 44. SCLM Promote

The fields on the SCLM Promote panel are:

PROJECT
The project that you specified on the SCLM Primary Option Menu.

FROM GROUP
The group from which to promote the material that the architecture member refers to.

TYPE
The type of the architecture member.

MEMBER
The name of the architecture member to be promoted.

For information on architecture members, see Chapter 2, "Architecture Definition."

PROMOTE SCOPE

Select one of the following:

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: Do not specify a scope other than NORMAL unless each source member you want translated has compilation unit dependencies. Otherwise, specify a scope equal to or greater than the scope specified with the SCOPE keyword in the FLMLANGL macro.

PROMOTE MODE

Select one of the following:

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 despite verification errors and to promote only those members with correct accounting 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 anyway, you can use this mode to promote those software components.

The use of the unconditional mode does not guarantee application integrity, and you should use it with extreme caution. It is, however, an effective method of promoting dependent software components that you plan to integrate at a later date.

Report only

To perform verification and report generation processing. The report contains a list of members eligible for promotion.

OUTPUT CONTROL

Specify destinations, such as TERMINAL, PRINTER, or DATASET, for the outputs. Also specify which printer output class you want to use and the volume on which SCLM should save data sets.

JOB STATEMENT INFORMATION

Enter the EXECUTE command if you want interactive processing, or enter the SUBMIT command if you want batch processing.

See Figure 46 on page 100 for a sample of the job statement information you must provide if you select batch processing.

Promote Report

Figure 45 on page 97 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 (*) (see Note).

Note: An *out-of-scope* software component is an architecture that is referenced with a LINK or CREF 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 second promote that follows a failed promote, it indicates the work completed by that promote only.

For more information on key and non-key groups, see "Key/Non-Key Groups" on page 8.

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)

                        P R O M O T E    R E P O R T

                              02/15/89   09:35:41


                          PROJECT:       PROJ1
                          TO GROUP:      INT
                          FROM GROUP:    STAGE1
                          TYPE:          ARCHDEF
                          ARCH. MEM.:    LMOD3
                          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
 --------  --------  --------  --------  ---------  -----------  -----------

 ARCHCOPY  02/14/89  16:52:00                X          X            X
 LMOD3     02/14/89  16:54:00                X          X            X
```

```
                                                                  PAGE    3

 TYPE: LIST

                                                  COPIED TO  PURGED FROM  PURGED FROM
 MEMBER      DATE      TIME     MESSAGE            INT          STAGE1       USER1
 --------  --------  --------  --------  ---------  -----------  -----------

 MODULE3   02/15/89  09:30:00                X          X            X
 MODULE5   02/15/89  09:29:00                X          X            X
 MODULE6   02/15/89  09:29:00                X          X            X
```

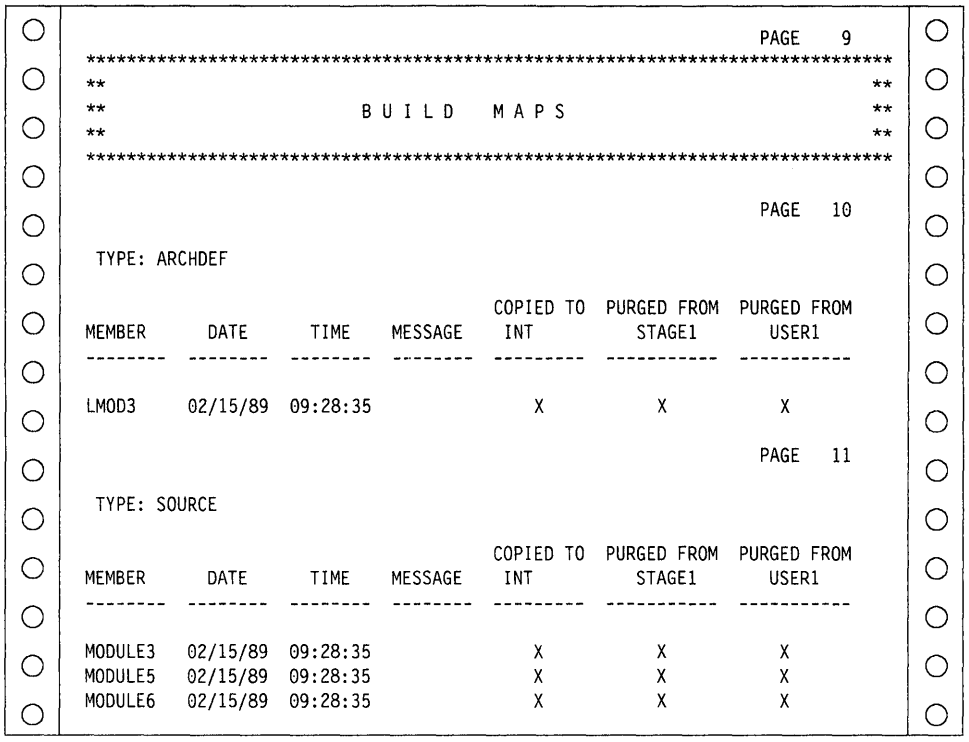
```
                                                                  PAGE    4

 TYPE: LMAP

                                                  COPIED TO  PURGED FROM  PURGED FROM
 MEMBER      DATE      TIME     MESSAGE            INT          STAGE1       USER1
 --------  --------  --------  --------  ---------  -----------  -----------

 LMOD3     02/15/89  09:31:00                X          X            X
```

Figure 45 (Part 1 of 3). Promote Report

```
                                                                    PAGE    5

 TYPE: LOAD

                                                 COPIED TO  PURGED FROM  PURGED FROM
 MEMBER      DATE      TIME     MESSAGE           INT        STAGE1       USER1
 --------    --------  --------  --------  ---------  -----------  -----------

 LMOD3       02/15/89  09:31:00                X          X            X

                                                                    PAGE    6

 TYPE: OBJ

                                                 COPIED TO  PURGED FROM  PURGED FROM
 MEMBER      DATE      TIME     MESSAGE           INT        STAGE1       USER1
 --------    --------  --------  --------  ---------  -----------  -----------

 MODULE3     02/15/89  09:30:00                X          X            X
 MODULE5     02/15/89  09:29:00                X          X            X
 MODULE6     02/15/89  09:29:00                X          X            X
```

```
                                                                    PAGE    7

 TYPE: SOURCE

                                                 COPIED TO  PURGED FROM  PURGED FROM
 MEMBER      DATE      TIME     MESSAGE           INT        STAGE1       USER1
 --------    --------  --------  --------  ---------  -----------  -----------

 MODULE3     02/14/89  16:33:00                X          X            X
 MODULE5     02/14/89  17:03:00                X          X            X
 MODULE6     02/14/89  16:48:00                X          X            X

                                                                    PAGE    8

 TYPE: SOURCE2

                                                 COPIED TO  PURGED FROM  PURGED FROM
 MEMBER      DATE      TIME     MESSAGE           INT        STAGE1       USER1
 --------    --------  --------  --------  ---------  -----------  -----------

 INCLUDE2    02/14/89  19:49:00                X          X            X
 INCLUDE3    02/14/89  16:50:00                X          X            X
```

Figure 45 (Part 2 of 3). Promote Report



```
                                                                        PAGE    9
********************************************************************************
**                                                                            **
**                        B U I L D   M A P S                                 **
**                                                                            **
********************************************************************************

                                                                        PAGE   10

 TYPE: ARCHDEF

                                                COPIED TO  PURGED FROM  PURGED FROM
MEMBER      DATE      TIME     MESSAGE          INT          STAGE1       USER1
--------  --------  --------  --------  ---------  -----------  -----------

LMOD3     02/15/89  09:28:35                X            X            X

                                                                        PAGE   11

 TYPE: SOURCE

                                                COPIED TO  PURGED FROM  PURGED FROM
MEMBER      DATE      TIME     MESSAGE          INT          STAGE1       USER1
--------  --------  --------  --------  ---------  -----------  -----------

MODULE3   02/15/89  09:28:35                X            X            X
MODULE5   02/15/89  09:28:35                X            X            X
MODULE6   02/15/89  09:28:35                X            X            X
```

Figure 45 (Part 3 of 3). Promote Report

Processing Errors

The promote function can recover from most database errors. However, data set overflow and data contention, as described below, may 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.
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. You should not promote or build members while they are processing during another promote. Compiler errors or promote verification errors in one or more of the concurrent jobs can occur. You can recover from all errors by running the failed function again.

Batch Processing

The Verify Batch Job Information panel shown in Figure 46 is the standard panel for the SCLM functions that allow you to select batch processing. When you enter the SUBMIT command 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.

```
---------------------- VERIFY BATCH JOB INFORMATION ----------------------
COMMAND ===>

Enter/verify JOB statement information below to continue SUBMIT processing:

  ===> // USERID$ JOB  (ACCOUNT,DEPT,BIN),'TSOUSERNAME',
  ===> // USGCLASS=A,CLASS=A,NOTIFY=USERID,
  ===> // USER=,GROUP=????????,PASSWORD=????????
  ===> //*
  ===> //*
  ===> //*
  ===> //*
  ===>
```

Figure 46. Verify Batch Job Information

Output Disposition

The Output Disposition panel shown in Figure 47 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.

```
---------------------------- OUTPUT DISPOSITION ----------------------------
COMMAND ===> PD

  PK - Print and keep data set      K - Keep data set (without printing)
  PD - Print and delete data set    D - Delete data set (without printing)

   If END command is entered, data set is kept without printing.

DATASET NAME: 'userid.filename'

General purpose print/punch SYSOUT class information:
   PRINT ===> A
   PUNCH ===>


JOB STATEMENT INFORMATION:
    ===> //jobname JOB (wrkpkg,dept,bin),'NAME',CLASS=C,MSGCLASS=H,
    ===> //  USER=????????,PASSWORD=????????,
    ===> //  GROUP=????????,NOTIFY=????????
    ===> //*
```

Figure 47. Output Disposition

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 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 browse, edit, library, and sublibrary management utility functions do not create report or message data sets and, consequently, do not display the Output Disposition panel.

Chapter 5. SCLM Services

General-Use Programming Interface

The SCLM services are general-use programming interfaces, which you can use for programming purposes.

This chapter describes each of the SCLM services and the syntax conventions and return codes for the services. It discusses how to call the services from your terminal with interactive command processing, procedures, or programs. This chapter also provides several brief examples of sample command data sets, procedures, and programs.

Included in each service description is an example of its use in the command procedure format and the Pascal call format. See Chapter 6, "A Sample Program Using SCLM Services," for an example of service invocations and declarations coded in Pascal.

For information on how to use the SCLM services without ISPF/PDF, see "Development and Performance" on page 274. For instructions on encrypting and decrypting partitioned data sets, see "Data Set Protection" on page 276.

Invoking the SCLM Services

Invoke the SCLM services by a program function dialog through a call to FLMCMD or FLMLNK, or by a command function dialog (CLIST or REXX) through the ISPF/PDF interface.

Notation Conventions Used in this Chapter

This chapter uses the following notation conventions to describe the format of the SCLM services:

Uppercase Uppercase commands or parameters must be spelled out as shown (in either uppercase or lowercase).

Lowercase Lowercase parameters are variables; substitute your own values.

Underscore Underscored parameters are the system default.

Brackets ([ ]) Parameters in brackets are optional.

Braces ({ }) Braces show two or more parameters from which you must select one.

OR (|) The OR (|) symbol shows two or more parameters from which you must select one.

Single Quotes (' ')
Single quotes show service names and keywords in call invocation examples.

Stacked Parameters
Stacked parameters show two or more parameters from which you can select. If you do not choose any, ISPF/PDF uses the default parameter.

© Copyright IBM Corp. 1989, 1990

Command Invocation of the SCLM Services

Call the SCLM services 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 eight characters long.

project_name
Alphanumeric; up to eight characters long.

prj_def_name
Alphanumeric; up to eight characters long.

The remaining parameters are positional. They must appear in the order described for each service.

Although lowercase parameters are optional, SCLM uses default values for those parameters you do not choose.

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
```

Using Command Invocation Variables

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 *TSO Extensions Version 2 Command Language Reference* (SC28-1881) and *TSO Extensions Version 2 CLISTs* (SC28-1876).

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:

```
FLMCMD FILE,ddname
```

The ddname is the data definition name allocated to the FLMCMD command dataset. If you do not specify the ddname, SCLM prompts you for command lines. The record length of the command data set cannot exceed 256 bytes. For more information, see "Interactive Command Processing" on page 106.

All messages from FLMCMD appear on your screen. To reroute the messages to another destination, allocate the FLMMSGS ddname to the desired destination.

Note: "Performance Considerations" on page 276 discusses the use of the FILE format for efficient processing.

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,MODULE2,TESTAC,XXX#04,USERID
*
* The following command consists of four lines,
* and calls the SCLM UNLOCK service.
UNLOCK,PROJ1,,+
USER1,+
SOURCE,+
MODULE2,XXX#04
```

Command Data Set Conventions

Command data sets use the following conventions:

- SCLM processes all commands in the command data set regardless of the success or failure of previous commands.
- 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 column one of the succeeding lines. 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 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 = &LASTCC
  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. During interactive command processing, you can enter comment lines but you cannot enter continuation lines.

To end interactive command processing, enter the QUIT command.

If you allocate the ddname to your screen and also specify it on the FILE format of FLMCMD, you can get unpredictable results.

The following example shows a sample interactive command session.

```
 READY
FLMCMD FILE
Enter a command line; press Enter to process a command; or "QUIT":
LOCK,PROJ1,,USER1,SOURCE,MODULE2,TESTAC,XXX#04,USERID

     <messages will appear here>

Enter a command line; press Enter to process a command; or "QUIT":
UNLOCK,PROJ1,,USER1,SOURCE,MODULE2,XXX#04

     <messages will appear here>

Enter a command line; press Enter to process a command; or "QUIT":
QUIT
 READY
```

The FLMLNK Subroutine Interface

Programs in the FLMLNK subroutine interface call the SCLM services. This chapter shows 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

SCLM services can be issued from function modules that reside either below or above the 16-megabyte line. The interface module FLMLNK has the attributes RMODE(24) and AMODE(ANY). These attributes allow both 24-bit and 31-bit addressing mode callers. Data areas above the 16-megabyte line are also supported.

Note: The FLMLNK module is shipped with the RMODE(24) attribute to provide compatibility with function modules that have the AMODE(24) attribute and that will use a load and call interface to FLMLNK. Modules that reside above the 16-megabyte line (RMODE(ANY)) and include FLMLNK in their load module can override the RMODE(24) attribute during link edit. FLMLNK can reside above the 16-megabyte line.

Standard register conventions are used. Registers 2-14 are preserved across the call.

Call Invocation

Other than for Pascal and FORTRAN, the general call format for invoking SCLM services from functions by using FLMLNK is:

```
CALL FLMLNK(service_name,parameter1,parameter2,...);
```

FLMLNK Parameter Conventions

service_name
Alphanumeric; up to eight characters long.

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. You cannot omit required parameters from the call statement. 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.
- Some of the service input parameters are optional, but SCLM uses a default value if you do not choose a parameter.
- To omit a parameter, insert a blank enclosed in single quotes (' ') in its place.
- 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 the following 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,SCLMID*8,GROUP*8
DATA         SERVIS/'DELETE  '/
DATA         SCLM_ID/'SCLM00001'/
DATA         GROUP/'USER1   '/
             .
             .

LASTRC=FLMLNK(SERVICE,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=DELETE
```

Pascal Example

```
CONST
  SERVICE = 'DELETE  ';
  SCLM_ID = 'SCLM00001';
  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;
chars SERVICE, SCLMID, GROUP, ...
SERVICE = "DELETE  "
SCLMID = "SCLM00001"
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('SCLM00001'),
        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

COBOL does not allow literals within a call statement. Therefore, SCLM does not require the use of literals. You can specify all parameters as variables, as in the following example:

COBOL Example

```
WORKING-STORAGE TYPE.
   77  SERVIS      PICTURE A(8) VALUE 'DELETE  '.
   77  SCLM_ID     PICTURE A(8) VALUE 'SCLM00001'.
   77  GROUP       PICTURE A(8) VALUE 'USER1   '.
       .
       .

PROCEDURE DIVISION.
   CALL 'FLMLNK' USING SERVICE SCLM_ID GROUP  ...  .
```

For service calls in COBOL, initialize parameter variables by using literals in assignment statements:

```
MOVE 'DELETE' TO SERVIS.
```

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. However, if you use different attributes to allocate the ddnames, SCLM creates the data set using the attributes specified, but the format of the resulting file may not be usable.

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.

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 6, "A Sample Program Using SCLM Services."

Note: SCLM frees all memory associated with an output pointer parameter at the start of the next service call. Copy any data associated with an output pointer parameter that is to be referenced after the start of the next service call to the function module's local storage.

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 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 6, "A Sample Program Using SCLM Services."

Pointer Parameter Descriptions

The following 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 in length. An END record denotes the end of the message array. Figure 48 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: MODULE1
Record 3: END
```

Figure 48. $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. For a description of the record field contents, see "Accounting Record" on page 55.

The following fields contain data common to all members:

acct_group	8 characters
acct_type	8 characters
acct_member	8 characters
SCLM_version	2 characters ('60')
accounting_status	1 character: E Editable N Non-editable L Lockout I Initial
change_date	6 characters (YYMMDD format)
change_time	6 characters (HHMMSS format)
change_group	8 characters
change_userid	8 characters
member_version	Fullword integer
language	8 characters
authorization_code	8 characters
authorization_code_change	8 characters
access_key	16 characters
creation_date	6 characters (YYMMDD format)
creation_time	6 characters (HHMMSS format)
map_date	6 characters (YYMMDD format)
map_time	6 characters (HHMMSS format)
predecessor_date	6 characters (YYMMDD format)
predecessor_time	6 characters (HHMMSS format)
promote_date	6 characters (YYMMDD format)
promote_time	6 characters (HHMMSS format)
promote_userid	8 characters
db_qual	8 characters

All of the following eight-character fields are blank unless the accounting_status is N.

translator_version	language_version
map_name	map_type

The following fields contain statistical data for a member. The fields preceded with an asterisk refer to statistics that the parsers, supplied by SCLM, do not collect.

total_lines	* assignment_stmts
comment_lines	non_comment_stmts
non_comment_lines	number_of_user_entries
blank_lines	number_of_includes
* prolog_lines	number_of_compools
total_stmts	number_of_changecodes
comment_stmts	number_of_cus
* control_stmts	

Each field is a fullword integer.

$stats_info: A pointer to a record containing a member's statistical information. The following describes the format of the record fields. The fields preceded with an asterisk refer to statistics that the parsers, supplied by SCLM, do not collect.

total_lines	total_stmts
comment_lines	comment_stmts
non_comment_lines	* control_stmts
blank_lines	* assignment_stmts
* prolog_lines	non_comment_stmts

Each of the fields is a fullword integer. For a description of the record field contents, see "Statistics" on page 57.

$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, compool, compilation unit, change code, and user entry information. Each record in the array is 228 bytes in length.

Some of the SCLM services place restrictions on the data that you can specify with this parameter. See the description for the service you want to use to verify whether it restricts the $list_info parameter data.

The records in the array contain two fields. The first field, which is four characters, indicates the record type. Valid record type values are:

END	Indicates the end of the array
INCL	Indicates an include
COMP	Indicates a compool; only used for compool languages, such as JOVIAL
CODE	Indicates a change code
USER	Indicates user data
CU	Indicates a compilation unit; only used for the Ada language.

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 is a description of the data in the second field for each record type:

END	No data.
INCL	Member name (8 characters) upon which the "member" has an include dependency.
COMP	Compool name (8 characters) upon which the "member" has a compool dependency.
CODE	A record detailing a change code associated with the "member." The total record length is 20 bytes. The record contains a change code (8 characters), a change code date stamp (6 characters, YYMMDD format), and a change code time stamp (6 characters, HHMMSS format).
USER	User data (128 characters) associated with the "member."

CU A record containing two parts, which describes a compilation unit. The total record length is 224 bytes.

Part one of the record is 112 bytes in length and identifies a compilation unit that is contained within the "member." Part one contains the following information:

cu_name Compilation unit name (110 characters).

cu_type Compilation unit type (1 character). Valid values are:

- B Indicates that the cu_name is either a package body or procedure body
- S Indicates that the cu_name is either a package specification or procedure specification
- X Indicates the cu_name to use for the Ada bind process.

generic_flag Compilation unit attribute (1 character). Valid values are:

- G Indicates that the compilation unit is generic
- I Indicates that PRAGMA INLINE has been referenced
- B Indicates that the compilation unit is generic and PRAGMA INLINE has been referenced
- N Indicates that the compilation unit is not generic and PRAGMA INLINE has not been referenced.

Part two identifies a compilation unit that the part one compilation unit depends upon. Part two contains the following information:

depend_cu_name
Compilation unit name (110 characters).

depend_cu_type Compilation unit type (1 character). Valid values are:

- B Indicates that the depend_cu_name is either a package body or procedure body
- S Indicates that the depend_cu_name is either a package specification or procedure specification.

depend_cu_depend_type
Indicates the type of dependency (1 character). Valid values are:

- U The dependency is upward; indicates that the dependent compilation unit (depend_cu_name) must be compiled before the current compilation unit (cu_name).
- D The dependency is downward; indicates that the current compilation unit (cu_name) must be compiled before the dependent compilation unit (depend_cu_name).

Notes:

1. A compilation unit (cu_name/cu_type) may appear on more than one CU entry in the $list_info array. To identify all compilation unit dependencies, the compilation unit must appear once (in part one of the record) for each of its dependent compilation units.
2. All Ada compilation units need at least one dependency. But if for some reason a compilation unit were to have no dependencies, only one CU entry would appear for that compilation unit. The depend_cu_name, depend_cu_type, and depend_cu_depend_type are set to blank for the CU entry.
3. Only one CU record may be present in the $list_info array if a record with a cu_type of X exists. SCLM uses X to abbreviate XREF or cross-reference.
4. A CU record with a cu_type of X must have the following:
   - A dependency
   - A generic_flag of N
   - A cu_name that matches the depend_cu_name
   - A depend_cu_depend_type of U.

   Also, a CU record with a cu_type of X should have a depend_cu_type of B.
5. 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 49 shows the contents of a list information array. Two change codes (PR1234 on 12/16/87 at 12:01:33 and CR000032 on 1/4/88 at 00:53:16) and a user entry indicating a customized member are associated with the "member."

The "member" also contains one compilation (the TEST_DRIVER generic package specification) unit with a downward dependency on the TEST_DRIVER package body and an upward dependency on the COMMON_TYPES package specification.

```
Record 1: CODEPR1234  871216120133
Record 2: CODECR000032880104005316
Record 3: USERTEST MEMBER - CUSTOMIZED
Record 4: CU  TEST_DRIVER...<99 blanks>...SGTEST_DRIVER...<99 blanks>...SD
Record 5: CU  TEST_DRIVER...<99 blanks>...SGCOMMON_TYPES..<98 blanks>...SU
Record 6: END
```

Figure 49. $list_info Contents

SCLM Service Return Codes

Each service returns a numeric code, called a *return code*, indicating the results of the operation. The following are possible return codes:

0 Indicates successful completion. SCLM may or may not generate messages.

4 Indicates a warning condition. SCLM may or may not generate messages.

8 Indicates an error condition. SCLM generates messages detailing the error.

> 8 Indicates a severe error condition. SCLM does not generate messages.

> 32 Indicates an ABEND. The hexadecimal value of the return code determines system completion.

Return codes and their meanings vary for each service and are listed with each service description.

For command invocation, SCLM returns the code in the CLIST variable *&lastcc*. 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.

SCLM Service Descriptions

This section contains information about the services available for SCLM.

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, a semicolon (;) ends statements. This is a Pascal convention, but 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.

BUILD—Build a Member

The BUILD service compiles, links, and integrates software components according to a project's architecture definition. Before building a member, 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.

For more information on the SCLM build function, see "Build Function" on page 14.

Command Invocation Format

```
FLMCMD BUILD,project
           ,[prj_def]
           ,group
           ,type
           ,member
           ,[userid]
           ,[E|L|N|S]
           ,[C|F|R|U]
           ,[Y|N]
           ,[Y|N]
           ,[prefix_userid]
           ,[dd_bldmsgs]
           ,[dd_bldrept]
           ,[dd_bldlist]
           ,[dd_bldexit]
```

Call Invocation Format

```
lastrc := FLMLNK('BUILD',sclm_id
                        ,group
                        ,type
                        ,member
                        ,{userid|' '}
                        ,{E|L|N|S}
                        ,{C|F|R|U}
                        ,{Y|N}
                        ,{Y|N}
                        ,{prefix_userid|' '}
                        ,dd_bldmsgs
                        ,dd_bldrept
                        ,dd_bldlist
                        ,dd_bldexit);
```

Parameters

project
The project name. The maximum parameter length is eight characters.

prj_def
The project definition name used for the build. It defaults to the project parameter. The maximum parameter length is eight characters.

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 eight characters.

group
The group in which the build occurs. The maximum parameter length is eight characters.

type
The type containing the member to be built. The maximum parameter length is eight characters.

member
The member to be built. The maximum parameter length is eight characters.

userid
The user ID of the person requesting the build. It defaults to your TSO prefix or user ID if no TSO prefix has been created. The maximum parameter length is eight characters.

E|L|N|S
Indicates the build scope (E = extended, L = limited, N = normal, S = subunit). The maximum parameter length is 24 characters. See the field definitions in "Build (Option 4)" on page 91 for more details.

C|F|R|U
Indicates the build mode (C = conditional, F = forced, R = report, U = unconditional). The maximum parameter length is 24 characters. See the field definitions in "Build (Option 4)" on page 91 for more details.

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. The maximum parameter length is 24 characters. See "Build Listings" on page 17 for more details on build listings.

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. The maximum parameter length is 24 characters. See "Build Report" on page 16 for more details.

prefix_userid
The high group qualifier to be used when allocating and cataloging temporary data sets. 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 with the following attributes: `RECFM=F, LRECL=80, BLKSIZE=80`. The maximum parameter length is eight characters. See "Build Messages" on page 16 for more details.

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 with the following attributes: `RECFM=FBA, LRECL=80, BLKSIZE=3120`. The maximum parameter length is eight characters. See "Build Report" on page 16 for more details.

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 with the following attributes: `DISP=MOD, RECFM=VBA, LRECL=137, BLKSIZE=3120`. The maximum parameter length is eight characters. See "Build Listings" on page 17 for more details.

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 administrator 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 with the following attributes: `RECFM=FB, LRECL=160, BLKSIZE=3200`. The maximum parameter length is eight characters.

Return Codes

Possible return codes are:

0 Normal completion. See the dd_bldmsgs parameter description for more details.

4 Warning condition. See the dd_bldmsgs parameter description for more details.

8 Error condition. See the dd_bldmsgs parameter description for more details.

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.

20 Severe error condition. SCLM does not produce messages because the SCLM ID is invalid.

24 Severe error condition. SCLM does not produce messages because SCLM services have not been initialized. See "START—Generate an Application ID for a Services Session" on page 157 for information on initializing an SCLM services session.

32 Severe error condition. SCLM does not produce messages for one of the following reasons:

- You requested an invalid service.
- You supplied an invalid parameter list for the requested service.
- The version of the FLMLNK subroutine does not match the version of the SCLM services module (for future use).

Examples

These examples call the BUILD service.

Command Invocation

```
FLMCMD BUILD,PROJ1,,USER1,ARCHDEF,CMOD1,,,U,,N
```

This service command builds the CMOD1 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.

Call Invocation

```
lastrc := FLMLNK('BUILD',sclm_id,
                 'USER1','ARCHDEF','CMOD1',
                 ' ','N','F','N','Y',
                 'PROJECT.WORKFILE ',
                 'BLDMSGS','BLDREPT','BLDLIST','BLDEXIT');
```

The service call builds the CMOD1 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-group 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.

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 on SCLM accounting records, see "Accounting Record" on page 55.

Command Invocation Format

You cannot use command procedures to call this service.

Call Invocation Format

```
lastrc := FLMLNK('DBACCT',sclm_id
                         ,group
                         ,type
                         ,member
                         ,found_group
                         ,$acct_info
                         ,$list_info
                         ,$msg_array);
```

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 eight characters.

group
The group in which the accounting record search begins. The maximum parameter length is eight characters.

type
The type containing the accounting record retrieved. The maximum parameter length is eight characters.

member
The member whose accounting record is retrieved. The maximum parameter length is eight 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 eight characters.

$acct_info
An output parameter pointing to a record containing the static portion of the member's accounting record. See "Pointer Parameters" on page 110 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 Parameters" on page 110 for more details on $list_info.

$msg_array
An output parameter pointing to the message array. See "Pointer Parameters" on page 110 for more details on $msg_array.

Return Codes

Possible return codes are:

0 Normal completion.

4 Warning condition. SCLM could not find the accounting record.

8 Error condition. See the $msg_array parameter description for more details.

20 Severe error condition. SCLM does not produce messages because the SCLM ID is invalid.

24 Severe error condition. SCLM does not produce messages because SCLM services have not been initialized. See "START—Generate an Application ID for a Services Session" on page 157 for information on initializing an SCLM services session.

32 Severe error condition. SCLM does not produce messages for one of the following reasons:

- You requested an invalid service.
- You supplied an invalid parameter list for the requested service.
- The version of the FLMLNK subroutine does not match the version of the SCLM services module (for future use).

Example

This example calls the DBACCT service.

Call Invocation

```
lastrc := FLMLNK('DBACCT',sclm_id,
                 'USER1','SOURCE','MODULE1',
                 found_group,
                 $acct_info,$list_info,$msg_array);
```

This service call returns the first occurrence of the accounting record for the MODULE1 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 in the $msg_array.

DBUTIL—Generate a Tailored Data Set and Report

The DBUTIL service retrieves information from the project database and creates a tailored data set and a report. SCLM generates the tailored data set in the format you specify. It also reflects the contents of the project database based on the selection criteria you supply. You can use the tailored data set 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 104 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. For more information on the SCLM database contents utility function, see "Database Contents Utility" on page 74.

Command Invocation Format

```
FLMCMD DBUTIL,project,[prj_def]
              ,[acct_group1|*],[acct_group2]
              ,[acct_group3],[acct_group4]
              ,[acct_group5],[acct_group6]
              ,[acct_type|*],[acct_member|*]
              ,[authcode|*],[change_code|*]
              ,[change_group|*],[change_userid|*]
              ,[language|*],[YES|NO]
              ,[ACCT|BMAP|*]
              ,[IN|OUT|*]
              ,[arch_group],[arch_type],[arch_member]
              ,[EXTENDED|NORMAL|SUBUNIT]
              ,[YES|NO]
              ,[YES|NO]
              ,[report_name],[dd_msgs]
              ,[dd_rept],[dd_tailor]
              ,[report_line]
```

Call Invocation Format

You cannot use call procedures to start this service.

Parameters

project
The project name. The maximum parameter length is eight characters.

prj_def
The project definition name to be used for the data extraction. It defaults to the project. The maximum parameter length is eight characters.

acct_group1 - acct_group6|*_
The accounting group associated with the accounting member. The maximum parameter length is eight characters. You can specify up to six individual acct_groups, an asterisk for all, or up to six valid patterns. For more information on patterns see "Specifying Selection Criteria" on page 75.

acct_type|*_
One of a set of partitioned data sets, which makes up a group associated with the accounting record and which contains members of a particular data type. The maximum parameter length is eight characters. You can specify an individual acct_type, an asterisk for all of them, or a valid pattern. For more information on patterns see "Specifying Selection Criteria" on page 75.

acct_member|*_
The discrete element of a project database, which represents a single data type of a software component. The maximum parameter length is eight characters. You can specify an individual acct_member, an asterisk for all of them, or a valid pattern. For more information on patterns see "Specifying Selection Criteria" on page 75.

authcode|*_
The current authorization code for the member. The maximum parameter length is eight characters. You can specify an individual authcode, an asterisk for all of them, or a valid pattern. For more information on patterns see "Specifying Selection Criteria" on page 75.

change_code|*_
A change code you assign to indicate why you are updating the member. The maximum parameter length is eight characters. You can specify an individual change_code, an asterisk for all of them, or a valid pattern. For more information on patterns see "Specifying Selection Criteria" on page 75.

change_group|*_
The name of the group in which the member was last updated. The maximum parameter length is eight characters. You can specify an individual change_group, an asterisk for all of them, or a valid pattern. For more information on patterns see "Specifying Selection Criteria" on page 75.

change_userid|*_
The user ID of the person who made the last update to the member. The maximum parameter length is eight characters. You can specify an individual change_userid, an asterisk for all of them, or a valid pattern. For more information on patterns see "Specifying Selection Criteria" on page 75.

language|*
The language of the member. The maximum parameter length is eight characters. You can specify an individual language, an asterisk for all of them, or a valid pattern. For more information on patterns see "Specifying Selection Criteria" on page 75.

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. See "Accounting Information Fields" on page 76 for more information.

ACCT|BMAP|*
Specify the following data type to report on:

ACCT	Accounting information
BMAP	Build map information
*	Build map and accounting information.

The maximum parameter length is 24 characters.

IN|OUT|*
Specify the following 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.

arch_group
The group used to identify the lowest level in the hierarchy where the architecture begins. The maximum parameter length is eight characters.

arch_type
The type containing the architecture definition that controls the selected members. The maximum parameter length is eight characters.

arch_member
The member containing the architecture definition that controls the selected members. The maximum parameter length is eight 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.

YES|NO
Specify YES to include page header information in the tailored data set. The maximum parameter length is 24 characters.

YES|NO
Specify YES to sum numeric data fields and to show the sum totals in the tailored data set. The maximum parameter length is 24 characters.

report_name
The title of the report to be written in the tailored data set. The maximum parameter length is 35 characters.

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 with the following attributes: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is eight 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 with the following attributes: RECFM=FBA, LRECL=80, BLKSIZE=3120. The maximum parameter length is eight 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 data set. Otherwise, before you call the DBUTIL service, you must allocate the ddname with the following attributes: RECFM=F,V,FB,VB; LRECL = 512 (maximum). The maximum parameter length is eight 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.

If you use the SCLM @@FLM$XN or @@FLM$UD variables, keep in mind that their values can exceed eight characters. Place these variables at the end of the report line to ensure that the columns in the report line up evenly.

The default value for the report_line is the following:

```
@@FLMMBR @@FLMLAN @@FLMCML @@FLMNCL @@FLMBLL @@FLMTLS @@FLMCMS @@FLMNCS
```

Return Codes

Possible return codes are:

0 Normal completion. See the dd_msgs parameter description for more details.

4 Warning condition. See the dd_msgs parameter description for more details.

8 Error condition. See the dd_msgs parameter description for more details.

> 8 Severe error condition and SCLM does not produce messages. See "Return Codes" on page 157 for a description of the return code.

Example

This example calls the DBUTIL service.

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 report 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. For more information on accounting records and build maps, see "Accounting Record" on page 55.

If you delete a member from a development group, delete the same member in the next higher non-key group if it exists there.

Command Invocation Format

```
FLMCMD DELETE,project
              ,[prj_def]
              ,group
              ,type
              ,member
              ,access_key
              ,[ACCT|BMAP|TEXT]
```

Call Invocation Format

```
lastrc := FLMLNK('DELETE',sclm_id
                          ,group
                          ,type
                          ,member
                          ,access_key
                          ,{ACCT|BMAP|TEXT}
                          ,$msg_array);
```

Parameters

project
The project name. The maximum parameter length is eight characters.

prj_def
The project definition name to be used for the delete. It defaults to the project. The maximum parameter length is eight 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 eight characters.

group
The group in which the delete is to occur. The maximum parameter length is eight characters.

type
The type containing the member, accounting record, and/or build map to be deleted. The maximum parameter length is eight characters.

member
The name of the member, accounting record, and/or build map to be deleted. The maximum parameter length is eight 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 eight characters. For more information on access keys, see "Edit Function" on page 10.

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.

$msg_array
An output parameter pointing to the message array. See "$msg_array" on page 110 for more details.

Return Codes

Possible return codes are:

0 Normal completion.

4 Warning condition. The member, accounting record, and/or build map were not found.

8 Error condition. See the $msg_array parameter for more details.

20 Severe error condition. SCLM does not produce messages because the SCLM ID is invalid.

24 Severe error condition. SCLM does not produce messages because SCLM services have not been initialized. See "START—Generate an Application ID for a Services Session" on page 157 for information on initializing an SCLM services session.

32 Severe error condition. SCLM does not produce messages for one of the following reasons:

- You requested an invalid service.
- You supplied an invalid parameter list for the requested service.
- The version of the FLMLNK subroutine does not match the version of the SCLM services module (for future use).

Examples

These examples call the DELETE service.

Command Invocation

```
FLMCMD DELETE,PROJ1,,USER1,SOURCE,MODULE2,XXX#04,ACCT
```

This service command deletes the build map and accounting record for the MODULE2 member of the SOURCE type in the USER1 group. The project name is PROJ1. The access key for the member is XXX#04.

Call Invocation

```
lastrc := FLMLNK('DELETE',sclm_id,
                 'USER1','SOURCE','MODULE2',
                 'XXX#04',
                 'ACCT',
                 $msg_array);
```

This service call deletes the accounting record and the build map for the MODULE2 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.

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

```
lastrc := FLMLNK('END',appl_id
                       ,msg_line);
```

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 eight characters.

msg_line
An output parameter that has a buffer containing the END service error message. The maximum parameter length is 80 characters.

Return Codes

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.

24 Severe error condition. SCLM does not produce messages because SCLM services have not been initialized. See "START—Generate an Application ID for a Services Session" on page 157 for information on initializing an SCLM services session.

32 Severe error condition. SCLM does not produce messages for one of the following reasons:

- You requested an invalid service.
- You supplied an invalid parameter list for the requested service.
- The version of the FLMLNK subroutine does not match the version of the SCLM services module (for future use).

Example

This example calls the END service.

Call Invocation

```
lastrc := FLMLNK('END',appl_id,msg_line);
```

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.

FREE—Free an SCLM ID from its Association with a Database

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 the project database 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

```
lastrc := FLMLNK('FREE',sclm_id
                        ,msg_line);
```

Parameters

sclm_id
The SCLM ID to be freed. The INIT service must generate the SCLM ID. The maximum parameter length is eight characters.

msg_line
An output parameter that is a buffer containing the FREE service error message. The maximum parameter length is 80 characters.

Return Codes

Possible return codes are:

0 Normal completion.

8 Error condition. See the msg_line parameter description for more details.

24 Severe error condition. SCLM does not produce messages because SCLM services have not been initialized. See "START—Generate an Application ID for a Services Session" on page 157 for information on initializing an SCLM services session.

32 Severe error condition. SCLM does not produce messages for one of the following reasons:

- You requested an invalid service.
- You supplied an invalid parameter list for the requested service.
- The version of the FLMLNK subroutine does not match the version of the SCLM services module (for future use).

Example

This example calls the FREE service.

Call Invocation

```
lastrc := FLMLNK('FREE',sclm_id,msg_line);
```

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.

INIT—Generate an SCLM ID for a Database

The INIT service initializes an SCLM ID. During this process, it also initializes both the specified project definition and the project database. 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.

Command Invocation Format

You cannot use command procedures to call this service.

Call Invocation Format

```
lastrc := FLMLNK('INIT',appl_id
                        ,project
                        ,prj_def
                        ,sclm_id
                        ,msg_line);
```

Parameters

appl_id
The application ID to which the generated SCLM ID is to be associated. The application ID must be generated by the START service. The maximum parameter length is eight characters.

project
The project name. The maximum parameter length is eight characters.

prj_def
The project definition name to be initialized for the SCLM ID. The maximum parameter length is eight 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 eight characters.

msg_line
An output parameter that is a buffer containing the INIT service error message. The maximum parameter length is 80 characters.

Return Codes

Possible return codes are:

0 Normal completion.

8 Error condition. See the msg_line parameter description for more details.

24 Severe error condition. SCLM does not produce messages because SCLM services have not been initialized. See "START—Generate an Application ID for a Services Session" on page 157 for information on initializing an SCLM services session.

32 Severe error condition. SCLM does not produce messages for one of the following reasons:

- You requested an invalid service.
- You supplied an invalid parameter list for the requested service.
- The version of the FLMLNK subroutine does not match the version of the SCLM services module (for future use).

Example

This example calls the INIT service.

Call Invocation

```
lastrc := FLMLNK('INIT',appl_id,'PROJ1','PROJ1',sclm_id,msg_line);
```

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 private library or assigns the member an access key, or both. Locking a member guarantees you exclusive use of the member until you either unlock or promote it. Locking a member also ensures that updates to the member can occur only in the specified private library until you unlock or promote the member. The member to be locked does not have to exist in a private library or anywhere in the hierarchy.

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. For an explanation on using access keys, see "Development Scenario" on page 274. 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.

For more information on the LOCK service and access keys, see "Edit Function" on page 10.

Command Invocation Format

```
FLMCMD LOCK,project
           ,[prj_def]
           ,group
           ,type
           ,member
           ,[authcode]
           ,[access_key]
           ,[userid]
```

Call Invocation Format

```
lastrc := FLMLNK('LOCK',sclm_id
                        ,group
                        ,type
                        ,member
                        ,authcode
                        ,access_key
                        ,[userid|' ']
                        ,found_group
                        ,max_prom_group
                        ,$acct_info
                        ,$list_info
                        ,$msg_array);
```

Parameters

project
The project name. The maximum parameter length is eight characters.

prj_def
The project definition name to be used for the lock. It defaults to project. The maximum parameter length is eight 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 eight characters.

group
The group in which the member is to be locked. The specified group must be a private library. The maximum parameter length is eight characters.

type
The type containing the member to be locked. The maximum parameter length is eight characters.

member
The member to be locked. The maximum parameter length is eight characters.

authcode
The authorization code to be used for the lock. SCLM uses the authorization code for the verification steps described for the LOCK service in "Edit Function" on page 10. If you do not supply an authcode or a blank 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 eight 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. For more information on access keys, see "Edit Function" on page 10.

userid
User ID of the person requesting the lock. It defaults to the current system user ID. The maximum parameter length is eight 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 eight characters.

max_prom_group
An output parameter that indicates the highest group in the hierarchy that the member can be promoted to. This member's maximum promotable group is based on the authorization code you use for the lock. The maximum parameter length is eight characters.

$acct_info
An output parameter pointing to a record containing the static portion of the member's accounting record. See "$acct_info" on page 111 for more details.

$list_info
An output parameter pointing to an array of records that contains the dynamic portion of the member's accounting record. See "$list_info" on page 112 for more details.

$msg_array
An output parameter pointing to the message array. See "$msg_array" on page 110 for more details.

Return Codes

Possible return codes are:

0 Normal completion.

8 Error condition. See the $msg_array parameter description for more details.

20 Severe error condition. SCLM does not produce messages because the SCLM ID is invalid.

24 Severe error condition. SCLM does not produce messages because SCLM services have not been initialized. See "START—Generate an Application ID for a Services Session" on page 157 for information on initializing an SCLM services session.

32 Severe error condition. SCLM does not produce messages for one of the following reasons:

- You requested an invalid service.
- You supplied an invalid parameter list for the requested service.
- The version of the FLMLNK subroutine does not match the version of the SCLM services module (for future use).

Examples

These examples call the LOCK service.

Command Invocation

```
FLMCMD LOCK,PROJ1,,USER1,SOURCE,MODULE2,,XXX#04
```

This service command locks the MODULE2 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 userid parameters are defaults.

Call Invocation

```
lastrc := FLMLNK('LOCK',sclm_id,
                 'USER1','SOURCE','MODULE2',
                 'TESTAC',
                 'XXX#04',
                 'USERID',
                 found_group, max_prom_group,
                 $acct_info,$list_info,$msg_array);
```

This service call locks the MODULE2 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.

PARSE—Parse a Member for Statistical and Dependency Information

The PARSE service parses a member for statistical information 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. For more information on the PARSE service, see "Edit Function" on page 10.

If the ISPF/PDF pack mode is on, the parser that was supplied with SCLM returns statistical values reflecting the pack mode. For instance, the number of comment and noncomment lines stored in the accounting records is less than the number of comment and noncomment lines that appear when you display the member. However, the number of comment and noncomment statements is the same when you parse the member.

The member to be parsed does not have to be locked nor does it have to reside in a private library.

Command Invocation Format

You cannot use command procedures to call this service.

Call Invocation Format

```
lastrc := FLMLNK('PARSE',sclm_id
                        ,group
                        ,type
                        ,member
                        ,language
                        ,{'Y'|'N'}
                        ,ddname
                        ,$stats_info
                        ,$list_info
                        ,$msg_array);
```

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 eight characters.

group
The group in which the member is to be parsed. The maximum parameter length is eight characters. Note that a member can be parsed in any group; the specified group does not have to be a private library.

type
The type containing the member to be parsed. The maximum parameter length is eight characters.

member
The member to be parsed. The maximum parameter length is eight characters.

language
The language of the member to be parsed. The maximum parameter length is eight characters.

Y|N
Y indicates parser listings to be copied to the ddname parameter if parser errors occur. N indicates all parser listings 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 may or may not produce a listing. See "Invoking User-Defined Parsers" on page 239 for more information on project-defined parsers.

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 eight characters.

If the parser for the specified language does not produce a listing, you should 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 may or may not produce a listing. See "Invoking User-Defined Parsers" on page 239 for more information on project-defined parsers.

$stats_info
An output parameter pointing to a record containing the member's statistical information derived from parsing the member. See "$stats_info" on page 112 for more details.

$list_info
An output parameter pointing to an array of records that contains the member's include, compool, compilation unit, change code, and user entry information derived from parsing the member. See "$list_info" on page 112 for more details.

$msg_array
An output parameter pointing to the message array. See "$msg_array" on page 110 for more details.

Return Codes

Possible return codes are:

0 Normal completion.

4 Warning condition. A parser error occurred.

8 Error condition. See the $msg_array parameter description for more details.

20 Severe error condition. SCLM does not produce messages because the SCLM ID is invalid.

24 Severe error condition. SCLM does not produce messages because SCLM services have not been initialized. See "START—Generate an Application ID for a Services Session" on page 157 for information on initializing an SCLM services session.

32 Severe error condition. SCLM does not produce messages for one of the following reasons:

- You requested an invalid service.
- You supplied an invalid parameter list for the requested service.
- The version of the FLMLNK subroutine does not match the version of the SCLM services module (for future use).

Example

This example calls the PARSE service.

Call Invocation

```
lastrc := FLMLNK('PARSE',sclm_id,
                 'USER1','SOURCE','MODULE2',
                 'PASCAL',
                 'Y',
                 'PARSEDD ',
                 $stats_info,$list_info,$msg_array);
```

This service call parses the MODULE2 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 successfully complete 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.

For more information on the SCLM promote function, see "Promote Function" on page 17.

Command Invocation Format

```
FLMCMD PROMOTE,project
              ,[prj_def]
              ,group
              ,type
              ,member
              ,[userid|' ']
              ,[E|N|S]
              ,[C|R|U]
              ,[dd_prommsgs]
              ,[dd_promrept]
              ,[dd_promexit]
              ,[dd_copyerr]
```

Call Invocation Format

```
lastrc := FLMLNK('PROMOTE',sclm_id
                          ,group,type,member
                          ,[userid|' ']
                          ,{E|N|S}
                          ,{C|R|U}
                          ,dd_prommsgs,dd_promrept
                          ,dd_promexit,dd_copyerr);
```

Parameters

project
The project name. The maximum parameter length is eight characters.

prj_def
The project definition name to be used for the promote. It defaults to project. The maximum parameter length is eight 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 eight characters.

group
The group the promote occurs from. The maximum parameter length is eight characters.

type
The type containing the member to be promoted. The maximum parameter length is eight characters.

member
The name of the architecture member or source member to be promoted. The maximum parameter length is eight characters.

userid
The user ID of the person requesting the promote. It defaults to the current system user ID. The maximum parameter length is eight characters.

E|N|S
Indicates the promote scope (E = extended, N = normal, S = subunit). The maximum parameter length is 24 characters. See the field definitions in "Promote (Option 5)" on page 94 for more details.

C|R|U
Indicates the promote mode (C = conditional, R = report, U = unconditional). The maximum parameter length is 24 characters. See the field definitions in "Promote (Option 5)" on page 94 for more details.

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 with the following attributes: `DISP=MOD`, `RECFM=F`, `LRECL=80`, `BLKSIZE=80`. The maximum parameter length is eight characters. See "Promote Messages" on page 19 for more details on promote messages.

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 with the following attributes: `RECFM=FBA`, `LRECL=80`, `BLKSIZE=3120`. The maximum parameter length is eight characters. See "Promote Report" on page 19 for more details.

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 administrator 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 with the following attributes: RECFM=FB, LRECL=160, BLKSIZE=3200. The maximum parameter length is eight 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, you must allocate the ddname before you call the PROMOTE service. If you allocate the copy error ddname, you should allocate it to da(*), the default output device. The maximum parameter length is eight characters.

Return Codes

Possible return codes are:

0 Normal completion. See the dd_prommsgs parameter description for more details.

8 Error condition. See the dd_prommsgs parameter description for more details.

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.

20 Severe error condition. SCLM does not produce messages because the SCLM ID is invalid.

24 Severe error condition. SCLM does not produce messages because SCLM services have not been initialized. See "START—Generate an Application ID for a Services Session" on page 157 for information on initializing an SCLM services session.

32 Severe error condition. SCLM does not produce messages for one of the following reasons:

- You requested an invalid service.
- You supplied an invalid parameter list for the requested service.
- The version of the FLMLNK subroutine does not match the version of the SCLM services module (for future use).

Examples

These examples call the PROMOTE service.

Command Invocation

```
FLMCMD PROMOTE,PROJ1,,USER1,ARCHDEF,CMOD1,,,U
```

This service command promotes the CMOD1 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.

Call Invocation

```
lastrc := FLMLNK('PROMOTE',sclm_id,
                 'USER1','ARCHDEF','CMOD1',
                 'USERID',
                 'E',
                 'R',
                 'PROMMSGS','PROMREPT','PROMEXIT','COPYDD  ');
```

This service call performs a report-only promote on the CMOD1 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 generates an SCLM architecture report. The report consists of the architecture definition. For more information on the SCLM architecture report, see "Architecture Report" on page 83.

Command Invocation Format

```
FLMCMD RPTARCH,project,[prj_def]
                      ,group
                      ,type
                      ,member
                      ,[HL|LEC|CC|GEN|TOP SOURCE|NONE]
                      ,dd_rptmsgs
                      ,dd_rptrept
```

Call Invocation Format

You cannot use call procedures to start this service.

Parameters

project
The project name. The maximum parameter length is eight characters.

prj_def
The project definition name to be used for generating the architecture report. It defaults to project. The maximum parameter length is eight characters.

group
The group the report is to be generated from. The maximum parameter length is eight characters.

type
The type containing the member to be reported on. The maximum parameter length is eight characters.

member
The member to be reported on. The maximum parameter length is eight 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.

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 with the following attributes: `RECFM=F, LRECL=80, BLKSIZE=80`. The maximum parameter length is eight 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 with the following attributes: `RECFM=FBA, LRECL=80, BLKSIZE=3120`. The maximum parameter length is eight characters. See "Architecture Report" on page 83 for more details on architecture reports.

Return Codes

Possible return codes are:

0 Normal completion. See the dd_rptmsgs parameter description for more details.

8 Error condition. See the dd_rptmsgs parameter description for more details.

> 8 Severe error condition; SCLM does not produce messages. See "Return Codes" on page 157 for a description of the return code.

Example

This example calls the RPTARCH service.

Command Invocation

```
FLMCMD RPTARCH,PROJ1,,USER1,SOURCE,MODULE1,NONE
```

This service command generates an architecture report for the MODULE1 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 private library you specify. (The LOCK, SAVE, or STORE service may have completed successfully for the member.) 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. Lock the member using the LOCK service. (The member may or may not yet exist.)
2. Update or create the member.
3. Start the SAVE service to parse the member and store the member's statistical, dependency, and historical information.

For more information on 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 preexisting historical information, such as user entry data, do not invoke the SAVE service. Use the LOCK, PARSE, and STORE services instead.

Command Invocation Format

```
FLMCMD SAVE,project,[prj_def]

       ,group,type,member

       ,[authcode],[access_key]

       ,[userid],[language]

       ,[Y|N]

       ,[ddname],[C|U]

       ,[C|U],[change_code]
```

Call Invocation Format

```
lastrc := FLMLNK('SAVE',sclm_id
                        ,group,type,member
                        ,authcode,access_key
                        ,[userid|' '],language
                        ,{Y|N}
                        ,ddname
                        ,{C|U}
                        ,{C|U}
                        ,{Y|N}
                        ,$list_info
                        ,max_prom_group
                        ,$msg_array);
```

Parameters

project
The project name. The maximum parameter length is eight characters.

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 eight characters.

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 eight characters.

group
The group in which the lock, parse, and store are to occur. The specified group must be a private library. The maximum parameter length is eight characters.

type
The type containing the member. The maximum parameter length is eight characters.

member
The member to be locked and parsed, and whose accounting information is to be stored. The maximum parameter length is eight characters.

authcode
The authorization code to be used for the lock. SCLM uses the authorization code for the verification steps described in "LOCK" on page 10. If you do not supply an authcode or a blank 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 eight characters.

access_key
The access key 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. For more information on access keys, see "Edit Function" on page 10.

userid
User ID of the person requesting the SAVE service. It defaults to the current system user ID. The maximum parameter length is eight characters.

language
The language of the member. The maximum parameter length is eight characters. You must specify the language the first time you save a member.

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 may or may not produce a listing. See "Invoking User-Defined Parsers" on page 239 for more information on project-defined parsers.

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 eight characters.

If the parser for the specified language's parser 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 may or may not produce a listing. See "Invoking User-Defined Parsers" on page 239 for more information on project-defined parsers.

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.

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.

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 administrator if your project is using a change code verification routine. See "Change Code Verification Routines" on page 261 for more details.

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 eight characters.

$list_info
An 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 may 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 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 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.

See "Pointer Parameters" on page 110 for more details on $list_info.

max_prom_group
An output parameter indicating the highest group in the hierarchy that the member can be promoted to. 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 eight characters.

$msg_array
An output parameter pointing to the message array. See "$msg_array" on page 110 for more details.

Return Codes

Possible return codes are:

0 Normal completion.

4 Warning condition. See the $msg_array parameter description for more details.

8 Error condition. See the $msg_array parameter description for more details.

20 Severe error condition. SCLM does not produce messages because the SCLM ID is invalid.

24 Severe error condition. SCLM does not produce messages because SCLM services have not been initialized. See "START—Generate an Application ID for a Services Session" on page 157 for information on initializing an SCLM services session.

32 Severe error condition. SCLM does not produce messages for one of the following reasons:

- You requested an invalid service.
- You supplied an invalid parameter list for the requested service.
- The version of the FLMLNK subroutine does not match the version of the SCLM services module (for future use).

Examples

These examples call the SAVE service.

Command Invocation

```
FLMCMD SAVE,PROJ1,,USER1,SOURCE,MODULE1,,XXX#05,,PASCAL,,,,,CC001234
```

This service command locks, parses, and stores the information for the member MODULE1 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.

Call Invocation

```
$list_info := NIL;  (* Sets the buffer address to X'00000000' *)

lastrc := FLMLNK('SAVE',sclm_id,
                 'USER1','SOURCE','MODULE1',
                 'TESTAC','XXX#05',
                 'USERID','PASCAL',
                 'Y','PARSEDD',
                 'U',
                 'C',
                 'Y',
                 $list_info,max_prom_group,$msg_array);
```

This service call locks, parses, and stores the information for member MODULE1 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.

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. Later, 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

```
lastrc := FLMLNK('START',appl_id);
```

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 eight characters.

Return Codes

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 National Language Support (NLS) 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 NLS table or the SCLM I/O load module.

32 Severe error condition. SCLM does not produce messages for one of the following reasons:

- You requested an invalid service.
- You supplied an invalid parameter list for the requested service.
- The version of the FLMLNK subroutine does not match the version of the SCLM services module (for future use).

Example

This example calls the START service.

Call Invocation

```
lastrc := FLMLNK('START',appl_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 (using either the PARSE service or your own parser), 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 private 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.

Use the following equation to determine the maximum amount of accounting information that you can store for an SCLM source member:

```
14(Number of compilation units) + 16(Number of user entries) +
Number of includes + Number of compools +
3(Number of change codes) <= 3980
```

For more information on the STORE service, see "STORE" on page 12.

Command Invocation Format

You cannot use command procedures to call this service.

Call Invocation Format

```
lastrc := FLMLNK('STORE',sclm_id
                        ,group,type,member
                        ,access_key
                        ,language
                        ,[userid|' ']
                        ,{'C'|'U'}
                        ,{'Y'|'N'}
                        ,$stats_info,$list_info
                        ,$msg_array);
```

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 eight characters.

group
The group in which the store is to occur. The specified group must be a private library. The maximum parameter length is eight characters.

type
The type containing the member whose information is to be stored. The maximum parameter length is eight characters.

member
The member whose information is to be stored. The maximum parameter length is eight 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. For more information on access keys, see "Edit Function" on page 10.

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 eight characters.

userid
The user ID of the person requesting the STORE service. Defaults to the current system user ID. The maximum parameter length is eight 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 administrator if your project is using a change code verification routine. If it is not, specify N. See "Change Code Verification Routines" on page 261 for more information.

$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 Parameters" on page 110 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, compool, compilation unit, change code, and user entry information. If the member has none of this information, you can pass a fullword zero buffer address.

All include, compool, compilation unit, 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, compool, and compilation unit entries in $list_info determines 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 Parameters" on page 110 for more information on the $list_info parameter and copying the array contents.

$msg_array
An output parameter pointing to the message array. See "Pointer Parameters" on page 110 for more information on $msg_array.

Return Codes

Possible return codes are:

0 Normal completion.

4 Warning condition. See the $msg_array parameter description for more details.

8 Error condition. See $msg_array parameter description for more details.

20 Severe error condition. SCLM does not produce messages because the SCLM ID is invalid.

24 Severe error condition. SCLM does not produce messages because SCLM services have not been initialized. See "START—Generate an Application ID for a Services Session" on page 157 for information on initializing an SCLM services session.

32 Severe error condition. SCLM does not produce messages for one of the following reasons:

- You requested an invalid service.
- You supplied an invalid parameter list for the requested service.
- The version of the FLMLNK subroutine does not match the version of the SCLM services module (for future use).

Example

This example calls the STORE service.

Call Invocation

```
lastrc := FLMLNK('STORE',sclm_id,
                 'USER1','SOURCE','MODULE2',
                 'XXX#04',
                 'PASCAL',
                 ' ',
                 'C',
                 'Y',
                 $stats_info,$list_info,$msg_array);
```

This service call stores the statistical and dependency information (obtained from $stats_info and $list_info) in member MODULE2's accounting record in the project database. The sclm_id parameter contains a valid SCLM ID returned from the INIT service.

The member MODULE2 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 Private Library

The UNLOCK service resets the member access key to blank. If you unlock a member, you cannot guarantee exclusive use of the member in a private library.

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 on the LOCK service and access keys, see "LOCK" on page 10.

Command Invocation Format

```
FLMCMD UNLOCK,project
             ,[prj_def]
             ,group
             ,type
             ,member
             ,[access_key]
```

Call Invocation Format

```
lastrc := FLMLNK('UNLOCK',sclm_id
                         ,group
                         ,type
                         ,member
                         ,access_key
                         ,$msg_array);
```

Parameters

project
The project name. The maximum parameter length is eight characters.

prj_def
The project definition name to be used for the unlock. It defaults to project. The maximum parameter length is eight 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 eight characters.

group
The group in which the member is to be unlocked. The specified group must be a private library. The maximum parameter length is eight characters.

type
The type containing the member to be unlocked. The maximum parameter length is eight characters.

member
The member to be unlocked. The maximum parameter length is eight 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 more information on access keys, see "Edit Function" on page 10.

$msg_array
An output parameter pointing to the message array. See "$msg_array" on page 110 for more details on $msg_array.

Return Codes

Possible return codes are:

0 Normal completion.

4 Warning condition. See the $msg_array parameter description for more details.

8 Error condition. See the $msg_array parameter description for more details.

20 Severe error condition. SCLM does not produce messages because the SCLM ID is invalid.

24 Severe error condition. SCLM does not produce messages because SCLM services have not been initialized. See "START—Generate an Application ID for a Services Session" on page 157 for information on initializing an SCLM services session.

32 Severe error condition. SCLM does not produce messages for one of the following reasons:

- You requested an invalid service.
- You supplied an invalid parameter list for the requested service.
- The version of the FLMLNK subroutine does not match the version of the SCLM services module (for future use).

Examples

These examples call the UNLOCK service.

Command Invocation

```
FLMCMD UNLOCK,PROJ1,,USER1,SOURCE,MODULE1,XXX#05
```

This service command unlocks the MODULE1 member of the SOURCE type in the USER1 group. The project name is PROJ1. The access key value for the member is XXX#05.

Call Invocation

```
lastrc := FLMLNK('UNLOCK',sclm_id,
                 'USER1','SOURCE','MODULE1',
                 'XXX#05',
                 $msg_array);
```

This service call unlocks the MODULE1 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.

Chapter 6. A Sample Program Using SCLM Services

This chapter contains an example of Pascal program invocations that call the following SCLM services in this order:

- START
- INIT
- LOCK
- PARSE
- STORE
- BUILD
- FREE
- END.

Pascal Example

The following is a sample Pascal program you can use 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.

Main Program SERV1

```
PROGRAM SERV1 ;

(*********************************************************************)
(*                                                                   *)
(* This program allows you to call SCLM services from a              *)
(* Pascal program.                                                   *)
(*                                                                   *)
(* 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)  *)
(*********************************************************************)
```

© Copyright IBM Corp. 1989, 1990

```
(*********************************************************************)
(*              Declare program and interface constants              *)
(*********************************************************************)
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 SERV1D ;

(* Include SCLM Interface procedure definitions. *)
%INCLUDE SERV1S ;
```

```
(*********************************************************************)
(*              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                 ;
  dd_bldmsgs                : char8                 ;
  dd_bldrpt                 : char8                 ;
  dd_bldlist                : char8                 ;
  dd_bldexit                : 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 this user ID',
           ' (which will be the private library to process).' );
  READLN  (userid)   ;
  WRITELN ('Enter the name of the type to process.');
  READLN  (pds_type) ;
  WRITELN ('Enter the name of the member to undergo processing.');
  READLN  (member)   ;
  WRITELN ('Enter the language of the source member to register.');
  READLN  (language) ;

  (* Default the group to process to be the user ID. *)
  group := userid ;

  (* 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. *)
project_def := project ;
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_bldrpt  *)
             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 undergone a build.' );
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 SERV1D

```
(*********************************************************************)
(*                 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 ;  (* type = ALFA *)
  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 ;
  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               : char6   ;
  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             : char6   ;
  creation_time             : char6   ;
  map_date                  : char6   ;
  map_time                  : char6   ;
  predecessor_date          : char6   ;
  predecessor_time          : char6   ;
  promote_date              : char6   ;
  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_compools        : 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         : char6 ;
    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 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         );
  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 SERV1S

```
(*********************************************************************)
(*          SCLM SERVICE INTERFACE PROCEDURE DEFINITIONS             *)
(*********************************************************************)

(*********************************************************************)
(*                  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 *)
```

Part 2. Project Administration

© Copyright IBM Corp. 1989, 1990

Chapter 7. Defining the Project

The primary concern of a project administrator is generating an SCLM *project definition*. A project definition enables SCLM to function for an individual project (with project-specific customization). Further, a project definition defines the development environment for a project by identifying languages and processors, types, groups, and hierarchies. It also defines *authorization codes,* which are used to control users' authority to update and promote members within a hierarchy, and internal data sets. *Internal data sets* contain accounting, statistical, dependency, status, and tracking information about all controlled members.

You can generate more than one project definition for a project. The primary project definition for an SCLM project is the *default* project definition. All other project definitions for a project are *alternate* project definitions. You should, however, keep the number of project definitions for a project to a minimum, preferably a single project definition. See "Step 10: Assemble and Link Project Definition" on page 206 for more information.

To generate a project definition, take the following steps:

1. Determine the structure of the project database.
2. Identify types of data to be maintained.
3. Establish authorization codes.
4. Create PROJDEFS data set.
5. Allocate the project data sets.
6. Protect the data sets.
7. Specify the project definition.
8. Modify the language definitions.
9. Modify the control options.
10. Assemble and link the project definition.
11. Build INFO member.

Step 1: Determine Database Structure

The project manager usually participates in determining the database structure for the project. As project administrator, you are responsible for generating and updating the project definition to accommodate project requirements. The following paragraphs discuss methods for defining an efficient database.

First, assess how many layers and groups you need for the project. Each layer corresponds to a phase of development. Large projects usually require many layers while smaller projects require fewer layers.

Next, diagram the database to illustrate the development strategy for the project. You must design the database in a tree structure, where the predecessor or release layer is the root, and the test, integration, and development layers are the branches. Each group can have no more than one parent group. Figure 1 on page 7 shows an example of a common SCLM database structure.

© Copyright IBM Corp. 1989, 1990

SCLM does not restrict the number of groups in a layer or the names of the groups. Each user can be assigned to an individual development group or share a group. If you want to have stage groups, as in Figure 1 on page 7, you can individualize them for each developer or group them for several users. Hierarchies are limited to up to 16 key groups. There are also MVS limitations on concatenating data sets that affect a project's database structure. See "Key/Non-Key Groups" on page 8 for details.

Figure 50 shows two more diagrams of common database structures.

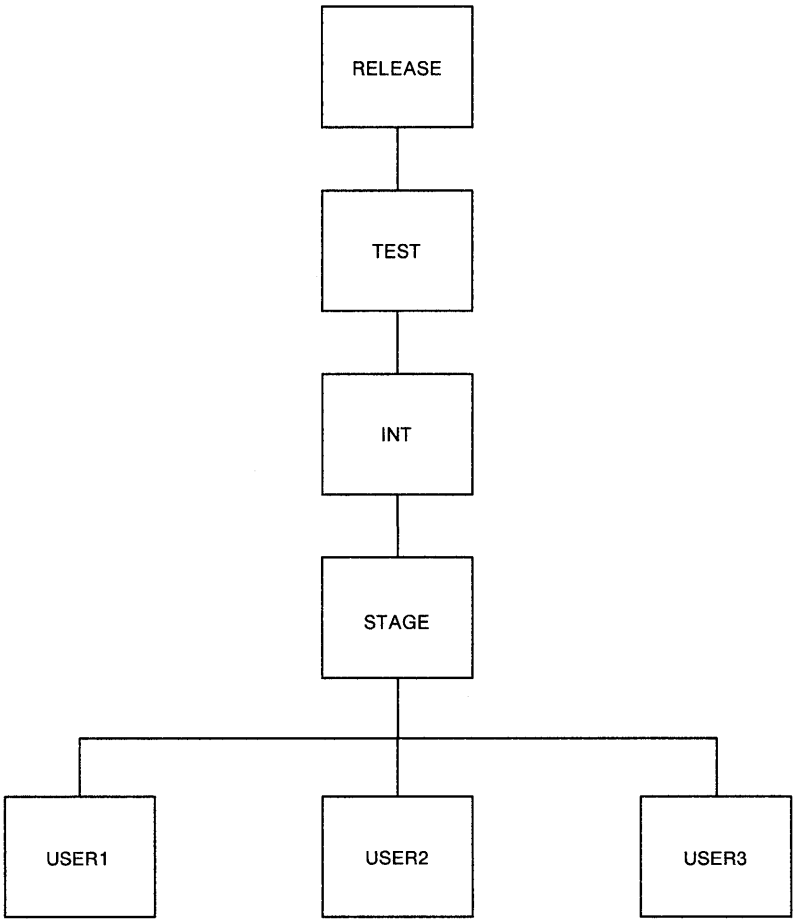


Figure 50 (Part 1 of 2). Example of Other Common SCLM Database Structures

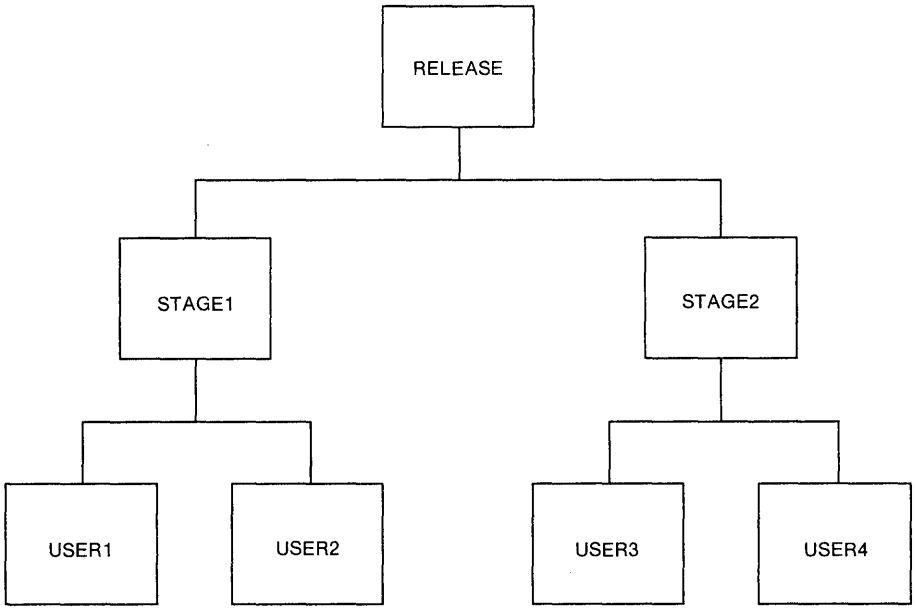


Figure 50 (Part 2 of 2). Example of Other Common SCLM Database Structures

Step 2: Identify Supported Types of Data

SCLM supports the same data that MVS partitioned data sets support. If size permits, you should group similar kinds of data into 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:

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

Similar kinds of data can reside in separate types. For example, you can divide source code into assembler source code and Pascal source code. Thus, you would need to identify an assembler type and a Pascal type.

Step 3: Establish Authorization Codes

You can use authorization codes in SCLM to determine whether a member can be updated in a given group and to control the groups a member can be modified in. Authorization codes can allow temporary copies of a member to exist concurrently in two different groups without creating integrity problems. You can also use authorization codes to prevent certain members from being promoted to a particular group of the hierarchy. You can use a character string up to eight characters long to represent an authorization code.

Defining Authorization Codes for a Group

You define a list of authorization codes for each group in the SCLM hierarchy. You can use these codes to control the contents of the groups in the SCLM hierarchy. A group can have any number of authorization codes associated with it. Only members assigned to one of the authorization codes associated with a group can exist at that group.

Assigning an Authorization Code to a Member

Assign a single authorization code to an editable member when you introduce that version of the member into the SCLM-controlled hierarchy using the migration utility, SCLM editor, LOCK service, or SAVE service. You must assign an authorization code to all versions of every editable member. If you do not assign an authorization code to a new version, SCLM uses the authorization code of the most current version of the member already in the hierarchy. If no previous version exists, SCLM uses a default authorization code defined in the project definition. Authorization codes do not affect non-editable members.

You can change the authorization code assigned to a version of a member by using the SCLM library utility.

The use of authorization codes is necessary in certain circumstances but can be misused if not controlled properly. For other possible uses of authorization codes, see "Authorization Code Usage" on page 252. In most cases, you should use a single authorization code for the entire hierarchy. The example on page 205 illustrates how a single authorization code is used for the first hierarchy shown in Figure 50 on page 190.

Step 4: Create PROJDEFS Data Set

Define a project database structure to SCLM using a load data set containing a compiled and linked project definition. Name the load data set in the following manner:

```
project_id.PROJDEFS.LOAD
```

where project_id is the name of the project. The complete project definition, once assembled and linked, resides in the project's PROJDEFS data set. The PROJDEFS data set should be protected with Resource Access Control Facility (RACF) or an equivalent security system. As the project administrator, only you should have update authority to the data set.

Allocate a source and object data set with the same naming conventions. For example:

```
project_id.PROJDEFS.SOURCE
```

and

```
project_id.PROJDEFS.OBJ
```

These data sets contain the source and object forms of the project definition.

Step 5: Allocate Project Data Sets

The SCLM database for a project consists of a series of partitioned data sets and a VSAM data set under a single high-level qualifier. All partitioned data set names must conform to SCLM naming conventions. See "Projects, Groups, and Types" on page 6. The VSAM data set requires that the project high-level qualifier be an alias in a VSAM user catalog or system catalog. You should create a VSAM user catalog and alias specifically for the project, although you can use an existing VSAM catalog and alias.

Defining a VSAM User Catalog and Alias

To create a user catalog, you must use Access Method Services (AMS). Use the following form of the DEFINE command to create a new user catalog:

```
DEFINE USERCATALOG (NAME(entryname) VOLUME(volume)
        TRACKS(primary secondary))
```

You can also use the DEFINE command to create a user catalog alias for the high-level qualifier for the project. Use the following form of the DEFINE command:

```
DEFINE ALIAS (NAME(high-group qualifier) RELATE(entryname))
```

Note: For more information on the DEFINE command, see *OS/VS2 Access Method Services*, GC26-3841.

Allocating SCLM Partitioned Data Sets

Form data set names for the database in the following manner:

```
project.group.type
```

SCLM does not restrict the format of a data set. However, you must allocate data sets of the same type with the same attributes. The table below shows a listing of recommended data set attributes for some typical types.

Type	RECFM	LRECL	BLKSIZE
Source	FB	80	3120
Object	FB	80	3200
Load	U	0	6144
Listings	VBA	137	3120
Linkedit Maps	VBA	137	3120
Architecture	FB	80	3120
Other Text	FB	80	3120

Allocate data sets according to expected content. You should allocate all data sets anticipating 60% capacity. Determine directory blocks using the following formula:

```
# Directory Blocks =  (# members) / 5
```

Note: During development or maintenance, developers constantly update data sets, and they usually require additional space. The formula shown merely estimates what is needed and can vary from project to project.

Defining Accounting Data Set (SCLM Internal Data)

The accounting data set for the project must be a VSAM cluster. A *VSAM cluster* is a named structure consisting of a group of related components. You must define the VSAM cluster with the IDCAMS utility. The primary and secondary accounting data set must be defined correctly or else the results are unpredictable. The JCL used to define the accounting data set for the project follows.

```
//jobname  JOB (wkpkg,dpt,bin),'name'
//*****************************************************************
//*
//*  THIS JCL DEFINES A VSAM CLUSTER TO BE USED AS THE SCLM ACCOUNT
//*    DB FOR A GIVEN PROJECT
//*  THE HIGH LEVEL QUALIFIER MUST BE AN ENTRY IN AN ICF USER CATALOG
//*    IN ORDER TO CREATE THIS CLUSTER.
//*  TO SPECIFY THE FILE, CHANGE THE DEFINE CLUSTER IN THE
//*    FOLLOWING MANNER:
//*    1) CHANGE ALL XXX.YYY.ZZZ TO THE DESIRED FILE NAME.
//*        ACCOUNTING FILE NAMES ARE USUALLY CHOSEN IN THE FOLLOWING
//*        MANNER - "PROJECT.ACCOUNT.FILE" (WHICH IS THE DEFAULT
//*        USED IN THE PROJECT DEFINITION IF NONE IS SPECIFIED).
//*    2) MODIFY CYLINDERS (PRIMARY SECONDARY)
//*    3) SPECIFY THE VOLUME ON WHICH IT WILL BE ALLOCATED
//*
//*  A JOB STEP IS THEN PROCESSED TO INITIALIZE THE FILE
//*
//*****************************************************************
//STEP1   EXEC PGM=IDCAMS
//*
//SYSPRINT  DD SYSOUT=H
//*
//SYSIN  DD *
 DELETE 'XXX.YYY.ZZZ' CLUSTER
 DEFINE CLUSTER -
 (NAME('XXX.YYY.ZZZ') CYLINDERS(4 1) VOLUMES(VVVVVV) -
 KEYS(26 0) IMBED RECORDSIZE(264 32000) SHAREOPTIONS(4,3) -
 SPEED UNIQUE SPANNED) -
 INDEX(NAME('XXX.YYY.ZZZ.INDEX')) -
 DATA(NAME('XXX.YYY.ZZZ.DATA') CISZ(2024) FREESPACE(50 50))
/*
//*****************************************************************
//*
//*  INITIALIZE THE ACCOUNTING FILE
//*
//*****************************************************************
//STEP2   EXEC PGM=IDCAMS
//INPUT    DD *
 VSAM FILE INITIALIZATION RECORD
//SYSPRINT  DD SYSOUT=H
//SYSIN    DD *
 REPRO INFILE(INPUT) OUTDATASET('XXX.YYY.ZZZ')
/*
//
```

The JCL shown above establishes a VSAM cluster to store accounting and build map information for a sample project. You can adjust the JCL to allocate additional space by modifying the space parameter (shown as CYLINDERS in the example JCL). See the next section, "Space Computations for the Accounting Data Set Definition," for more information on modifying the space parameter. In the

project definition, you must define the data set name you chose for the accounting data set. See "Step 9: Modify Control Options" on page 202 for more information.

Space Computations for the Accounting Data Set Definition

SCLM stores internal data in VSAM data sets. Compute space parameters according to the specifications below.

Number of members being controlled	Cylinders (3350)
1,000	3
10,000	6

The following chart shows the conversion factors necessary to compute direct access capacities for various devices:

Device	Cylinders	Tracks/ Cylinder	Bytes/ Track	Bytes/ Cylinder	Bytes/ Device (millions)
3350	555	30	19,069	572,070	317.5
3375	959	12	35,616	427,392	409.8
3380	885	15	47,476	712,140	630.2

Defining Cross-Reference Data Set (SCLM Internal Data)

The cross-reference data set for the project must be a VSAM cluster. Use the IDCAMS utility to define the VSAM cluster. The primary and secondary cross-reference data set must be defined correctly or else the results are unpredictable. The following page shows the JCL used to define the cross-reference data set for the project.

```
//jobname  JOB (wkpkg,dpt,bin),'name'
//*****************************************************************
//*
//*  THIS JCL DEFINES A VSAM CLUSTER TO BE USED AS THE SCLM CROSS REF
//*    FOR A GIVEN PROJECT
//*  THE HIGH LEVEL QUALIFIER MUST BE AN ENTRY IN AN ICF USER CATALOG
//*    IN ORDER TO CREATE THIS CLUSTER.
//*  TO SPECIFY THE FILE, CHANGE THE DEFINE CLUSTER IN THE
//*    FOLLOWING MANNER:
//*    1) CHANGE ALL XXX.YYY.ZZZ TO THE DESIRED FILE NAME.
//*        ACCOUNTING FILE NAMES ARE USUALLY CHOSEN IN THE FOLLOWING
//*        MANNER - "PROJECT.CROSSREF.FILE,"
//*    2) MODIFY CYLINDERS (PRIMARY SECONDARY)
//*    3) SPECIFY THE VOLUME 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 *
 DELETE 'XXX.YYY.ZZZ' CLUSTER
 DEFINE CLUSTER -
 (NAME('XXX.YYY.ZZZ') CYLINDERS(4 1) VOLUMES(VVVVVV) -
 KEYS(128 0) IMBED RECORDSIZE(264 32000) SHAREOPTIONS(4,3) -
 SPEED UNIQUE SPANNED) -
 INDEX(NAME('XXX.YYY.ZZZ.INDEX')) -
 DATA(NAME('XXX.YYY.ZZZ.DATA') CISZ(2024) FREESPACE(50 50))
/*
//*****************************************************************
//*
//*  INITIALIZE THE CROSS REF FILE
//*
//*****************************************************************
//STEP2   EXEC PGM=IDCAMS
//INPUT   DD DSN=TEMP.XREF.RECORD,DISP=SHR
//OUTPUT  DD DSN=XXX.YYY.ZZZ,DISP=SHR
//SYSPRINT  DD SYSOUT=H
//SYSIN    DD *
 REPRO INFILE(INPUT) OUTFILE(OUTPUT)
/*
//
```

The JCL above establishes a VSAM cluster to handle cross-reference information for an average sized project. You can adjust the JCL to allocate additional space by modifying the CYLINDERS parameter.

You need to create an input data set (TEMP.XREF.RECORD) before running the sample JCL. The input data set must contain a nonblank record of at least 160 bytes in length to initialize the cross-reference data set. Define the data set name you chose for the cross-reference data set in the main project definition for the project. See "Step 9: Modify Control Options" on page 202 for more information.

Step 6: Protect Project Data Sets

Use a data access control facility to protect your SCLM project data sets. A commonly used security product is RACF.

All SCLM project data sets normally have a universal access READ to allow each developer maximum access to the application software. Development and staging data sets need update authority for the developers of those data sets, while higher-level data sets should be protected so that only developers who control those groups have update authority to the data sets. All developers in the project must have update access to the VSAM accounting data set.

As project administrator, you should restrict the project definition data set so that only you have update authority. All other developers need READ access only to this data set.

For additional information on RACF, refer to *OS/VS2 MVS Resource Access Control Facility (RACF) Command Language Reference,* SC28-0733.

See "Security" on page 271 for a discussion of non-bypassable security for the SCLM database.

Step 7: Specify the Project Definition

Create the project definition by using the SCLM macro set provided with the SCLM product. The six SCLM macros provided are:

FLMABEG Put this macro at the beginning of the project definition. It initializes the project definition by defining the project ID. You can use it only one time.

FLMAGRP Use this macro to define a group of authorization codes. You can use it multiple times.

FLMGROUP Use this macro to define one group in the project database. You can use it multiple times.

FLMTYPE Use this macro to define one type in the project database. You can use it multiple times.

FLMCNTRL Use this macro to specify project-specific control options. You can use it only one time.

FLMAEND Put this macro at the end of the project definition to conclude the project definition. You can use it only one time.

Chapter 8, "SCLM Macros," describes the use of these macros in detail. The example project definition on page 205 is used throughout this chapter as a reference in explaining how to generate the project definition

Note: Because these are S/370 assembler language macros, all rules pertaining to macros apply. Namely, assembler does not support blanks in macro parameters. If a line of code requires more than 71 characters, you must put a continuation character in column 72 and begin the remaining lines at column 16. Refer to *OS Assembler H Language,* GC26-3771, for more information on the use of macros.

Specifying the Project Identifier

Specify the identifier name for the project using the FLMABEG macro. This macro must appear at the top of every project definition you generate. If you want more than one project definition for a project, keep the identifier name in the alternates the same. See "Alternate Project Definitions" on page 257 for more information. The format of this macro is shown in "FLMABEG Macro" on page 210. In the example on page 205, the FLMABEG macro defines project PROJ1.

Defining Authorization Groups

Define authorization groups using the FLMAGRP macro. "FLMAGRP Macro" on page 212 shows the format of this macro. You must use authorization groups when the list of authorization codes defined for a group becomes too large for the group macro. Use a single authorization code for most projects that include normal development activities. The example on page 205 defines only one authorization code. The FLMAGRP macro is for illustration only; it is not required. You can obtain an equivalent result by removing the FLMAGRP macro and changing GRP1 to REL as the authorization code for each group.

Defining Groups and Authorization Codes

Define groups by using the FLMGROUP macro. Each group in the project requires an FLMGROUP statement. "FLMGROUP Macro" on page 223 shows the format of this macro.

The use of authorization codes is necessary in certain circumstances but can be misused if not controlled properly. In most cases, you should use a single authorization code for the entire hierarchy. The example on page 205 shows how this is done for the hierarchy shown in Figure 50 on page 190.

Defining Types

Define types using the FLMTYPE macro. "FLMTYPE Macro" on page 231 shows the format of this macro. In the example project definition depicted on page 205, type ARCHDEF is used to contain architecture members. See Chapter 2, "Architecture Definition," for more information. Maintain a separate architecture definition type. You may determine the actual name for this type.

Step 8: Modify Language Definitions

SCLM includes sample language definitions for the following widely used compilers and linkage editors:

- Ada
- BookMaster
- COBOL
- FORTRAN IV
- JOVIAL
- Pascal
- PL/I Optimizing Compiler
- SCRIPT/VS
- Series/1 assembler
- Series/1 COBOL
- Series/1 EDL
- Series/1 PL/I
- S/370 assembler
- S/370 assembler H.

You can get project support for these languages by copying each language definition member from the SCLM macro set into the data set containing the project definition (see "Step 4: Create PROJDEFS Data Set" on page 192) and by modifying the language definition according to project needs. You may need to modify language definitions depending on variations in compilers from project to project. The following paragraphs provide modification instructions. Put a copy reference for each language definition member in the project definition for each language the project will use. See page 205 for an example.

Creating Language Definitions

Produce language definitions with the following macros:

FLMLANGL Use this macro to specify the language identifier.

FLMTRNSL Use this macro once for each translator to be invoked for a language.

FLMALLOC Use this macro for each data set allocation required by a translator. Specify one for each ddname in the ddname substitution list for a translator. The *ddname substitution list* is a string of ddnames allocated for the translator. The ddnames are in the order you specify. Refer to *MVS/XA Data Administration: Utilities*, GC26-4018, for more information about the ddname list substitution.

FLMCPYLB Use this macro to identify other data sets to be concatenated to a ddname.

FLMSYSLB Use this macro to define a set of data sets by languages which contain project and/or system macros or includes.

FLMCMPLB Use this macro to define a set of data sets by language which contain project compool dependencies. It applies only to languages containing compool dependencies, such as JOVIAL.

Use the language definition macros in a specific pattern and combination. Use an FLMSYSLB or FLMCMPLB macro first, if you use them. Next, use FLMLANGL, the main macro, immediately followed by the FLMTRNSL macro. Use FLMALLOC multiple times with each FLMTRNSL macro. Then use FLMCPYLB one or more times after each FLMALLOC macro. You can repeat the FLMTRNSL macro, along with FLMALLOC and FLMCPYLB. Do not use FLMSYSLB, FLMCMPLB, and FLMLANGL again in the language definition.

The example on page 201 shows how to specify a language definition. Chapter 8, "SCLM Macros," gives details on the use of each language definition macro.

Modifying Language Definitions

You may have to modify your language definitions due to specific project needs. Limit these modifications to project-specific requirements. For each language, take the following actions as necessary:

- Specify data sets containing dependencies that are not to be tracked, such as system services (macro FLMSYSLB).
- Verify translator load module names and load data sets for accuracy (macro FLMTRNSL; keywords COMPILE and DSNAME).

- Verify existence and accuracy of copy libraries specified for the FLMCPYLB macro. Library names *must* match those in the FLMSYSLB macro for this language.
- 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).

  Note: If you change the language of a source member (for example PLIC to PLIO) using the SPROF macro, but do not perform a save on the member, the BUILD service recompiles the component automatically only if the new and old languages have unique version IDs. Project administrators should use a unique version ID on the FLMLANGL macro for each language definition in the project.
- 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.

The following pages provide an example of a language definition for the S/370 assembler H language.

```
*********************************************************************
* FLM@ASMH -- ASSEMBLER 'H'
*********************************************************************
*
ASMH     FLMSYSLB SCLM.RELEASE.MACROS
         FLMSYSLB SYS1.MACLIB
*
         FLMLANGL    LANG=ASMH
*
* PARSER TRANSLATOR
*
          FLMTRNSL   CALLNAM='ASM H PARSER',                          C
               FUNCTN=PARSE,                                          C
               COMPILE=FLMLSS,                                        C
               PORDER=1,                                              C
               OPTIONS=(PTABLEDD=,                                    C
               SOURCEDD=SOURCE,                                       C
               TBLNAME=FLMPASM,                                       C
               STATINFO=@@FLMSTP,                                     C
               LISTINFO=@@FLMLIS,                                     C
               LISTSIZE=@@FLMSIZ,                                     C
               CONTIN=72,                                             C
               EOLCOL=72)
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMPRJ.@@FLMGRP.@@FLMTYP(@@FLMMBR)
*
* BUILD TRANSLATOR(S)
*
*         --ASSEMBLER 'H' INTERFACE--
          FLMTRNSL   CALLNAM='ASSEMBLER H',                           C
               FUNCTN=BUILD,                                          C
               COMPILE=IEV90,                                         C
               DSNAME=SYS1.LINKLIB,                                   C
               VERSION=V05,                                           C
               GOODRC=0,                                              C
               OPTIONS=(XREF(SHORT),LINECOUNT(75),OBJECT,RENT)
*  1       --SYSLIN--
           FLMALLOC  IOTYPE=O,KEYREF=OBJ,RECFM=FB,LRECL=80,            C
              RECNUM=9000,DFLTTYP=OBJ
*  2       --N/A--
           FLMALLOC  IOTYPE=N
*  3       --N/A--
           FLMALLOC  IOTYPE=N
*  4       --SYSLIB--
           FLMALLOC  IOTYPE=I,KEYREF=SINC
            FLMCPYLB SCLM.RELEASE.MACROS
            FLMCPYLB SYS1.MACLIB
```

```
* 5       --SYSIN--
          FLMALLOC  IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,          C
              RECNUM=9000
* 6       --SYSPRINT--
          FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=FBA,LRECL=121,        C
              RECNUM=20000,PRINT=Y,DFLTTYP=LIST
* 7       --SYSPUNCH--
          FLMALLOC  IOTYPE=A
           FLMCPYLB NULLFILE
* 8       --SYSUT1--
          FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=15000
* 9       --SYSTERM--
          FLMALLOC  IOTYPE=A
           FLMCPYLB NULLFILE
```

In the previous example, FLMSYSLB and FLMCPYLB specify data sets that contain project and system macros or includes which are not tracked by SCLM. Note that the FLMSYSLB macro is always specified first. The name of the language, ASMH, is always specified next using the FLMLANGL macro.

Use keywords supported by the FLMTRNSL macro to specify the following: the translator to be invoked for a language, ASSEMBLER H; translator load module name, IEV90; version of the translator, 1.0; and translator options. An FLMALLOC macro addresses each ddname in the ddname substitution list for translator IEV90. Specify the kind of data set to be allocated, size, logical record length, and record format using keywords the FLMALLOC macro supports. An FMNCPYLM macro addresses each data set to be concatenated to a ddname (for example, SYS1.MACLIB).

Step 9: Modify Control Options

The SCLM project definition allows you to select project-specific controls that dictate SCLM processing. In particular, the following project controls are available:

- Accounting data set specification
- Secondary accounting data set specification
- Cross-reference data set specification
- Maximum lines per page
- Translator option override
- Maximum VIO limit
- Change code verification routine specification
- Build and promote user exit routine specification.

Specify control options using the FLMCNTRL macro. You can use only one reference to the FLMCNTRL macro for each project definition. "FLMCNTRL Macro" on page 218 shows the format of this macro.

In the example of a project definition on page 205, the accounting data set name is PROJ1.ACCOUNT.FILE, the maximum number of lines per page on listings is 75, and you can use overrides to the default translator options.

SCLM Internal Data Sets

The control options that follow allow you to designate specific SCLM internal data sets for the project.

Accounting Data Set Specification

The ACCT control option allows you to specify the VSAM accounting data set name. The data set you specify must be the name of the VSAM cluster you want. The default accounting cluster name is `project_id.ACCOUNT.FILE`, where `project_id` is the eight-character identifier for the project. The high-level qualifier must be a VSAM alias as described in "Step 5: Allocate Project Data Sets" on page 193.

Secondary Accounting Data Set Specification

The ACCT2 control option allows you to specify a backup VSAM accounting data set name for the project. Allocate this secondary VSAM data set following the same criteria as the primary one outlined in the previous section. You must choose a unique name for this data set and put it 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 could use this data set as a backup to restore the primary data set.

If you use this option, additional accounting updates that affect performance take place.

Cross-Reference Data Set Specification

The XREF control option allows you to specify the VSAM cross-reference data set, which is used to relate Ada compilation unit names with SCLM internal key information. The data set you specify must be the name of the VSAM cluster you want. The high-level qualifier must be a VSAM alias as described in "Step 5: Allocate Project Data Sets" on page 193. There is no default cross-reference cluster name.

Maximum Report Lines

Use MAXLINE to specify the maximum lines per page for all SCLM-generated reports. The default is 60.

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. See Chapter 2, "Architecture Definition," for specifying translator options in architecture members.

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. See Chapter 2, "Architecture Definition," for overriding translator defaults during an individual translation. The default for the OPTOVER control option is Y.

Maximum VIO Limit

The MAXVIO control option allows you to adjust the maximum VIO limit. Much of the processing in SCLM revolves around the use of temporary data sets. To increase performance, use VIO to allocate all temporary data sets. Due to memory limitations, however, SCLM establishes a VIO limit to restrict the amount of memory allocated. The maximum VIO limit forces SCLM to allocate the temporary data set on DASD if the function requests an amount greater than the limit. SCLM measures requests in the number of records. The default value is 5000. If SCLM functions fail for lack of memory (S80A ABEND), reduce this value.

Change Code Verification Routine Specification

The control option allows you to specify a change code verification routine to be used for the project. If you specify the routine, SCLM calls it to verify all change codes entered when a developer defines an update to a member to SCLM using the SCLM editor, migration utility, or services. If you specify a change code verification routine, developers can enter only members with valid change codes into the project database. There are performance implications associated with the specification of a change code verification routine. SCLM does not provide a default change code verification routine.

See "Change Code Verification Routines" on page 261 for details.

Build and Promote User Exit Routine Specification

The user exit options allow you to specify build and promote user exits to provide additional functions not supplied with SCLM. The user exits can perform logging functions, additional verification, or coordinated processing with non-SCLM tools. SCLM invokes the build user exit at the end of the build. SCLM invokes the promote verification user exit, the promote copy user exit, and the promote purge user exit routines at the end of promote verification, copy, and purge phases, respectively.

SCLM does not provide user exit routines. If you do not want any user exit routines for your project, you can skip the user exit specifications. See "Build and Promote User Exit Routines" on page 263 for more information.

The following page shows a sample project definition.

```
         TITLE '***PROJECT DEFINITION FOR PROJECT=PROJ1 ***'
PROJ1    FLMABEG
*
*        ************************************************************
*        *  DEFINE THE AUTHORIZATION CODES                          *
*        ************************************************************
*
GRP1     FLMAGRP AC"=(REL)
*
*        ************************************************************
*        *  DEFINE THE TYPES                                        *
*        ************************************************************
*
ARCHDEF  FLMTYPE
LIST     FLMTYPE
LMAP     FLMTYPE
LOAD     FLMTYPE
OBJ      FLMTYPE
SOURCE   FLMTYPE EXTEND=SOURCE2
SOURCE2  FLMTYPE
*
*        ****************************************************************
*        *  DEFINE THE GROUPS                                           *
*        ****************************************************************
*
USER1    FLMGROUP AC=(GRP1),PROMOTE=STAGE1,KEY=Y
USER2    FLMGROUP AC=(GRP1),PROMOTE=STAGE2,KEY=Y
USER3    FLMGROUP AC=(GRP1),PROMOTE=STAGE3,KEY=Y
STAGE1   FLMGROUP AC=(GRP1),PROMOTE=INT,KEY=N
STAGE2   FLMGROUP AC=(GRP1),PROMOTE=INT,KEY=N
STAGE3   FLMGROUP AC=(GRP1),PROMOTE=INT,KEY=N
INT      FLMGROUP AC=(GRP1),PROMOTE=TEST,KEY=Y
TEST     FLMGROUP AC=(GRP1),PROMOTE=RELEASE,KEY=Y
RELEASE  FLMGROUP AC=(GRP1),KEY=Y
*
*        ****************************************************************
*        *  PROJECT CONTROLS
*        ****************************************************************
*
         FLMCNTRL ACCT=PROJ1.ACCOUNT.FILE,                              C
              MAXLINE=75,                                               C
              OPTOVER=YES
*
```

```
*       ****************************************************************
*       *  LANGUAGE DEFINITION TABLES
*       ****************************************************************
*
COBOL   FLMSYSLB SYS1.EXAMPLE.MACROS
        COPY  FLM@ARCD          -- ARCHITECTURE DEF. LANGUAGE --
        COPY  FLM@TEXT          -- TEXT              LANGUAGE --
        COPY  FLM@SCRP          -- SCRIPT 3          LANGUAGE --
        COPY  FLM@ASM           -- 370 ASSEMBLER     LANGUAGE --
        COPY  FLM@COBL          -- COBOL             LANGUAGE --
        COPY  FLM@FORT          -- FORTRAN IV        LANGUAGE --
        COPY  FLM@PSCL          -- PASCAL            LANGUAGE --
        COPY  FLM@PLIC          -- PL/I CHECKOUT     LANGUAGE --
        COPY  FLM@PLIO          -- PL/I OPTIMIZER    LANGUAGE --
        COPY  FLM@L370          -- 370 LINKAGE EDITOR        --
*
*
        FLMAEND

            ( end of project definition example  )
```

Step 10: Assemble and Link Project Definition

Assemble all project definitions with the SCLM macro set using the standard IBM S/370 assembler. Once assembled, link the object using the standard IBM S/370 linkage editor, and store the project definition load module into the PROJDEFS data set. Make sure all project definition load modules are reentrant. Non-reentrant project definition load modules can cause error conditions. Specify the RENT option during link edit.

The load module name of the main (default) project definition for a project must match the project identifier on the FLMABEG macro. Alternate project definitions can have any load module name, but all alternate projects must have the same project identifier on the FLMABEG macro as the default project. All project definitions must reside in the PROJDEFS data set to allow invocation. Access the project definition when you invoke SCLM and use it for the duration of the SCLM invocation. An updated project definition does not affect an active SCLM invocation.

The SCLM macro set performs some verification of the project definition. When warning or error conditions are detected, the macros issue *MNOTES*, which are SCLM-specific diagnostic comments. 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. The return code from the assembler indicates the following:

0 The macros detected no errors.

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

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

Step 11: Build INFO Member

If you plan to use the Workstation Platform for Operating System/2 (OS/2), you must generate the members in the PROJDEFS.INFO data set. You create the PROJDEFS.INFO data set after your project definitions have been generated into your PROJDEFS data set.

The PROJDEFS.INFO data set is a partitioned data set with LRECL=40 and must be pre-allocated. It contains one member for each member in the PROJDEFS.LOAD data set. To create a member of the PROJDEFS.INFO data set, run the following JCL:

```
//INFO      EXEC PGM=ISRFLMGI,PARM=('project,projdef')
//STEPLIB   DD   DSN=<pdf_load_lib>,DISP=SHR
//SCLMLOAD  DD   DSN=<project>.PROJDEFS.LOAD,DISP=SHR
//SCLMINFO  DD   DSN=<project>.PROJDEFS.INFO,DISP=SHR
```

where:

project The project name to which this project definition belongs

projdef The project definition name for which the INFO member is being generated

pdf_load_lib The load library where the ISPF/PDF load modules reside if they are not in your normal system search sequence.

For example, if you wanted to generate the PROJDEFS.INFO member for a project definition called ALTVIEW under project PROJECT1, you would use the following JCL:

```
//INFO      EXEC PGM=ISRFLMGI,PARM=('PROJECT1,ALTVIEW')
//STEPLIB   DD   DSN=SYS1.PDF.LOAD,DISP=SHR
//SCLMLOAD  DD   DSN=PROJECT1.PROJDEFS.LOAD,DISP=SHR
//SCLMINFO  DD   DSN=PROJECT1.PROJDEFS.INFO,DISP=SHR
```

When ISRFLMGI completes, one of the following return codes is returned:

0 Member successfully created in <project>.PROJDEFS.INFO.

20 OPEN failed for ddname SCLMINFO.

21 OPEN failed for ddname SCLMLOAD.

22 Write to ddname SCLMINFO failed.

23 Invalid project name length.

24 No project specified.

25 Invalid project definition length.

26 No project definition specified.

27 Allocation of the Allocation Retrieval Area (ARL) failed.

28 OBTAIN failed.

29 The PROJDEFS.INFO data set is not partitioned with LRECL=40.

Chapter 8. SCLM Macros

General-Use Programming Interface

The SCLM macros are general-use programming interfaces, which you may use for programming purposes.

Introduction to SCLM Macro Instructions

SCLM supplies a set of macro instructions you can use to define project definitions. This chapter describes those macro instructions, explaining the format of each.

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.

This chapter uses the following notation conventions to describe the format of the SCLM macros:

Uppercase Uppercase commands or parameters must be spelled out as shown (in either uppercase or lowercase).

Lowercase Lowercase parameters are variables; substitute your own values.

Underscore Underscored parameters are the system default.

Brackets ([ ]) Parameters in brackets are optional.

Braces ({ }) Braces show two or more parameters from which you must select one.

OR (|) The OR (|) symbol shows two or more parameters from which you must select one.

Stacked Parameters
Stacked parameters show two or more parameters from which you can select. If you do not choose any, PDF uses the default parameter.

The example below shows the macro format for the SAMPLE macro.

```
SAMPLE PARM1=parm1 input
         ,PARM1A=XXX|YYY|ZZZ
        [,PARM2=parm2 input]
        [,PARM2A=Y|N]
```

Note that because these are S/370 assembler macros, all rules pertaining to macros apply. Assembler does not support blanks in macro parameters. 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. You can find more information on the use of macros in *OS Assembler H Language*, GC26-3771.

If an optional keyword is specified without a value, the default value is used; for example, PARM2A= causes PARM2A to default to Y.

© Copyright IBM Corp. 1989, 1990

FLMABEG Macro

Use this macro to define the project name of the project definition. It must appear before the other SCLM macros in the project definition.

Macro Format

```
name FLMABEG
```

Parameters

name
An eight-character project name. The project name is also the high-level qualifier for SCLM partitioned data sets.

Example

PROJ1 is the project name for the 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.

Macro Format

```
FLMAEND
```

Parameters

This macro has no parameters.

FLMAGRP Macro

Use this macro to define a group of authorization codes. You can then specify the group name in the AC field on the FLMGROUP macro to assign the group of authorization codes to that level.

Macro Format

```
name FLMAGRP AC=(code1,code2,...)
```

Parameters

name
An eight-character authorization group name containing no special characters or imbedded blanks.

AC = (*code1,code2,...*)
A list of authorization codes and authorization groups 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. See "Authorization Code Usage" on page 252 for more information.

Example

Authorization group GRP1 contains the authorization codes R3M0, R3M1, and R3M2. Authorization group GRP2 contains two authorization codes, R1M0 and R2M0, and one previously defined authorization group, GRP1, for a total of five authorization codes (R1M0, R2M0, R3M0, R3M1, and R3M2).

```
GRP1 FLMAGRP AC=(R3M0,R3M1,R3M2)
GRP2 FLMAGRP AC=(R1M0,R2M0,GRP1)
```

FLMALLOC Macro

Use this macro for each ddname in the ddname substitution list for a translator. Specify the kind of data set to be allocated, size, logical record length, and record format using this macro's keywords.

Macro Format

```
FLMALLOC IOTYPE={A|I|L|N|O|P|S|U|W}
   [,CATLG=N|Y]
   [,DDNAME=file_name]
   [,DIRBLKS=directory_blocks]
   [,DFLTTYP=default_type]
   [,KEYREF=keyword_reference]
   [,LRECL=record_length]
   [,NOSAVRC=no_save_rc]
   [,PRINT=N|Y|I]
   [,RECFM=record_format]
   [,RECNUM=number_of_records]
```

Parameters

IOTYPE = {A|I|L|N|O|P|S|U|W}
Specifies the type of files to be allocated and how these files will be used.

IOTYPE = A Allocate a data set or set of data sets to a specified ddname. You need the FLMCPYLB macro to identify the data sets. You can allocate a maximum of 16 data sets to the ddname.

IOTYPE = I Allocate libraries in the hierarchy for a particular type. The KEYREF parameter indicates the type. If the type has an extended type, the hierarchy for the extended type will be allocated after the first hierarchy.

IOTYPE = L Pass a member name in the ddname substitution list. See the PORDER parameter in "FLMTRNSL Macro" on page 228 for more information. The KEYREF parameter identifies the member name. This IOTYPE is commonly used to identify the load module name for S/370 linkage editor.

IOTYPE = N Skip over a field during ddname substitution. SCLM passes eight hexadecimal 00s to the translator.

IOTYPE = O Allocate a sequential data set for a translator output that is to be saved in the database. The KEYREF parameter identifies the output module name and type. Valid KEYREF values are OBJ, COMP, LIST, LOAD, LMAP, and OUTx. You need the RECFM, LRECL, and RECNUM parameters for allocation of the data set.

IOTYPE=P Allocate a partitioned data set into which the translator writes a member to be saved in the database. Use the KEYREF parameter to identify the target member for copying of the data set. Specify either KEYREF=LOAD or OUTx. You need the RECFM, LRECL, RECNUM, and DIRBLKS parameters for allocation of this data set.

IOTYPE=S Allocate a temporary data set and create the input stream for the translator. Use the KEYREF parameter to identify the members used to create the input stream.

IOTYPE=U Use the preallocated ddname specified on the DDNAME parameter.

IOTYPE=W Allocate a temporary work data set for translator use. You need the RECFM, LRECL, and RECNUM parameters for allocation of this data set.

SCLM generates ddnames for all IOTYPES during build processing. In all cases (except IOTYPE=U), the DDNAME parameter is optional, and you should use it only if the translator requires a specific ddname which cannot be substituted. The position of the FLMALLOC macros is very important because SCLM may pass ddnames directly to the translator (see PORDER field). SCLM passes ddnames to the translator in the order of the FLMALLOC macros.

SCLM does not deallocate temporary data sets that were allocated with IOTYPE=O, P, S, and W, and temporary hierarchies allocated with IOTYPE=A and I until all translators in the language definition complete processing. Thus, a translator output data set can become an input data set for the next translator step.

,CATLG=N|Y

Indicates whether a data set is to be cataloged. Valid for IOTYPE=W, O, P and S, SCLM allocates cataloged data sets with a predefined high-level qualifier, usually the user ID. This is necessary for translators that require all data sets to have RACF. The default is N, but, generally, you should not use it.

,DDNAME=*file_name*

The ddname to be used for this allocation. If you do not specify a ddname for the allocation, SCLM generates one for you.

,DIRBLKS=*directory_blocks*

The numeric directory block size of the data set. The default is zero. It is valid for IOTYPE=P.

,DFLTTYP=*default_type*

Indicates the name of the target area for translator outputs. Allocate translator outputs with IOTYPE=O or P. The target 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 target outputs with an output keyword such as OBJ, OUTx, or LOAD.

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 will be replaced by the name of the source member. If the substitution of the source type would result in a name longer than eight characters, the source type is truncated to produce an eight 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.

,KEYREF = *keyword reference*
Refers to a keyword (in the build map) or a statement (in the architecture definition). The member name and type 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.
- If IOTYPE = I, *keyword_reference* determines the type name of the hierarchy to allocate.
- If IOTYPE = O or P, *keyword_reference* identifies the member written to by the translator.

,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.

,NOSAVRC = *no_save_rc*
A return code value indicating whether or not SCLM stores translator output in this data set (valid for IOTYPE = O and P). Use this field when a translator produces an optional output, which can be determined by the return code from the translator. The build processor determines that the translator did not store output to the data set if *no_save_rc* is equal to a nonzero translator return code. The default is 0.

This feature can be used in conjunction with the field DEPPRCS on the FLMLANGL macro to allow or disallow dependency processing to continue. See "Processing Conditionally Saved Components" on page 251 for more information.

,PRINT = **N**|Y|I
Indicates whether SCLM sends the contents of a sequential data set to the listings data set. This parameter is valid only for data sets allocated with IOTYPE = W, S, or O. The valid values are:

- Print = N indicates the data set is not to be printed.
- Print = Y indicates the data set is to be printed.
- Print = I indicates the data set is to be initialized when allocated by SCLM and is to be printed.

Data sets that you specify for print but do not open with the invoked translator can result in an ABEND during printing. In such cases, specify `PRINT=I`. If you specify `PRINT=I`, build performance is slightly degraded. The default is N.

,RECFM = *record_format*
Record format of the data set. It is valid for IOTYPE = W, O, P, and S. The default is FB (fixed blocks).

,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.

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 the following FLMALLOC macros:

```
FLMALLOC IOTYPE=S
FLMALLOC IOTYPE=O,KEYREF=LIST,DFLTTYP=LISTING
FLMALLOC IOTYPE=O,KEYREF=OBJ,DFLTTYP=OBJECT
```

Building the member is the same as building the following architecture definition:

```
SINC xxxxxxxx SOURCE
LIST xxxxxxxx LISTING
OBJ  xxxxxxxx OBJECT
```

Always use the SINC keyword to identify the input member. The FLMALLOC macros you use can only specify the output targets (IOTYPE = I and P). You can also use the fields DFLTCRF and DFLTSRF on the FLMLANGL macro to define compool and source dependency types, respectively. If you need multiple SINC keywords, then you must use an architecture member to specify the software component. Options to override the translator options (using the PARMx keywords) also require that you use an architecture member.

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
```

FLMCMPLB Macro

Use this macro to define a set of data sets for a language containing project compool dependencies. SCLM does not track compool dependencies that exist in the hierarchy and that are found in these partitioned data sets. The total number of data sets may not exceed 16 for any language.

Macro Format

```
[language] FLMCMPLB dataset_name
```

Parameters

language
An eight-character language name. You must specify the language with the same name as the language you specify in the LANG parameter on the FLMLANGL macro. In order to specify multiple data sets for a language, omit the language on all but the first data set.

dataset_name
The partitioned data set containing members that SCLM does not track.

Example

When a JOVC source member is parsed, SCLM first checks the project hierarchy for each compool dependency it finds. If it finds the compool member in the hierarchy, SCLM tracks the compool member. If it does not find it, SCLM searches the two FLMCMPLB libraries specified for the language. If it finds the compool member in the concatenation of these data sets, the member is removed from the list of compool dependencies that SCLM tracks. However, if it does not find the compool member in the FLMCMPLB data sets, SCLM still tracks the nonexistent compool member.

```
JOVC     FLMCMPLB SYS1.SYSTEM.COMPLIB
         FLMCMPLB SYS1.EXCLUDE.COMPLIB
JOV      FLMCMPLB SYS1.SYSTEM.COMPLIB
```

FLMCNTRL Macro

Use this macro to specify project-specific control options. You can use this macro only one time.

Macro Format

```
FLMCNTRL [ACCT=primary_file]
  [,ACCT2=secondary_file]
  [,BLDEXT1=build_exit_routine]
  [,BEXT1DS=build_exit_dataset]
  [,BEXT1OP=build_exit_options]
  [,MAXLINE=max_line_count|60]
  [,MAXVIO=max_vio_count|5000]
  [,OPTOVER=N|Y]
  [,PEXT1DS=promote_exit1_dataset]
  [,PEXT2DS=promote_exit2_dataset]
  [,PEXT3DS=promote_exit3_dataset]
  [,PEXT1OP=promote_exit1_options]
  [,PEXT2OP=promote_exit2_options]
  [,PEXT3OP=promote_exit3_options]
  [,PRMEXT1=promote_exit1_routine]
  [,PRMEXT2=promote_exit2_routine]
  [,PRMEXT3=promote_exit3_routine]
  [,VERCC=change_code_routine]
  [,VERCCDS=change_code_dataset]
  [,VERCCOP=change_code_options]
  [,XREF=cross_reference_file]
```

Parameters

ACCT = *primary_file*

The name of the VSAM accounting data set for the project. The data set you specify must be the name of the VSAM cluster you want. The default accounting cluster name is `project.ACCOUNT.FILE`, where *project* is the project name specified on the FLMABEG macro. The high-level qualifier must be a VSAM alias as described in "Step 5: Allocate Project Data Sets" on page 193.

,ACCT2 = *secondary_file*
The name of a backup VSAM 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.

Because additional accounting updates take place if you use this option, the updates may degrade performance.

,BLDEXT1 = *build_exit_routine*
The member name of the build user exit routine. SCLM invokes the routine at the completion of the build process. It does not invoke the routine if the build mode is REPORT. Specify the load data set containing the routine in the BEXT1DS parameter. If you do not specify the BLDEXT1 parameter, then SCLM does not invoke the exit routine. See "Build and Promote User Exit Routines" on page 263 for more information.

,BEXT1DS = *build_exit_dataset*
The load data set name containing the member name specified in the BLDEXT1 parameter. Specify this parameter if you specify BLDEXT1. See "Build and Promote User Exit Routines" on page 263 for more information.

,BEXT1OP = *build_exit_options*
Option list to be passed to the build user exit routine. You can specify a maximum of 256 characters for the options. You can delimit the options with single quotes. The option list precedes the options passed by the build processor's parameters, thus allowing developers to specify run-time options with this parameter. See "Build and Promote User Exit Routines" on page 263 for more information.

,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 20, 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.

,OPTOVER = **N**|**Y**
Indicates whether translator option overrides are allowed or disallowed. If OPTOVER = Y, developers can add or override the translator options by specifying the keyword PARMx in the architecture member followed by the user options. The default is Y.

,PEXT1DS = *promote_exit1_dataset*
The load data set name containing the member name specified in the PRMEXT1 parameter. Specify this parameter if you specify PRMEXT1. See "Build and Promote User Exit Routines" on page 263 for more information.

,PEXT2DS = *promote_exit2_dataset*
The load data set name containing the member name specified in the PRMEXT2 parameter. Specify this parameter if you specify PRMEXT2. See "Build and Promote User Exit Routines" on page 263 for more information.

,PEXT3DS = *promote_exit3_dataset*
The load data set name containing the member name specified in the PRMEXT3 parameter. Specify this parameter if you specify PRMEXT3. See "Build and Promote User Exit Routines" on page 263 for more information.

,PEXT1OP = *promote_exit1_options*
The option list to be passed to the first promote user exit routine. You can specify a maximum of 256 characters for the options. Delimit the options with single quotes. The option list precedes the options passed by the promote process's parameters, thus allowing developers to specify run-time options in this parameter. See "Build and Promote User Exit Routines" on page 263 for more information.

,PEXT2OP = *promote_exit2_options*
The option list to be passed to the second promote user exit routine. You can specify a maximum of 256 characters for the options and delimit them with single quotes. The option list precedes the options passed by the promote processor's parameters, thus allowing developers to specify run-time options in this parameter. See "Build and Promote User Exit Routines" on page 263 for more information.

,PEXT3OP = *promote_exit3_options*
The option list to be passed to the third promote user exit routine. You can specify a maximum of 256 characters for the options and delimit them with single quotes. The option list precedes the options passed by the promote processor's parameters, thus allowing developers to specify run-time options in this parameter. See "Build and Promote User Exit Routines" on page 263 for more information.

,PRMEXT1 = *promote_exit1_routine*
The member name of the first user exit routine for the promote process. SCLM invokes this exit after the verification phase of the promote process. This member exists in the load data set specified by the PEXT1DS parameter. If you do not specify PRMEXT1, SCLM does not invoke the exit routine. See "Build and Promote User Exit Routines" on page 263 for more information.

,PRMEXT2 = *promote_exit2_routine*
The member name of the second user exit routine for the promote process. SCLM invokes this exit after the copy phase of the promote process. Ensure that this member exists in the load data set specified by the PEXT2DS parameter. If you do not specify PRMEXT2, SCLM does not invoke the exit routine. See "Build and Promote User Exit Routines" on page 263 for more information.

,PRMEXT3 = *promote_exit3_routine*
The member name of the third user exit routine for the promote process. SCLM invokes this exit after the purge phase of the promote process. This member must exist in the load data set specified by the PEXT3DS parameter. If you do not specify PRMEXT3, SCLM does not invoke the exit routine. See "Build and Promote User Exit Routines" on page 263 for more information.

,VERCC = *change_code_routine*
The name of the change code verification routine. If you do not specify the VERCC parameter, SCLM does not invoke the verification routine. See "Change Code Verification Routines" on page 261 for more information.

,VERCCDS = *change_code_dataset*
The load data set name containing the member name specified in the VERCC parameter. If you specify this parameter, the verification routine named by VERCC must exist in this data set. See "Change Code Verification Routines" on page 261 for more information.

,VERCCOP = *change_code_options*
The option list to be passed to the change code verification routine. You can specify a maximum of 256 characters for the options and delimit them with single quotes. The option list precedes the options passed by the routines invoking the verification routine, thus allowing developers to specify run-time options in this parameter. See "Change Code Verification Routines" on page 261 for more information.

,XREF = *cross_reference_file*
The name of a VSAM cross-reference data set for the project. Use the cross-reference data set to store compilation unit dependency information. The data set you specify must be the name of the VSAM cluster you want. The high-level qualifier must be a VSAM alias as described in "Step 5: Allocate Project Data Sets" on page 193. This data set is not necessary unless you use a language with cross-reference dependencies, for example, Ada. There is no default cross-reference cluster name.

Example

The accounting data set is specified. Allocate data sets using the FLMALLOC macro with VIO if the RECNUM parameter is not greater than 10000. The macro calls a change code verification routine when a member is parsed.

```
FLMCNTRL ACCT=PROJ1.ACCOUNT.FILE,                              X
  MAXVIO=10000,                                                X
  VERCC=CCCHECK,                                               X
  VERCCDS=PROJ1.VERIFY.LOAD
```

FLMCPYLB Macro

Use this macro to identify additional data sets to be concatenated to a ddname. It is preceded by an FLMALLOC macro used to specify the ddname.

Macro Format

```
FLMCPYLB dataset_name|NULLFILE
```

Parameters

dataset_name|**NULLFILE**

Use the FLMCPYLB macro to allocate a data set to a ddname. Place the FLMCPYLB after an FLMALLOC macro with IOTYPE=I or A. See the IOTYPE parameter on the FLMALLOC macro for more information. In all other cases, SCLM ignores the data sets. If you specify more than one FLMCPYLB, SCLM concatenates the data sets in the order they are specified. When you use them with the IOTYPE=I, SCLM allocates the data sets behind the type hierarchy libraries. SCLM can concatenate up to 16 data sets. Thus, when you use IOTYPE=I, ensure that the number of levels in the hierarchy (primary levels), plus the number of FLMCPYLB macros you specify, do not exceed 16. If you concatenate more than 16 data sets, the project definition assembles without errors, but using it produces unpredictable results.

Specify `NULLFILE` for the data set name for allocation of a dummy data set.

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 when the translator is invoked.

FLMGROUP Macro

Use this macro to define one group in the project definition.

Macro Format

```
name FLMGROUP

     [AC=(code1,code2,...)]

     [,KEY=N|Y]

     [,PROMOTE=next_group]
```

Parameters

name
An eight-character group name.

AC = (*code1,code2,...*)
A list of authorization codes and authorization groups which defines the authorization codes that are valid for the given group. If *code#* is an authorization group, then you must have previously defined it with the FLMAGRP macro. See "Authorization Code Usage" on page 252 for more information.

The first authorization code you specify is the default authorization code used when a member is introduced to SCLM in this group. The maximum number of characters allowed within the parentheses is 256.

If you omit this parameter, you cannot edit any members in this group. In addition, no editable members may be promoted into or out of this group.

,KEY = **N|Y**
Defines whether the group is a key group or a non-key group. The default is Y. See "Key/Non-Key Groups" on page 8 for more information.

,PROMOTE = *next group*
Defines the next group within the hierarchy for this group. If you do not specify it, SCLM does not allow any promotions out of this group.

Example

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 can be promoted into 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
```

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 (using the FLMTRNSL macro) after FLMLANGL.

Macro Format

```
FLMLANGL LANG=language

   [,ADABIND=N|Y]

   [,ARCH=N|Y]

   [,BUFSIZE=buffer_size|100]

   [,CANEDIT=Y|N]

   [,COMPOOL=N|Y]

   [,CUQUAL=compilation_unit_qualifier]

   [,DEPPRCS=Y|N]

   [,DFLTCRF=default_compool_reference]

   [,DFLTSRF=default_source_reference]

   [,IMPSPEC=Y|N]

   [,SCOPE=LIMITED|NORMAL|SUBUNIT|EXTENDED]

   [,VERSION=language_version]
```

Parameters

LANG = *language*
An eight-character language name. It is stored with the accounting information of editable members. The developer specifies this name when you first define a member to SCLM.

,ADABIND = **N|Y**
Indicates whether this language controls the bind operation for compilation units. Use only for Ada languages. The default is N.

,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 parser translator. The parser translator returns dependency information in the allocated memory. The default size is 100. SCLM requires one record for each include, compool, change code, user data record, or compilation unit the parser 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.

,COMPOOL = N|Y
Indicates whether this is a compool language. This keyword controls whether or not build processes the compool dependencies for source members parsed under this language. The default is N.

,CUQUAL = *compilation_unit_qualifier*
The compilation unit qualifier associated with cross-reference data sets for this language. Use this parameter to distinguish between different Ada languages, for example, IBM Ada with MVS targets as opposed to VM targets. SCLM uses the parameter as the lowest level qualifier of a sublibrary name when you invoke the compiler. This parameter is required for Ada languages.

,DEPPRCS = Y|N
Indicates whether other components depending on this component will be rebuilt if some outputs from the translator were not saved. See "Processing Conditionally Saved Components" on page 251 for more information.

,DFLTCRF = *default_compool_reference*
The type name containing the compool dependencies. SCLM ignores this parameter during a build if it uses an architecture member to build the source member. For architecture members, you define the type name of compool dependencies with a CREF keyword. If you do not specify it, SCLM sets this parameter to blank.

,DFLTSRF = *default_source_reference*
The type name containing source dependencies. The build process allocates the hierarchy of this type. SCLM ignores this parameter during a build if it uses an architecture member to build the source member. For architecture members, you define the type name of source references with an SREF keyword. Allocate the hierarchy for the type by specifying an FLMALLOC macro with IOTYPE = I and KEYREF = SREF to identify the type. If you do not specify this parameter, SCLM sets it to blank.

,IMPSPEC = Y|N
Indicates whether implicit specifications are allowed for builds of Ada languages with compilation unit dependencies. For example, if a procedure BODY is present but the SPEC is not and IMPSPEC = N, the build function issues an error message. The default is Y.

,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. Scopes are only used to control the processing scope of compilation unit dependencies during builds and promotes. See Chapter 9, "Advanced Topics," for more information. The default is NORMAL.

,VERSION = *language_version*
The eight-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. See "Modifying Language Definitions" on page 199 for additional information.

Example

The language definition for PASCAL is defined.

```
FLMLANGL LANG=PASCAL,VERSION=1.0
```

FLMSYSLB Macro

Use this macro to define a set of data sets for a language that contains project system macros or includes. SCLM does not track included members that do not exist in the hierarchy but are found in these partitioned data sets. Do not define more than 16 data sets for any language.

Macro Format

```
[language] FLMSYSLB dataset_name
```

Parameters

language
An eight-character language name. The language must be the same name as the language specified in the LANG field on the FLMLANGL macro. In order to specify multiple data sets for a language, omit the language on all but the first data set.

dataset_name
The partitioned data set containing members that are not to be tracked by SCLM.

Example

When an ASMH source member is parsed, the project hierarchy is first checked for each include dependency found. If the included member is found in the hierarchy, SCLM tracks it. If it is not found, SCLM searches the three FLMSYSLB libraries specified for the language for the include member. If SCLM finds the include member in the concatenation of these data sets, it removes the member from the list of included dependencies that it tracks. However, if it does not find the member in the FLMSYSLB data sets, SCLM still tracks the nonexistent include member.

```
ASMH     FLMSYSLB  SYS1.MACHLAL
         FLMSYSLB  SYS1.MACLIB
         FLMSYSLB  PROJECT.SPECIAL.INCLUDES
ASM      FLMSYSLB  SYS1.MACHLAL
         FLMSYSLB  SYS1.MACLIB
SIMPLE   FLMSYSLB  SYS1.MACHLAL
         FLMSYSLB  SYS1.MACLIB
```

FLMTRNSL Macro

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 keywords.

Macro Format

```
FLMTRNSL CALLNAM='call_name'

   ,COMPILE=compiler

   [,DSNAME=dataset_name]

   [,FUNCTN=PARSE|BUILD|COPY|PURGE]

   [,GOODRC=good_return_code|0]

   [,NOSVEXT=no_save_external_rc|0]

   [,OPTFLAG=N|Y]

   [,OPTIONS=option_list]

   [,PARMKWD=parameter_keyword]

   [,PORDER=0|1|2|3]

   [,VERSION=translator_version]
```

Parameters

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 imbedded blanks in the call name, surround the string with single quotes.

,COMPILE = *compiler*
The entry point name on the translator load module.

,DSNAME = *dataset_name*
The name of the data set containing the translator load module (COMPILER parameter). If the translator load module resides in JOBLIB, STEPLIB, TASKLIB, or LPA, do not specify data set name.

,FUNCTN = **PARSE|BUILD|COPY|PURGE**
Identifies the function performed by the translator. The parse function invokes the parse translators. The build function invokes translators. The promote function invokes copy and purge translators. The default is PARSE.

,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 message greater than *good_return_code* from a translator, the process has failed. No outputs are saved in the hierarchy. The default is 0.

,NOSVEXT = *no_save_external_rc*|**0**
A return code value indicating whether any translator outputs targeted to an external data set, for example, Ada sublibraries, were saved (valid for FUNCTN=BUILD). Use this parameter in conjunction 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. See "Processing Conditionally Saved Components" on page 251 for more information. The default is 0. If you have DEPPRCS=Y, this parameter has no effect.

,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*
The default translator options (maximum 256 characters). Delimit the options with single quotes or parentheses. They can also contain variables to provide dynamic information to a translator. See Chapter 3, "SCLM Variables."

,PARMKWD = *parameter_keyword*
The keyword (PARM0..PARM9) used in architecture members to specify additional options for this translator. SCLM takes the options specified for this keyword and concatenates them after the translator options (OPTIONS parameter) to form the final option list for the translator.

,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 3. 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 213. The following list defines the valid values for the translator parameter order:

0 No parameters passed

1 Pass option list

2 Pass ddname subsitution list

3 Pass option list followed by ddname substitution list.

,VERSION = *translator_version*
An eight-character representation of the translator version. This parameter is for informational purposes only. SCLM stores it in its internal data for each output member saved from the translators. If you do not specify this parameter, SCLM sets it to blank.

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 4 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 one FLMTYPE in the project definition.

Macro Format

```
name FLMTYPE [EXTEND=extended_type]
```

Parameters

name
An eight-character type name.

EXTEND = *extended_type*
An eight-character name that defines an alternate type to use when resolving include dependencies. This parameter allows the include dependencies to exist in a type different from the one containing the including member. If you do not specify it, you must ensure that all include dependencies being tracked reside in the same type containing the including member. Define the name to SCLM as another type.

EXTEND creates a single-level deep search sequence. The extended type cannot be further extended. For example, if the SOURCE was extended to SOURCE2, and SOURCE2 was extended to SOURCE3, SCLM resolves only SOURCE and SOURCE2.

Example

Six types are defined. Type SOURCE2 is an extension of type SOURCE. In SCLM, if a member exists in type SOURCE, its include dependencies can exist in either SOURCE or SOURCE2.

```
OBJ      FLMTYPE
LIST     FLMTYPE
LMAP     FLMTYPE
LOAD     FLMTYPE
SOURCE   FLMTYPE EXTEND=SOURCE2
SOURCE2  FLMTYPE
```

Chapter 9. Advanced Topics

This chapter describes advanced topics that aid you in managing complex configurations. Topics discussed in this chapter include:

- Impact assessment techniques
- New languages definitions
- Authorization code usage
- Dynamic include tracking
- Alternate project definitions
- Primary non-key group testing techniques
- Change code verification routines
- Build and promote user exit routines
- Project conversion to SCLM
- Security
- Backup and recovery of the project database
- Dependency processing implementation
- Development and performance
- Workstation platform for OS/2
- The SSI field in load module directories.

Impact Assessment Techniques

Making updates to an application without full knowledge of their effect on the application can create costly surprises. Impact assessment is a technique you can use to assess the impacts of updates to an application *before* they occur. In effect, it allows developers to determine what effect changing a given component of the application will have on the rest of the application or a given subapplication. Impact assessment enables you to avoid costly and unnecessary integration work.

Follow the procedure below to use the build processor to create an impact assessment:

1. Use the SCLM editor to update the members you want to change.
2. Invoke the build function in the report mode on the top architecture definition for the application affected.
3. Examine the resulting build report. This report reflects all translator invocations that would have occurred as a result of the updates.
4. If the results are acceptable, you can build the proper architecture definition and promote it.
5. If the results are too costly, you can do the updates at another time. To avoid accidental builds, use the SCLM library utility to delete the members that were modified in Step 1.

© Copyright IBM Corp. 1989, 1990

You can construct 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 technique outlined above will help prevent costly recompilation impacts.

New Language Definitions

An important feature of SCLM is that you can tailor it to work with almost any language. You define the languages supported for a particular SCLM project in the project definition. The definition of a language to be supported by SCLM is called a *language definition*.

Define a language definition using the FLMLANGL macro and translator definitions. Use translator definitions to specify what translators are to be called by SCLM to support a language, and when and how these translators are to be called. The parameters of the FLMLANGL macro define some global information about a language, such as its name.

Define a translator definition, in turn, using the FLMTRNSL macro and allocation definitions. Use allocation definitions to specify what ddnames are to be allocated to support a translator and how these ddnames are to be allocated and used. The parameters of the FLMTRNSL macro define all the attributes needed to call a given translator. Of particular interest is the FLMTRNSL FUNCTN parameter, which defines the function or purpose for which a translator is to be called. SCLM uses translators for the following functions:

- Parsing source code to determine statistics and dependency information. SCLM calls these translators during the save process of editing.
- Translating one form of code into another. Some sample code translations are:
  - COBOL code to object code and listings
  - Script input to a formatted document
  - Object modules to load modules.

  SCLM calls these translators during the build process.
- Copying intermediate code for compilation units. SCLM calls these translators during the promote process.
- Purging intermediate code for compilation units. SCLM calls these translators primarily during the promote process.

Because most languages do not have compilation units, you only need translators for the functions of translating code and parsing.

In turn, you define an allocation definition using the FLMALLOC macro and, in some cases, a series of copy library definitions. Copy library definitions simply consist of a FLMCPYLB macro specification.

In summary, a language definition consists of a hierarchy of the following subdefinitions:

- System library definition
- Language identifier definition
- Translator definition
- Allocation definition
- Copy library definition.

Because a macro exists for each of these definitions and because each macro accepts a number of different parameters, you can specify an unlimited variety of language definitions.

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 8, "SCLM Macros." Typically, if you want to write a new language definition, you should copy an old language definition, and then modify it to meet your specific needs. The best way to learn how to modify language definitions is to study language definitions supplied with SCLM and observe what makes them work.

In the remainder of this section, several translator definitions are examined more closely in order to describe some of the more complicated things that can be done with language definitions.

Using Multiple Translators in a Language Definition

Most SCLM-supplied language definitions have two translator definitions. The first translator definition defines the parser to be used, and the second translator definition defines the translator to be used during a build. SCLM provides you with the ability to create language definitions that have more than one translator definition for a function. SCLM invokes these build processes in the defined order and passes data forward. This capability allows you to customize the SCLM product for unique build processing requirements in your project.

The following example shows a language definition that uses multiple translators. Notice that the ddname SYSLIN is used for both translators. For the SYSLIN ddname in the link editor translator, however, `IOTYPE=U` is used because the translator is using a preallocated ddname.

```
*********************************************************************
* COBLOAD   COBOL II COMPILER AND LINKEDIT
*********************************************************************
COBLOAD  FLMSYSLB    PROJ1.COBOL.COPYLIB
         FLMLANGL    LANG=COBLOAD
*
* PARSER TRANSLATOR
*
         FLMTRNSL   CALLNAM='COBOL PARSER',                          C
               FUNCTN=PARSE,                                         C
               COMPILE=FLMLSS,                                       C
               PORDER=1,                                             C
               OPTIONS=(PTABLEDD=,                                   C
               SOURCEDD=SOURCE,                                      C
               TBLNAME=FLMPCOB,                                      C
               STATINFO=@@FLMSTP,                                    C
               LISTINFO=@@FLMLIS,                                    C
               LISTSIZE=@@FLMSIZ,                                    C
               CONTIN=0,                                             C
               EOLCOL=72)
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMPRJ.@@FLMGRP.@@FLMTYP(@@FLMMBR)
*
* BUILD TRANSLATOR(S)
*
*         --COBOL INTERFACE--
          FLMTRNSL   CALLNAM='COBLOAD COBOL II COMPILER',            C
               FUNCTN=BUILD,                                         C
               COMPILE=IGYCRCTL,                                     C
               DSNAME=IGZ.V1R2M0.COB2COMP,                           C
               VERSION=2.0,                                          C
               GOODRC=0,                                             C
               OPTIONS=(XREF,LIB,APOST)
*  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
            FLMCPYLB PROJ1.COBOL.COPYLIB
*  5       (* SYSIN *)
           FLMALLOC  IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,         C
               RECNUM=2000
*  6       (* SYSPRINT *)
           FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=FBA,LRECL=121,       C
               RECNUM=5000,PRINT=N,DFLTTYP=LIST
*  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
            FLMCPYLB NULLFILE
* 13       (* SYSUT5 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
*
*
* 370/LINKAGE EDITOR
*
          FLMTRNSL   CALLNAM='COBLOAD 370 LINK EDITOR',                C
               FUNCTN=BUILD,                                           C
               COMPILE=IEWL,                                           C
               VERSION=2.0,                                            C
               GOODRC=0,                                               C
               OPTIONS=(DCBS,MAP)
*
*  1       (* SYSLIN *)
           FLMALLOC  IOTYPE=U,DDNAME=SYSLIN
*
*  2       (* LOAD MODULE NAME *)
           FLMALLOC  IOTYPE=L,KEYREF=OUT1
*
*  3       (* SYSLMOD *)
           FLMALLOC  IOTYPE=P,KEYREF=OUT1,RECFM=U,LRECL=6144,           C
               RECNUM=500,DIRBLKS=20,DDNAME=SYSLMOD,DFLTTYP=LOAD
*
*  4       (* SYSLIB  *)
           FLMALLOC  IOTYPE=A,DDNAME=SYSLIB
            FLMCPYLB IGZ.V1R2M0.COB2LIB
            FLMCPYLB SYS1.LINKLIB
*
*  5       (* N/A *)
           FLMALLOC  IOTYPE=N
*
*  6       (* SYSPRINT *)
           FLMALLOC  IOTYPE=O,KEYREF=OUT2,RECFM=FBA,LRECL=121,          C
               RECNUM=2500,PRINT=N,DFLTTYP=LMAP
*
*  7       (* N/A *)
           FLMALLOC  IOTYPE=N
*
*  8       (* SYSUT1 *)
           FLMALLOC  IOTYPE=W,RECFM=U,LRECL=6144,RECNUM=200,            C
               DDNAME=SYSUT1
*
*  9       (* N/A *)
           FLMALLOC  IOTYPE=N
*
```

```
* 10       (* N/A *)
           FLMALLOC  IOTYPE=N
*
* 11       (* N/A *)
           FLMALLOC  IOTYPE=N
*
* 12       (* SYSTERM *)
           FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
            FLMCPYLB NULLFILE
*
```

Invoking User-Defined Parsers

SCLM allows you to replace an SCLM-supplied source parser with a user-defined parser. This option is particularly 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 which the SCLM-supplied parsers can recognize.

When you write a new parser for a language, you must do the following:

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 what you determine in step (1), that passes to SCLM the statistical and dependency information for a module written in this new language. This program is called a parser.
3. Tell SCLM how to invoke your parser.

At the end of this section is a parser, written in PL/I and Assembler, for the ISPF skeleton language. We'll take you through the three steps above and use the SKELS parser as an example.

Defining Information Tracked by SCLM

SCLM tracks two kinds of information for each module: statistical information and dependency information. Statistical information includes such data as the total lines and the number of comments in the module. See "Statistics" on page 57 for a description of the 10 statistics kept by SCLM. Dependencies tracked by SCLM are generally the names of other modules that are used when this module is built. When you write a new parser, you have to define exactly how the parser derives this information from a module.

For the SKELS (ISPF skeleton) language, we define the 10 SCLM statistics as follows:

Total lines	The number of lines in the skeleton
Comment lines	The number of lines that start with)CM
Noncomment lines	The number of lines that do not start with)CM
Blank lines	The number of lines that start with)BLANK
Prolog lines	The number of comment lines before the first noncomment line or the number of comment lines if there are no noncomment lines
Total statements	Same as Total Lines
Comment statements	Same as Comment Lines
Control statements	The number of lines that start with ')'
Assignment statements	Always zero (0) for skeletons
Noncomment statements	Same as Noncomment Lines

Note that these definitions may not exactly agree with the definitions in "Statistics" on page 57. It does not matter. It is your language and these statistics are kept for your benefit, not for SCLM.

For this example, suppose that if a line starts with)IM, SCLM is to track the named skeleton as a dependency. You also want to use the user fields in the account records to store the name of each ISPF table that appears on a)DOT control statement.

Writing the Parser

There are several things to consider 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'
  ```

- There is a set of SCLM variables (see Chapter 3, "SCLM Variables") that can be used 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 structure and array as it parses the module. SCLM can pass the address of the structure and of 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 module's account record.

In the SKELS parser example, the routine consists of five 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 Readies the parse structure and the parse array (LIST_INFO) to be passed back to SCLM.

DATRC Is an assembler routine that returns a fullword integer return code to SCLM. PL/I routines cannot return fullword integer return codes.

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 before the source listing. For your parser, you need the following:

- 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 the example, there are several keywords on the macros that bear close scrutiny.

On the FLMLANGL macro, the LANG keyword indicates the string (in this case it is "SKEL") 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 parameter, the COMPILE and DSNAME keywords tell SCLM that the parser can be found in 'GERKEN.PROJECT.LOAD(FLM@SKLS)'. The OPTIONS keyword 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 will be 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 out a message explaining the situation and what needs to be done to correct it. If the SKELS parser passes back a return code greater than that specified on the GOODRC keyword of the FLMTRNSL macro, the contents of this listings file is 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=GERKEN.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 @@FLMPRJ.@@FLMGRP.@@FLMTYP(@@FLMMBR)
           (* LISTING     *)
           FLMALLOC IOTYPE=W,RECFM=VBA,LRECL=133,                    C
               RECNUM=6000,DDNAME=ERROR,PRINT=Y
```

Figure 51. 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:              ***/
/***                                                               ***/
/***                   1) Increase the value of BUFSIZE in the     ***/
/***                      FLMLANGL macro for this parser, or       ***/
/***                   2) 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 52 (Part 1 of 11). Parser for ISPF Skeletons

```
PSKELS: PROC(PARMLIST) OPTIONS(MAIN);
   DCL PARMLIST     CHAR(256) VAR; /* Parameter list                */
   DCL PARMLISTx    CHAR(256) 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;
   DCL DATRC        EXTERNAL ENTRY OPTIONS(ASM INTER);
   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 DATRC(RETCODE);
```

Figure 52 (Part 2 of 11). Parser for ISPF Skeletons

```
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.                        ***/
/***                                                           ***/
/*****************************************************************/
```

Figure 52 (Part 3 of 11). Parser for ISPF Skeletons

```
   PARMLISTX = PARMLIST;
   I = INDEX(PARMLIST,',');
   FXB_OV = SUBSTR(PARMLIST,1,I-1);
   STAT_PTR = PTR_OV;
   PARMLIST = SUBSTR(PARMLIST,I+1,LENGTH(PARMLIST)-I);

   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;
```

Figure 52 (Part 4 of 11). Parser for ISPF Skeletons

```
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;
```

Figure 52 (Part 5 of 11). Parser for ISPF Skeletons

```
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 skeleon.  For each     ***/
/***                line read, increment the appropriate           ***/
/***                counters.                                      ***/
/***                                                               ***/
/*********************************************************************/
   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;
```

Figure 52 (Part 6 of 11). Parser for ISPF Skeletons

```
/*********************************************************************/
/***  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;
/*********************************************************************/
/***  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;
```

Figure 52 (Part 7 of 11). Parser for ISPF Skeletons

```
/*****************************************************************/
/***  If this line starts with )DEFAULT, then the special      ***/
/***  character (the left parenthesis) for control cards may   ***/
/***  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 52 (Part 8 of 11). Parser for ISPF Skeletons

```
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 non-blank after that blank.  ***/
/***             3) Move that non-blank 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;
```

Figure 52 (Part 9 of 11). Parser for ISPF Skeletons

```
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;
   /**/
   /*   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 ');
```

Figure 52 (Part 10 of 11). Parser for ISPF Skeletons

```
            /**/
          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 52 (Part 11 of 11). Parser for ISPF Skeletons

```
/*********************************************************************/
/***                                                               ***/
/***  LISTINFO Structure                                           ***/
/***                                                               ***/
/***  Maps the static portion of the account record.               ***/
/***                                                               ***/
/***  The number of elements declared for this array should not    ***/
/***  be greater that 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 53. 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 54. STATINFO Module

```
*********************************************************************
*
*  Interface program to load R15 with return code from exits and
*  other routines written in PL/I.  This routine is used because
*  PL/I routines cannot pass a return code to a caller through
*  register 15.
*
*********************************************************************
DATRC    CSECT
DATRC    AMODE 31                    SET AMODE
         L     15,0(1)               LOAD ADDRESS OF RETURN CODE FROM
*                                    PL/I ROUTINE
         L     15,0(15)              LOAD RETURN CODE FROM THAT ADDRESS
         L     13,4(13)              LOAD PREVIOUS SAVE AREA
         L     13,4(13)              LOAD THE SAVE AREA ONCE REMOVED
         RETURN (14,12),RC=(15)      GO BACK TO WHOEVER CALLED THE
*                                    PL/I ROUTINE
         END   DATRC                 END OF DATRC SOURCE
```

Figure 55. DATRC Module

Processing Conditionally Saved Components

SCLM provides a feature to conditionally start rebuilds of dependencies for a software component if some outputs produced by the translator were not saved. However, these rebuilds only take place if the translator can indicate to SCLM, by means of its return code, which output data sets were not saved.

For example, suppose a translator can determine if a developer changed only comments in the source code. In such cases, the translator creates a listing output in order for the listings to match the current source. However, creating object code for the source is unnecessary because comment changes to source do not alter object code. Therefore, you do not need to rebuild components dependent on the object code because the object code did not change.

To access this feature, use the FLMALLOC, FLMLANGL, and FLMTRNSL macros. A description on how to use these macros follows.

The FLMALLOC macro defines a temporary data set to be used by the translator. For example, output produced by the translator is saved in these temporary data sets. In this macro, you supply the parameter NOSAVRC. SCLM determines if the translator saved output in a specific temporary data set by comparing the return code value specified on the NOSAVRC parameter with the translator return code. If the NOSAVRC value is equal to a translator return code other than zero, then the build function determines that no output was saved to this data set. Under this condition, build does not save the temporary data set to the SCLM hierarchy. Or, in other words, the build function considers the current version of this output in the SCLM hierarchy to be up-to-date.

Likewise, the translator can directly store output in an external database not under SCLM control. For example, the Ada translator controls output stored in the Ada database, not SCLM. Under such circumstances, the build function requires a signal from the translator to detect whether or not some of the external outputs were saved to an external database. SCLM uses NOSVEXT on the FLMTRNSL macro in the same fashion as the parameter NOSAVRC on the FLMALLOC macro to detect whether or not external outputs were saved.

Because the build function can control whether a translator output data set was saved in the SCLM hierarchy or in an external database, dependencies on this output that were not saved do not require a rebuild. On the FLMLANGL macro, you define the parameter DEPPRCS to enable this feature. If DEPPRCS=N, then SCLM rebuilds dependencies on the software component only if all translator outputs are saved. If DEPPRCS=Y, then SCLM rebuilds dependencies provided there is a good translator return code.

Authorization Code Usage

Authorization codes restrict promotions and drawdowns on a member-by-member basis. This section discusses some uses of authorization codes.

First, some facts about authorization codes:

- An authorization code is a character value up to 8 bytes long.
- When you create the project definition, you assign zero or more authorization codes to each group.
- Each member of every library 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 be one of the authorization codes that have been assigned to the group.
- When you promote a member from one group to the next, the member retains its authorization code. If, as a result, an older version was replaced, the authorization code assigned to that older version is not kept.
- You cannot promote or migrate members into a group that has no authorization codes.

In the example below, there is a simple hierarchy with four groups: RELEASE, TEST, PRIV1 and PRIV2. 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 PRIV1. Group PRIV2 has DEV and L0 as its authorization codes.

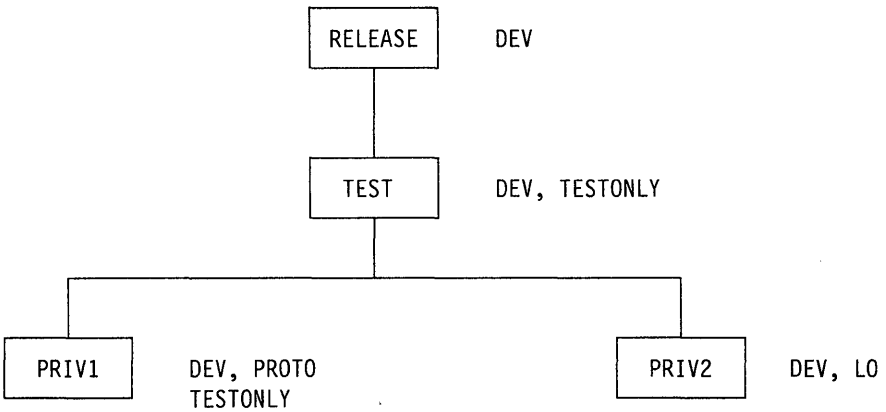


Figure 56. Sample Hierarchy with Authorization Codes

This information can be coded in the project definition as follows:

```
RELEASE  FLMGROUP   KEY=Y,AC=(DEV)
TEST     FLMGROUP   KEY=Y,AC=(DEV,TESTONLY),PROMOTE=RELEASE
PRIV1    FLMGROUP   KEY=Y,AC=(DEV,TESTONLY,PROTO),PROMOTE=TEST
PRIV2    FLMGROUP   KEY=Y,AC=(DEV,L0),PROMOTE=TEST
```

From the example above, we see the following:

- A member in PRIV1 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 PRIV1 with an authorization code of TESTONLY can be promoted to TEST, but cannot be promoted to RELEASE.
- Similarly, a member in PRIV1 or PRIV2 with an authorization code of DEV can be promoted all the way up to group RELEASE.
- A member in PRIV2 cannot have an authorization code of TESTONLY or PROTO—it must be either DEV or L0.

When you edit a member in a development level, SCLM looks at the authorization code you specified and tells you the following:

- If that authorization code is not valid for that development group. You have to enter an authorization code that is assigned to that group. If you ask for help, SCLM shows you a list of valid authorization codes for that group.
- If use of that 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 because of the authorization code you specified.
- If use of that authorization code leads to a potential promotion conflict with another member of the same name. An example of this problem follows.

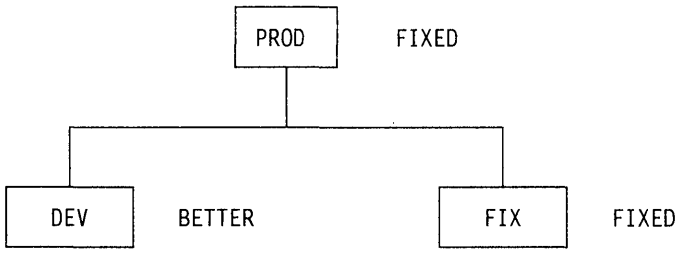
  SCLM allows you to have two members of the same name and type residing in two different development groups (such as PRIV1 and PRIV2 above) under certain conditions. Each of those members has an authorization code assigned to it and those codes, along with the authorization codes assigned to the groups 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 put those members in those libraries. However, if there is at least one group through which both members can be promoted, then changes made to one member are lost when the other member is promoted. In that case, SCLM does not let you put the members in those libraries with those authorization codes.

  If a member exists in group PRIV1, SCLM uses authorization codes to determine whether or not you can edit a member with the same name and type in group PRIV2:

Auth. Code for member in PRIV1	Auth. Code for member in PRIV2	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 PRIV2.
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.

Concurrent Development and Maintenance

We can use the information in the previous section to set up a project in which we can make modifications to what we have in production (development) while being able to make quick fixes to production modules (maintenance). We will use a very simple hierarchy as an example. In reality the hierarchy would have many more groups and levles.



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
```

We have three groups. PROD is our production library, DEV is our development library and FIX is our maintenance library. In practice, there would be a much larger sub-hierarchy under both DEV and FIX in order to allow for multiple developers and to allow 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, there is no authorization code that is assigned to both DEV and PROD. It is this aspect of our project definition that prevents the promotion of any modules from group DEV into group PROD.

If a programmer is going to make changes to a module for the next release of an application, that module would be drawn down from PROD into DEV and would be given an authorization code of BETTER. Changes would be made and tested in DEV. Meanwhile, a user encounters a problem with that application and another programmer determines that the fix requires a change to the module that has been drawn down to DEV.

The module to be fixed can be be drawn down 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 be told to run the application from group FIX until the fix has been verified and moved to PROD.

Before you promote the fix, you should incorporate the changes you made for the fix 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.

Dynamic Include Tracking

General-Use Programming Interface

Dynamic include tracking is a general-use programming interface, which you can use for programming purposes.

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 has the ability to track dynamic includes given the following two conditions:

- The translator produces an output data set containing the list of the dynamic includes.
- 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 given 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 is the record layout used to store the dynamic includes:

```
COUNT    : 4 bytes
MEMBER1  : 8 bytes
TYPE1    : 8 bytes
MEMBER2  : 8 bytes
TYPE2    : 8 bytes
    .
    .
    .
MEMBER#  : 8 bytes
TYPE#    : 8 bytes
```

You must specify the number of dynamic includes in the first four 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. Be sure to remove any duplicate include references.

End of General-Use Programming Interface

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. This capability is powerful but potentially troublesome. Integrity problems can arise through the use of multiple project definitions.

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.

The example on page 258 shows an alternate project definition with a primary non-key integration group defined for the project database structure shown in Figure 57 on page 259.

```
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
GHOST    FLMGROUP AC=(REL),KEY=N,PROMOTE=INT
USER1    FLMGROUP AC=(REL),KEY=Y,PROMOTE=GHOST
USER2    FLMGROUP AC=(REL),KEY=Y,PROMOTE=GHOST
USER3    FLMGROUP AC=(REL),KEY=Y,PROMOTE=GHOST
*
*
*    PROJECT CONTROLS
*
         FLMCNTRL ACCT=PROJ1.ACCOUNT.FILE,                          C
              MAXLINE=75,OPTOVER=YES
*
*
*    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@COBL     -- COBOL          LANGUAGE    --
         COPY  FLM@FORT     -- FORTRAN IV     LANGUAGE    --
         COPY  FLM@PSCL     -- PASCAL         LANGUAGE    --
         COPY  FLM@PLIO     -- PL/I OPTIMIZER LANGUAGE    --
         COPY  FLM@L370     -- 370 LINKAGE EDITOR         --
*
 FLMAEND
```

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 in a primary non-key group. In this way, you do not affect development work going on in the SCLM database. 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 database until their testing is complete. By switching to the alternate project definition, developers and integrators can easily test against the primary non-key configuration. Because the primary non-key level is a non-key group, code still exists in the normal database 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 database. Code developed using this scenario is potentially more complete and accurate than code developed in a normal scenario.

The following two figures compare a default database structure with an alternate database structure. Figure 57 shows an example default database structure for a project. You can perform all normal development activities within this organization.

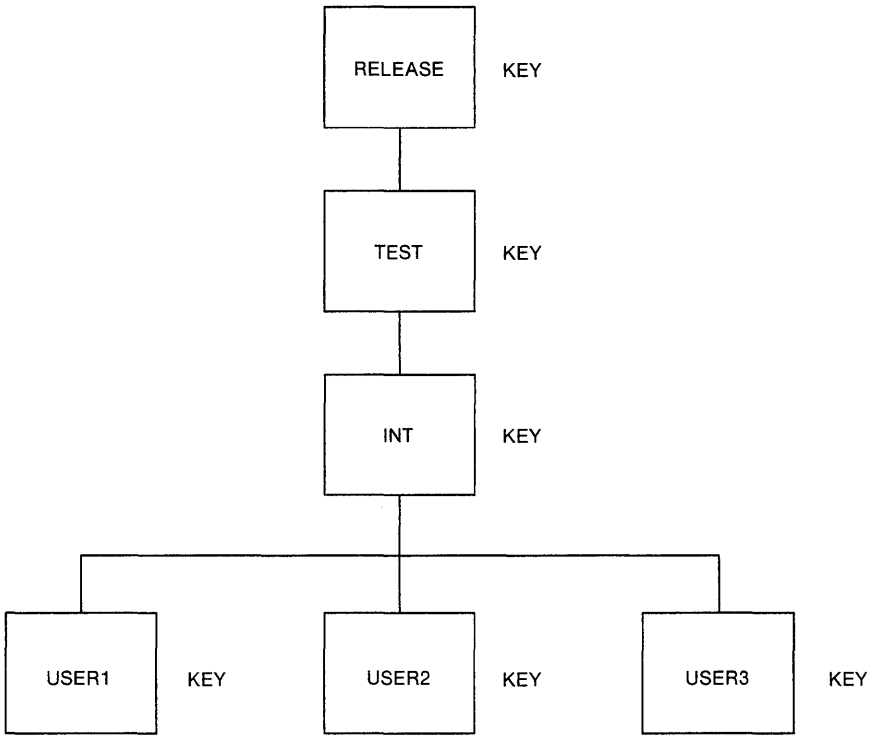


Figure 57. Default (primary) Project Database Structure

Figure 58 shows an alternate database structure with a primary non-key integration group for the project shown in Figure 57.

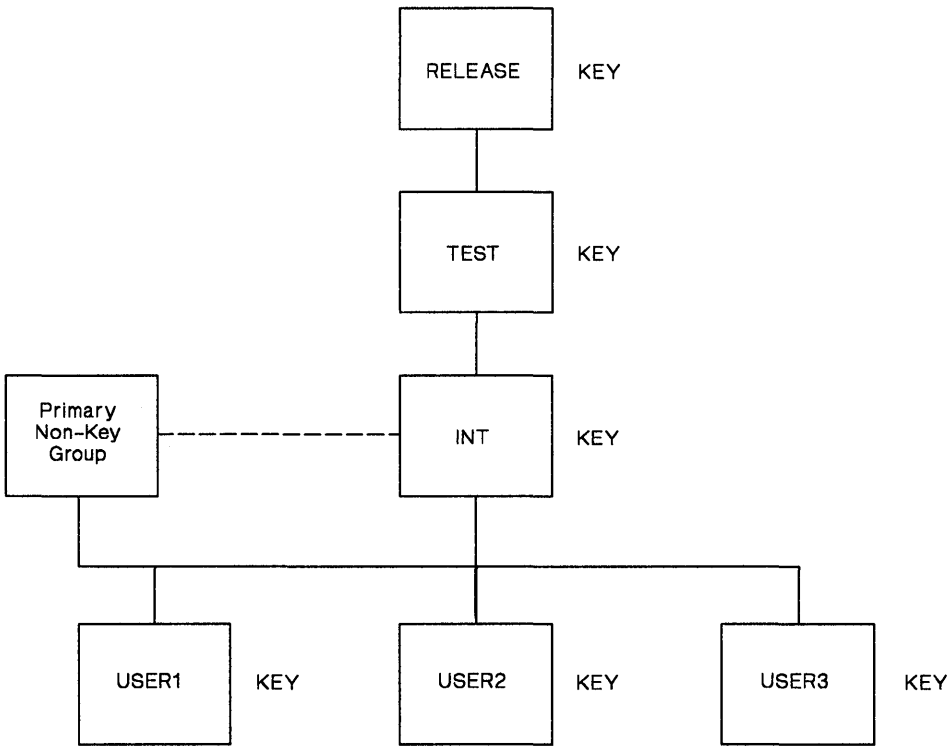


Figure 58. Alternate Project Database Structure with Primary Non-key Integration Group

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 private 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 private libraries into the normal integration group. The promotion to the normal integration libraries causes the components to be deleted from the respective private 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 or through SCLM services.

Change Code Verification Routines

General-Use Programming Interface

A change code verification routine is a general-use programming interface, which you can use for programming purposes.

SCLM calls a change code verification routine to verify all change codes that are entered when you update a member using the SCLM editor, migration utility, or services. See "Change Code Verification Routine Specification" on page 204.

This section explains how to create a change code verification routine.

Change Code Verification Routine Requirements

To validate member updates for a project against a project-defined set of criteria, you can supply a user-generated verification routine to SCLM. If you supply this routine to SCLM, the SCLM editor, migration utility, and SAVE service invoke it. The following paragraphs describe requirements you must follow when designing this routine.

SCLM passes a string of seven parameters separated by commas to the verification routine. Register 1 contains the address of 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 saves members only if it receives a return code of 0 from the verification routine. SCLM informs you if it detects a non-zero return code.

A project can use any combination of the parameters to determine the validity of change codes entered. The format and description of parameters SCLM passes to the verification routine are as follows:

OPTION LIST Up to 256-character parameters specified on the FLMCNTRL macro using the macro parameter VERCCOP. For more information, see "FLMCNTRL Macro" on page 218. Delimit this string so that the SCLM parameters that follow can be identified by the verification routine.

GROUP The eight-character name of the group in which the member is being created or modified (capitalized, left justified, blank padded).

TYPE The eight-character name of the member type being created or modified (capitalized, left justified, blank padded).

MEMBER The eight-character name of the member that is being created or modified (capitalized, left justified, blank padded).

LANGUAGE The eight-character name of the language specified for the member (capitalized, left justified, blank padded).

USERID The eight-character user ID of the developer performing the modification (capitalized, left justified, blank padded).

AUTHCODE The eight-character authorization code for the member (capitalized, left justified, blank padded).

CHANGE CODE The eight-character change code that has been entered (capitalized, left justified, blank padded).

The verification routine can be complicated or simple. You can use a problem report/change request (PR/CR) tracking system to track changes to the application, or you can just maintain valid PR/CR values in a data set that you can access for verification. You should link any routine you produce using linkage editor options RENT and REUS to make the routines reentrant and the invocation as efficient as possible. You can write the verification routine in any language. Use the standard IBM 370 linkage convention.

If the verification routine needs accounting information in addition to the parameters passed by SCLM, include the SCLM internal data access that the DBACCT service provides.

Change Code Verification Routine Example

The following example shows a simple program written in Pascal to perform minimal verification. This routine verifies that the change code has been entered. A return code of 0 indicates that the change code is valid. A return code of 4 indicates that the change code failed verification.

The example calls the Pascal PARMS function to retrieve the string of input parameters. The example calls the Pascal RETCODE procedure to pass the verification routine return code to SCLM in register 15.

The Pascal PARMS function and the RETCODE procedure follow the IBM 370 subroutine linkage convention.

```
PROGRAM EXITCCV;
(*********************************************************************)
(*  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 in register 15.                 *)
(*                         0 -  Change code is valid.                *)
(*                         4 -  Change code is invalid.              *)
(*********************************************************************)
(*  Process:                                                         *)
(*   This program verifies that a change code has been entered.      *)
(*********************************************************************)

VAR
     comma_index      :   INTEGER;
     i                :   INTEGER;
     input_data       :   STRING(120);
     return_code      :   INTEGER;
```

```
BEGIN  (* program EXITCCV *)

     (*  Initialize the variables.  *)
     input_data  :=  PARMS;
     return_code :=  0;

     (*  Parse until you get the change code. *)
     FOR
        i := 1 to 6
     DO
        BEGIN
           comma_index := INDEX(input_data,',');
           input_data  := SUBSTR(input_data,comma_index+1);
        END;  (*FOR*)

     (*  If the change code is blank, signal an error.  *)
     IF TRIM(input_data) = ''
     THEN
        BEGIN
           return_code := 4;
        END;  (*IF*)

     (*  Set the return code.  *)
     RETCODE(return_code);

END.               (* EXITCCV *)
```

End of General-Use Programming Interface

Build and Promote User Exit Routines

General-Use Programming Interface

The build and promote user exit routines are general-use programming interfaces, which you can use for programming purposes.

Specify build and promote user exits to provide additional functions not supplied with SCLM. The sections below provide details on creating build and promote user exit routines. See "Build and Promote User Exit Routine Specification" on page 204.

User Exit Routine Requirements

If you specify a user option parameter, SCLM passes it to the user exit routine, followed by a string of eight parameters separated by commas. The address of this input data is contained at the address contained 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 always considered to be successful and processing continues. Non-zero return code values from user exit routines are handled in the following ways:

- Both the build user exit (BLDEXT1) and the promote purge phase user exit (PRMEXT1) can return any positive integer value and normal processing continues.
- The processing that occurs after the promote verification phase user exit (PRMEXT2) has been invoked depends on the promote mode in effect. In conditional mode, any non-zero return code causes promote processing to terminate. In unconditional mode, any return code greater than zero, but less than 20, allows promote processing to continue.
- The processing that occurs after the promote copy phase user exit (PRMEXT3) has been invoked depends only on the return code value returned. A return code greater than zero, but less than 20, allows normal promote processing to continue. A return code greater than or equal to 20 causes promote processing to terminate regardless of the specified promote mode.

The format and description of the parameters passed from SCLM through all user exits are:

option list Up to 256 characters long. Parameter specified in the FLMCNTRL macro using macro parameter BEXT1OP, PEXT1OP, PEXT2OP, and PEXT3OP. Delimit this string so that the SCLM parameters that follow can be identified by the user exit routine.

'xxxxxxxx' An eight-character literal value indicating the exit type (capitalized, left justified, blank padded). Valid types are:

- **BUILD** Build (BLDEXT1)
- **PVERIFY** Promote verification (PRMEXT1)
- **PCOPY** Promote copy (PRMEXT2)
- **PPURGE** Promote purge (PRMEXT3).

PROJECT The eight-character name of the project (capitalized, left justified, blank padded).

LIBDEF The eight-character name of the project definition (capitalized, left justified, blank padded).

USERID The eight-character value of the user's logon ID.

GROUP The eight-character name of the group (capitalized, left justified, blank padded). The group is the "from level" for the promote and the "build level" for the build.

TYPE The eight-character name of the type (capitalized, left justified, blank padded).

MEMBER The eight-character name of the member (capitalized, left justified, blank padded).

SCOPE The eight-character name of the scope (capitalized, left justified, blank padded). Valid scopes are as follows:

- **Build scope** Limited, normal, subunit, extended.
- **Promote scope** Normal, subunit, extended.

MODE The thirteen-character name of the mode (capitalized, left justified, blank padded). Valid modes are as follows:

Build mode Forced, conditional, unconditional, and report only.

Promote mode Conditional, unconditional, and report.

GROUP The eight-character name of the group (capitalized, left justified, blank padded). The group is the "to-group" for the promote exit routines. This parameter is blank for the build exit routine.

A user exit routine can be complicated or simple. One purpose of a user exit routine is to track changes to an application. The SCLM outputs can be copied and maintained in a data set that you can access for reports. The passed parameters can be used to maintain a time log of SCLM builds and promotes. You should link-edit all user exit routines with the options RENT and REUS to make the routines reentrant and, therefore, the invocation as efficient as possible. You can write a user exit routine in any language.

If you need accounting information in addition to the input parameters passed by SCLM, application program access to SCLM internal data is available using the DBACCT service. See "DBACCT—Retrieve Accounting Records for a Member" on page 122 for more information.

Build and Promote User Exit Output Data Sets

If you specify control options for the build or promote exit routine, SCLM generates a sequential data set containing a record for each member changed or verified by build or promote. 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 (DD) names for build and promote exit output data sets are BLDEXT and PROMEXIT respectively. Following are the attributes of the output data sets; they 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. All fields are separated by a blank. The following list defines the fields generated for every exit routine:

GROUP This field specifies the 8-character name of the group beginning in column 1.

TYPE This field specifies the 8-character name of the type beginning in column 10.

MEMBER This field specifies the 8-character name of the member beginning in column 19.

STATUS This field gives the 16-character status beginning in column 28.

BUILT This field indicates whether or not the specified member was successfully built. This is written by BLDEXT1.

PROMOTABLE/NOT PROMOTABLE
These fields indicate whether or not the member is eligible for promotion (written by PRMEXT1).

COPY SUCCESSFUL/COPY FAILED
These fields indicate whether or not the member was successfully copied (written by PRMEXT2).

PURGE SUCCESSFUL/PURGE FAILED
These fields indicate whether or not the member was successfully purged (written by PRMEXT3).

The following is an example of a build user exit output data set:

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

User Exit Routine Example

An example program written in Pascal to perform minimal user exit activity follows. This routine writes the passed parameters to the data set PROMOUT1, copies the user exit output data set contents to the PROMOUT1 data set, and passes a return code of zero (0) to SCLM.

The program calls the Pascal PARMS function to retrieve the string of input parameters. It calls the Pascal RETCODE procedure to pass the verification routine return code to SCLM in register 15. The Pascal PARMS function and RETCODE procedure assume the IBM S/370 subroutine linkage convention.

```
PROGRAM EXIT001;
(*********************************************************************)
(*   Promote User Exit                                               *)
(*********************************************************************)
(*  Inputs:                                                          *)
(*    PARMS -                                                        *)
(*     option list    -  Options specified in FLMCNTRL macro.        *)
(*     exit type      -  PVERIFY, PCOPY, or PPURGE literal.          *)
(*     ,project       -  Name of the project.                        *)
(*     ,libdef        -  Name of the project definition.             *)
(*     ,userid        -  User ID performing the promote.             *)
(*     ,group         -  Group the member is being promoted from.    *)
(*     ,type          -  Type the member is being promoted from.     *)
(*     ,member        -  The member being promoted.                  *)
(*     ,scope         -  NORMAL, SUBUNIT, or EXTENDED literal.       *)
(*     ,mode          -  CONDITIONAL, UNCONDITIONAL, or REPORT.      *)
(*     ,group         -  Group the member is being promoted to.      *)
(*                                                                   *)
(*    PROMEXIT        -  Promote user exit output data set.          *)
(*                                                                   *)
(*********************************************************************)
(*  Output:                                                          *)
(*    PROMOUT1        -  Output text file contains promote log       *)
(*                       info for this promote phase.                *)
(*                                                                   *)
(*    return_code     -  Return code in register 15.                 *)
(*                       0 - Successful.                             *)
(*********************************************************************)
(*  Process:                                                         *)
(*   This program saves the contents of the PROMEXIT file.           *)
(*********************************************************************)

VAR
     out_file         :   TEXT;
     in_file          :   TEXT;
     parm_string      :   STRING(100);
     line             :   STRING(52);
```

```
BEGIN  (* program EXIT001 *)

   (*  Open the file for write *)
   REWRITE(out_file,'DDNAME=PROMOUT1');

   (*  Open the file for read  *)
   RESET(in_file,'DDNAME=PROMEXIT');

   (*  Retrieve input parameters and write them to the output file *)
   parm_string := PARMS;
   WRITELN(out_file,'User exit 1 entered.');
   WRITELN(out_file,'Parms=',TRIM(parm_string));
   WHILE NOT EOF(in_file) DO
      BEGIN
         READLN(in_file, line);
         WRITELN(out_file,line);
      END;

   (*  Close both files and set the program return code  *)
   CLOSE(out_file);
   CLOSE(in_file);
   RETCODE(0);
END.
```

End of General-Use Programming Interface

Project Conversion 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 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 resides in one logical place and code under development resides elsewhere, then you have at least a two-level hierarchy. Before migration can begin, you must place the source code to be converted into partitioned data sets.

The advantages of using the method above are many. 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 get an indication of the performance that you can expect of SCLM during the development process.

Prerequisites for Existing Hierarchies

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

- 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 Project Data Sets" on page 193, and copy data from the existing data sets to the partitioned data sets.

- Create the project definition to be used with the converted hierarchy. See Chapter 7, "Defining the Project," for details.
- Delete all unnecessary data from the libraries being converted. If you cannot identify the unnecessary data, SCLM can help, as you will see later in this section.
- If you intend to use non-key groups in the converted hierarchy, ensure that they do not contain any data prior to 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 private 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 will be visible during the first group conversion. For more information, see "Build Function" on page 14.

To determine which architecture members you need:

1. Determine whether all translations 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 nontranslatable 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 now defines, through its dependencies, the entire application architecture.

Once you create the architecture members for the topmost group, you may need to add modifications in the lower groups of the hierarchy. Members that were added during the development process, which 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 2, "Architecture Definition," for a description of the process and syntax for defining architecture members.

Register Existing PDF Members with SCLM

Specifically, editable members and non-editable 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 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 non-editable members. Object code, listings, and load modules are examples of non-editable members. The SCLM build function must be called to create all of the non-editable members to be tracked within the hierarchy. If all of the customization related to language translators has been completed and tested, run the build processor in the unconditional mode using the topmost architecture member for your application. 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.

Initialize Non-key Groups

Initialize non-key groups using the promote function once the group promoting into the non-key group has been built successfully. Initially, the non-key group cannot contain any members. Use the conditional mode of the promote function to initialize non-key groups. Run the promote function using the topmost architecture definition to initialize non-key groups.

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 can occur for editable members. Errors can be caused when an included member or macro cannot be found within the hierarchy. If you want the included 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, can be located during the build process. Errors found at the upper groups of the hierarchy may have been fixed by versions existing in lower groups. If this is the case, once the correct version is converted, it is eligible for promotion to the group requiring the fix. If a correct version is not present in the hierarchy, the incorrect version may be fixed in place or introduced at the bottom of the hierarchy.

Security

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 database security. Consider limiting authority to data sets in the hierarchy above the development level as follows:

- All developers require READ authority to these data sets.
- The build coordinator responsible for promotes requires UPDATE authority.

SCLM works cooperatively with the security system to provide total control of all software components.

Backup and Recovery of Project Database

SCLM does not have built-in backup and recovery for a project database. Use standard IBM utilities for backup or recovery. You need to use a manual process to coordinate this activity and to ensure data integrity.

Use what is convenient for your project to create backup copies of the SCLM-controlled data. You could use IEBCOPY utilities to write to tape or HSM to write to a backup disk. The important point is that the entire database is synchronized. Therefore, you must save and restore it as a unit. For example, the source, object, load, and listing data sets are a matched set along with their associated internal data in the VSAM data sets. Therefore, you must back up and restore all of these data sets in a coordinated fashion. The VSAM data set contains the internal data for the entire project database. Restoring it implies that all project data sets are being restored.

The recommended procedure for backing up the project database is to run a background job when no one is working with the database. 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. Therefore, repeated backups of that group accomplish nothing. You should always back up the topmost level immediately after an integration of production code takes place, but do not back it up when the level is frozen. The lower groups in the hierarchy are subject to change much more often, and the private library code usually changes daily. You can best determine which groups need to be backed up and when. Again, remember that you must back up the entire group as a unit, including the internal data in the VSAM data set.

Be careful when recovering a project database. 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 version. Also, the VSAM data set, if restored completely, reflects the status of the entire database when that data set was saved.

Synchronizing Accounting Data Sets

The SCLM FLMCNTRL macro allows you to select dual accounting data sets to be maintained. In the event that a non-recoverable problem occurs with one of the accounting data sets, use the following JCL to synchronize the two accounting data sets. You can use the same JCL to back up or restore an internal data set.

```
//jobname  JOB (wkpkg,dpt,bin),'name'
//*************************************************************
//*                                                           *
//*  JCL TO INITIALIZE/SYNCHRONIZE BACKUP ACCOUNTING FILE     *
//*                                                           *
//*************************************************************
//STEP1    EXEC PGM=IDCAMS
//INPUT    DD DISP=OLD,DSN='PROJ1.ACCOUNT.FILE'
//OUTPUT   DD DISP=OLD,DSN='PROJ1.ACCOUNT2.FILE'
//SYSPRINT DD SYSOUT=H
//SYSIN    DD *
 REPRO INFILE(INPUT) OUTFILE(OUTPUT)
/*
//
```

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.

Dependency Processing Implementation

SCLM supports three kinds of dependencies, which are derived from the parsing of a member. They are:

- Include
- Compool
- Compilation unit.

This information is stored as SCLM internal data, and it enables SCLM to process members in the correct order.

The following describes the processing involved for each dependency:

- Include

  Include dependencies exist where members are required for the proper construction of the including member. In other words, a member and its included members are built (compiled) together. A member that cannot be included is defined as a *compilable member*. Both compilable members and included members can have include dependencies.

  Updates to an included member flag both the included member and its including member for rebuild. Likewise, if an included member is flagged for rebuild, then its including member is also flagged for rebuild. This ripple effect eventually flags the compilable member (the topmost member in this chain) for rebuild.

- Compool

  Compool dependencies for an up-to-date member are translator-produced outputs from other software components that require processing before the member is processed. SCLM allows compool dependency processing only if the field COMPOOL=Y is set on the FLMLANGL macro for the language definition of the member containing these dependencies. SCLM originally implemented this kind of dependency to handle the processing order for JOVIAL programs. However, this dependency can be used for other languages also.

SCLM supplies only the member names of the compools in the dependency list for a member. Therefore, a developer must specify the type name of the compool. Specifying the compool type depends on whether you use an architecture member to define the software component for the member containing the compool dependencies. If you use an architecture member, specify the compool type with the CREF keyword. If you do not use an architecture member, SCLM derives the software component from the language definition of the depending member. In the latter case the compool type is specified with the DFLTCRF parameter on the FLMLANGL macro.

Note: Because you can specify only one type, all compool dependencies must reside in the same type.

Once the compool has been identified, SCLM determines from its internal data the (last) software component used to build the output. This fact implies that the software component must have been built the first time in order for SCLM to contain internal data information on the compool. This software component is then processed only if it falls within the architecture definition selected to be processed.

- Compilation Unit

  Compilation units are subunits residing in a member. Compilation units can have dependencies on one another. However, SCLM treats these dependencies as dependencies between the members containing the compilation units. SCLM originally implemented this kind of dependency to handle the processing order for Ada programs.

  Two kinds of dependencies can exist between two compilation units: upward and downward dependencies. An *upward dependency* identifies those members that SCLM must process *before* processing of the member containing the dependencies. A *downward dependency* identifies those members that SCLM must process *after* processing the member containing the dependencies.

  SCLM can identify the software component containing the dependent compilation unit due to the information it stores as internal data. For example, SCLM can determine that member X contains compilation unit X1 and that compilation unit X1 has a dependency (upward or downward) on compilation unit Y1, which resides in member Y. Thus SCLM concludes that member X has a dependency on member Y. Once it finds the dependent member, SCLM determines whether an architecture member is being specified to process the member. First, it checks whether any of the architecture members within the architecture definition reference the member. If so, it identifies the architecture member as the software component to process. Otherwise, SCLM determines if the associated intermediate form for the dependent compilation unit has been built and, if so, retrieves the name of the software component used to build it (similar to compool dependencies). However, if the intermediate form was never built, SCLM creates the software component from the language definition of the dependent member.

Development and Performance

SCLM services provide the means to extend an existing development environment with SCLM functions. Development and performance considerations for using the SCLM services are discussed in this section. The functions provided by the services and their interfaces are described in Chapter 5, "SCLM Services."

Development Scenario

SCLM users who perform their development activities with ISPF are provided with an SCLM development environment. SCLM has an ISPF dialog interface that provides access to all of the SCLM services. Especially convenient is the integration of the LOCK and SAVE services into the ISPF/SCLM editor. Not all developers requiring configuration management of their host software are ISPF/SCLM users. For example, someone might use a PC connected to a host. Using a programmable workstation (PWS), a developer can construct host software but use editors, analysis tools, text formatters, and other tools in a non-ISPF environment.

The following development scenario lists the basic steps that you can take to use SCLM services and maximize productivity using a programmable workstation.

Step 1: Lock a member
Step 2: Update or create a member
Step 3: Start the SAVE service
Step 4: Build a member
Step 5: Unlock a member
Step 6: Promote a member.

Step 1: Lock a Member: Before you make any modifications to a member, you need to lock it. For information on why locking a member is necessary, see "Edit Function" on page 10. To lock a member, enter the following command:

```
FLMCMD LOCK,PROJ1,,USER1,SOURCE,PROGRAM1,,USERKEY
```

If the LOCK service completes successfully, you now have member PROGRAM1 locked in the USER1 private library, and you can proceed with the modifications. The access key is USERKEY.

If the LOCK service does not complete successfully, it returns a non-zero return code and sets a message in $msg_array.

Step 2: Update or Create a Member: Now that the member is locked, download member PROGRAM1 if it exists in the hierarchy. Once you download it, you can use any tools to update it.

Step 3: Start the SAVE Service: When modifications are complete, upload member PROGRAM1 back to the host and into the USER1 private library. At this point, call the SAVE service to parse the member and store the member's statistical, dependency, and historical information. If PROGRAM1 is a FORTRAN program, the command invocation is as follows:

```
FLMCMD SAVE,PROJ1,,USER1,SOURCE,PROGRAM1,,USERKEY,,FORTRAN
```

Note that you must specify the access key to call the SAVE service.

Step 4: Build (or compile) a Member: After modifying a member, you can compile it. The compilation process is simplified under SCLM by the BUILD service. To compile PROGRAM1, enter the following command:

```
FLMCMD BUILD,PROJ1,,USER1,SOURCE,PROGRAM1
```

Note that the access key is not an input parameter to the BUILD service. If compilation errors occur and the build fails, return to "Step 2: Update or Create a Member" on page 274 to correct the errors.

Step 5: Unlock a Member: If the build was successful in Step 4, you can promote PROGRAM1 so that other developers can access it and possibly make further enhancements. Before you can promote a member, the member must have a blank access key. To remove an access key, call the UNLOCK service as follows:

```
FLMCMD UNLOCK,PROJ1,,USER1,SOURCE,PROGRAM1,USERKEY
```

Step 6: Promote a Member: After removing the access key, promote member PROGRAM1 using the following command:

```
FLMCMD PROMOTE,PROJ1,,USER1,SOURCE,PROGRAM1
```

SCLM relieves much of the burden from programmers in the development process. But the burden could be even further reduced by providing an automated interface to the previously described steps, therefore eliminating the need to enter long commands. Candidates for automation include the following:

- Automated LOCK service invocation and member download
- Automated upload of the member to the host and SAVE service invocation
- Automated BUILD service invocation
- Automated UNLOCK and PROMOTE service invocation.

Projects not using the ISPF dialog interface for member updates should always use a unique, nonblank access key for each developer. Use the developer-unique access key to guarantee that the developer who locked the member is the only person replacing the member when modifications are complete.

In the following situation, access keys are not used:

1. Developer 1 and developer 2 are both working on members in the same private library.
2. Developer 1 locks member PROGRAMX at a development group using a blank access key, copies the member to his PWS, and begins modifying the member.
3. Shortly afterward, developer 2 locks member PROGRAMX at the same development group also using a blank access key, copies the member to his PWS, and begins making a different set of modifications.

In this example, both users assume that they have exclusive use of member PROGRAMX. Unfortunately, the first set of modifications returned to the host is lost when the second set of modifications is returned to the host.

Data Set Protection

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, SCLM enqueues data sets before performing any updates.

The enqueue SCLM uses is identical to the one ISPF uses. If you do not use the ISPF dialog to invoke SCLM services, use an enqueue that matches ISPF and SCLM before updating a partitioned data set.

The following is a description of the ISPF and SCLM ENQ macro parameters:

qname	SPFEDIT
control	E (exclusive)
scope	SYSTEMS
rname length	44
rname	44 character buffer in the format `project.group.type`

For example, to replace a member in the USER1 private library with SOURCE type for project SAMPPROJ1, the ENQ macro may look as follows:

```
ENQ(QNAME,RNAME,E,44,SYSTEMS)
.
.
.
QNAME  DC CL8'SPFEDIT '
RNAME  DC CL44'SAMPPROJ1.USER1.SOURCE'
```

For more information on the ENQ and DEQ macros, refer to *MVS/XA Supervisor Services and Macro Instructions*, GC28-1154.

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 a project's project definition load module into memory and initializes (or opens) the project databases. Both of these functions take time. Therefore, to optimize the SCLM services execution time, minimize the number of START and INIT service calls.

You can reduce the number of START and INIT 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 service 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 service 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.

Workstation Platform for OS/2

The Workstation Platform for OS/2 provides a programmable workstation (PWS) interface to the SCLM host library of ISPF/PDF. A PWS can be a Personal System/2 (PS/2) or an equivalent workstation.

The Workstation Platform consists of three main functional areas:

Facilities	Functions provided
Library List	The library list provides an interactive interface on the PWS to SCLM on the host. It displays lists of available libraries and library members, and allows for the invocation of *tools* to work on members. The available tools may differ depending on the library types selected. A central site coordinator can create a standard set of tools and distribute it to users of the Workstation Platform for OS/2. In addition, you can easily add your own tools to the system and site-supplied set.
Tools	Tools are programs that operate on SCLM members. They can be called by using the library list, rather than by other tools. Tools may call Workstation Platform for OS/2 *services* and are invoked as OS/2 commands.
Library Services	Library services perform SCLM library functions.

For complete information on the library lists and the library services provided, refer to *ISPF/PDF User's Guide and Reference for the Workstation Platform for OS/2 User's Guide*.

By using the facilities of the Workstation Platform for OS/2, an organization can take advantage of both the library control facilities of SCLM and the tools available for OS/2. In addition, tools can be written to integrate directly with SCLM from the PWS.

You and integrated tools can use the Workstation Platform for OS/2 to access the contents of the host SCLM development level as if the PWS and the host were the same system. For example, references to controlled library members are always made using the host entry level library name, even though the member may actually exist (as a copy) at the PWS.

Since Workstation Platform for OS/2 uses HLLAPI as its PWS-host communication vehicle, tools that require upload and download services can use the SEND and RECEIVE services provided with the OS/2 Communications Manager.

A system view of the Workstation Platform for OS/2 is shown in Figure 59.

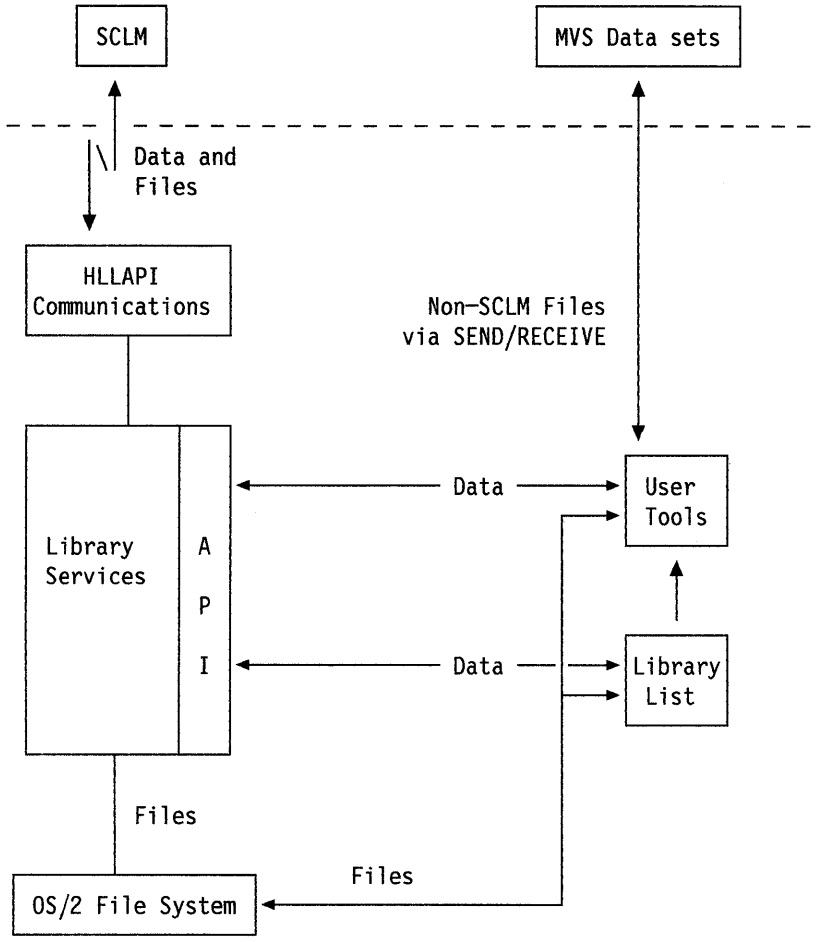


Figure 59. Workstation Platform for OS/2 System Overview

The SSI Field in Load Module Directories

SCLM uses the 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: 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.

```
SSI field  |-----------------------------------------------|
           0 1 ---- 5 6 ------ 11 12 ------------------ 31

         flag   hour      minute          Julian date
         bit    binary    binary          packed decimal
```

Chapter 10. Language Restrictions

There are restrictions to the support SCLM provides for languages. This chapter describes what these restrictions are.

SCLM Parser Restrictions

The SCLM parsers gather statistics on various language constructs. See "PARSE" on page 11 for more information. This section describes the constructs that the SCLM parser 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, compool, or compilation unit references. The existence of extra or missing includes, compools, and compilation units can result in latent errors occurring in the build and promote processors.

SCLM does not support three general types of language constructs due to internal differences. Each of these differences involves include and compool references. The constructs discussed in this chapter are:

- Cross-section references
- Non-explicit references
- Separation of references.

Cross-Section References

You can apply include references to member names only. This restriction applies to user-defined parsers as well as to SCLM-supplied parsers. SCLM always assumes that the type name of the include or compool reference is the same as the member in which the reference was found. Therefore, SCLM does not support include and compool references for members outside the type in which a member is located. For example, SCLM does not support the following PL/I source code statement:

```
%INCLUDE PROJ1.LIB1.SECT1(MEM1)
```

SCLM does not support similar constructs in other languages.

Non-explicit References

SCLM-supplied parsers do not support include and compool 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 or compool 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 internally defined. All include references must exist as

© Copyright IBM Corp. 1989, 1990

SCLM members, or they must exist in FLMSYSLB for build and promote to verify them.

- **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 include reference that SCLM does not support:

```
.set count = 1
.im member&count.
```

- **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

You *must* separate include and compool reference verbs of a language from referenced member names with blanks only, and they must appear on the same line.

SCLM does not support the following Pascal source statement because a comment separates the referenced member name.

```
%INCLUDE (* comment *) SEGNAME;
```

The include reference verb and the reference name must reside on the same line. SCLM does not support the following Pascal statement:

```
%INCLUDE
INCLSEG ;
```

Ada Language Restrictions

There are a few Ada constructs that SCLM does not fully support. To control code containing these constructs, you must take special actions. This part of the chapter describes these unsupported constructs and how to use SCLM to control code containing them.

Generic/INLINE Specification Ordering

SCLM currently requires the IS SEPARATE declaration of Generic and PRAGMA(INLINE) subunits to be declared before the IS SEPARATE declarations of subunits that call them. An *IS SEPARATE declaration* is the specification in an Ada compilation unit that a subunit is to be compiled separately. See the explanations of upward and downward dependencies on page 64.

This restriction lets SCLM properly order the compilation of Ada subunits. If you do not follow this restriction, compilations of generic subunits may fail, and compilations of PRAGMA(INLINE) statements may be ignored by the compiler.

Generic/INLINE Recursive Dependencies

The current version of SCLM does not allow a body of a compilation unit containing PRAGMA(INLINE) or Generic constructs to have dependencies on compilation units which in turn have direct or indirect dependencies on the Generic or PRAGMA(INLINE) compilation unit. An example of such a recursive dependency is a generic package with a body that has a WITH reference on another package, which, in turn, has a WITH reference to the generic package. This example is shown in Figure 60.

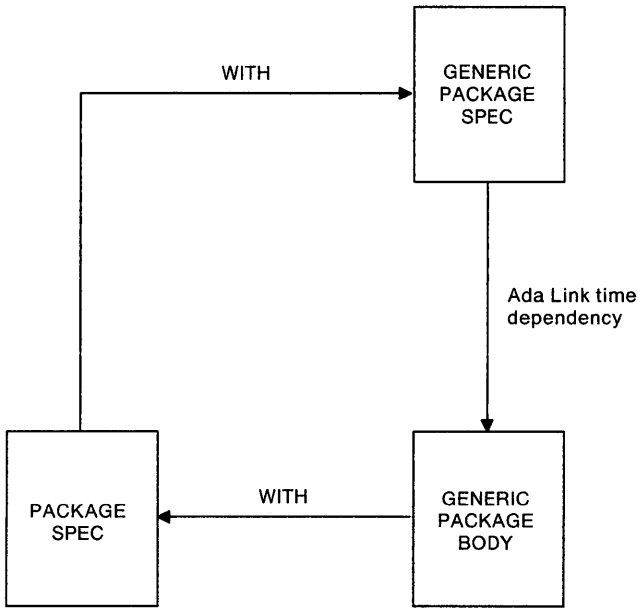


Figure 60. Example of a Disallowed Recursive Generic/Inline Dependency

SCLM can support code that contains recursive generic/INLINE references in two different ways.

First, you can place all compilation units involved in a recursive generic/INLINE reference in a single source member. SCLM only tracks dependencies between source members; SCLM does not use intra-member dependencies and, therefore, they are not a problem to handle.

Alternately, you can place all compilation units involved in a recursive generic/INLINE reference under the control of a single generic architecture member. SCLM does not track intra-architecture member dependencies, so it does not process these recursive references.

Ada Sublibrary Restrictions

SCLM can verify the integrity of a standard SCLM database, but is unable to extend this verification to an SCLM-controlled Ada sublibrary. A *sublibrary* is a data set that contains Ada intermediate form in a form that only the Ada compiler can use. When you update a member in an SCLM database, SCLM can discover this update whether or not you used SCLM to make the change. SCLM does this by verifying certain attributes of the member's directory against the accounting information it maintains for the member. SCLM is unable to make this check for the intermediate form of Ada compilation units.

If you compile Ada source into an Ada sublibrary outside the control of SCLM, then a subsequent build of the code proceeds as if the compile was never performed. Because the compiler updates Ada intermediate form, and Ada intermediate form can be used to create an object module using the Ada binder, it is possible that SCLM will create an object module that does not match the source code SCLM is maintaining in the database.

One of SCLM's primary functions is to ensure that machine-generated code matches its source code. If you make updates to an Ada sublibrary outside SCLM control, however, SCLM does not assure the match. Therefore, if you make updates to an Ada sublibrary without using SCLM, perform them manually and exercise caution.

The following subsections describe SCLM consequences for each kind of Ada sublibrary.

Ada Compilations

You should never make Ada compilations to an Ada sublibrary that is under SCLM control unless you are using SCLM. Compiling into an SCLM-controlled Ada sublibrary without using SCLM causes changes to the database that SCLM cannot discover. Thus, unpredictable results can occur when you perform later Ada compilations using SCLM.

Ada Sublibrary Content Updates

You can copy Ada intermediate form to and purge it from an Ada sublibrary. However, you should not copy to and purge from an SCLM-controlled Ada sublibrary without using SCLM. SCLM cannot discover this kind of change and, therefore, cannot perform the necessary recompilations this change requires. If you need to purge a particular compilation unit, use the SCLM sublibrary utility.

Ada Sublibrary Updates

You can recreate, delete, and rename Ada sublibraries. You should avoid making changes to Ada sublibraries unless absolutely necessary. When you install a new version of a compiler, for example, you may need to delete and recreate all Ada sublibraries for an SCLM project. If you replace an Ada sublibrary for which SCLM has accounting information, you must take special actions.

Whenever you recreate, delete, or rename an Ada sublibrary, you must delete all SCLM accounting information for that sublibrary using the SCLM sublibrary utility. See "Ada Sublibrary Management Utility" on page 68 for details about the SCLM sublibrary utility. The new name of a renamed Ada sublibrary must never match an SCLM-controlled sublibrary.

Multiple SINC Statements

Ada compiler performance can often be enhanced by specifying multiple SINC statements within the same architecture definition. By doing this, all of the source includes (INCL statements) are processed by one invocation of the compiler.

If, however, one or more of the source includes is a generic Ada unit, SCLM considers all of the source includes to be generic. Therefore, if a package references one of the source includes using a WITH language structure, it is referencing a generic Ada unit. If any part of the pseudo-generic package changes, the original package is flagged as out-of-date.

Chapter 11. IBM Ada Setup

Unlike other compilers, Ada compilers maintain their own data sets called sublibraries. To use the Ada language with SCLM, you must create a set of Ada sublibraries. This chapter describes the language definitions you must use and the setup operations you must perform to use the IBM Ada compiler.

The first part of this chapter describes language definitions for the IBM Ada compiler. The second part of the chapter describes the Ada sublibrary setup procedures that you must follow before you can use the IBM Ada compiler with SCLM.

Language Definitions

Ada compilations do not produce object modules. Ada compilations update an Ada sublibrary by storing Ada intermediate form in it. A second process, called Ada binding, produces object modules from Ada intermediate form. Thus, to produce object modules, two sets of invocations and, therefore, two language definitions are required. These language definitions are FLM@ADA and FLM@ADAB.

FLM@ADA

FLM@ADA controls Ada compiles. Ada compiles produce no object modules, but instead produce Ada intermediate form. Ada intermediate form is stored in data sets called Ada sublibraries. SCLM produces accounting records to track Ada intermediate form. These accounting records are called intermediate accounting records. SCLM creates an intermediate accounting record for each Ada compilation unit that compiles successfully. You must compile Ada source code in a certain order because SCLM uses intermediate accounting records to determine when it must recompile Ada source code. In addition, SCLM uses information in Ada source segment accounting records to determine appropriate compilation orders. You can browse this dependency information using the SCLM library utility. You can browse the information in an intermediate accounting record using the SCLM Ada sublibrary management utility.

FLM@ADAB

To produce object modules, you must use an Adabind language. This language produces object modules from Ada intermediate form stored in an Ada sublibrary.

Because the source code for many object modules can exist in an Ada sublibrary, you must specify the name of the compilation unit to be used to create an object module. Specify this compilation unit name by creating a source member that contains only this name and assigns the segment the language of ADABIND (FLM@ADAB). This source member can contain comments, blank lines, or a compilation unit name, in uppercase or lowercase. By default, building an Adabind source segment produces an object module with the same segment name as the Adabind segment.

© Copyright IBM Corp. 1989, 1990

Ada Sublibrary Setup

This section describes the setup operations you must perform before building and promoting source code using the IBM Ada compiler. The IBM Ada compiler requires special handling because not all of the output the compiler creates resides in partitioned data sets.

To use a language that invokes the IBM Ada compiler, you must create an Ada sublibrary for every data set that holds Ada source code. A sublibrary must have the same data set name as its source data set with the project CU qualifier appended to the end of it. A CU qualifier is the name of the compilation unit (CU) qualifier specified in the language definition. For example, for the SCLM data set `PROJ1.DEV1.ADA`, create a sublibrary by the name of `PROJ1.DEV1.ADA.cu_qualifier`.

To create the sublibraries for an SCLM project, perform the following steps:

1. Log on with 4000K of region. Use the Ada compiler to create sublibraries.
2. Make sure that a data set by the name of *userid.ADA.LIBRARY* does not exist. If the data set does exist, alter it in the next step and save it under a different name.
3. Create a sublibrary by issuing the following TSO command:

   ```
   EX 'ADA.V21.CLISTS(A370)' 'DUMMY NOC INIT(/=''PROJ1.DEV1.ADA(cu.qualifier)'')'
   ```

   where `ADA.V21.CLISTS` is the name of the data set where the A370 CLIST is installed, and `PROJ1.DEV1.ADA` is the name of an Ada source data set. You should perform this step for every data set to contain Ada source code to be compiled.

IBM Ada Compiler Restrictions

SCLM does not provide automatic support for all features of the IBM Ada compiler. However, you can overcome these limitations with a manual procedure using SCLM functions. This part of the chapter contains procedures that you can use to provide support for the following IBM Ada compiler capabilities:

- Debugger
- Multiple Load Modules
- Optimizer.

There are other ways to solve these problems depending on how you use the capabilities on a project.

Debugger

The SOURCE command for the debugger supplied with the IBM Ada compiler is not completely compatible with SCLM. The IBM Ada compiler records the source file name of each member compiled with the debug option in the sublibrary. The Ada debugger uses the source file name to list source statements when the "source" command is executed. When an Ada source member is promoted, the source file name of that member changes but the source file name recorded by the IBM Ada compiler does not. Thus the debugger command SOURCE does not work for source members that have been promoted since they were built. To avoid this problem, use the following procedure:

1. Customize language macro FLM@ADAD for use with all debug compiles. The sample language macro FLM@ADAD contains a translator that initializes the source file name stored by the Ada compiler.
2. After each promotion involving Ada source, force a recompilation of the promoted source members. You can accomplish this with the forced mode of the build function. By rebuilding the source member at the higher level, the new source file is established in the sublibrary.

   Note: You only need to perform the rebuild if the promoted code is to be run through the debugger.

Multiple Load Module Support

The IBM Ada compiler has the option of producing multiple object and load modules for one Ada main member. By using the multiple load system sublibrary, the compiler looks for a user-defined Multiple Load Module Definition Data Set (MLMDDS). If this data set exists, then the compiler produces all of the output defined within. If you do not specify this data set, then the compiler automatically produces two object modules: one to run *above the 16 megabyte line* and one to run *below the 16 megabyte line*. From these two object modules, you must generate load modules. Note that if the regular system sublibrary is specified instead, only one object file is generated. You must perform compilations and binds using the same system sublibrary.

Use of Multiple Load Modules

SCLM has the capability to support two object and load modules for each Ada main member under the multiple load system sublibrary. That is, if you use the multiple load system sublibrary to compile and bind an Ada main member, SCLM creates two object and load modules and inserts them into the SCLM hierarchy. If you specify the regular system sublibrary, SCLM creates one object file for the bind, and a separate call to the LE370 translator becomes necessary to create a load module. SCLM does not support the IBM Ada compiler MLM Definition Data Set option.

The following is a step-by-step approach to using multiple load modules.

- Use three separate language definitions:
  - An Ada language definition where you specify xxxxx.xxxxx.xxxxx.MLSUBLIB. This compiles the Ada members in preparation for the multiple load bind step. (See FLM@ADAM in the macro library.)
  - An Ada bind language definition where you specify xxxxx.xxxxx.xxxxx.MLSUBLIB. (See FLM@ADAO in the macro library.) This language definition calls an additional translator after the compiler translator that retrieves from the ADAOBJ ddname the two generated object files and separates them into two temporary data sets. Each data set has a unique ddname associated with it. You should then copy these object files to the hierarchy. This results in two separate object modules. The translator then calls load module FLMTSPLT.
  - A link-edit language definition where you use the object files you created in the previous step as input. (See FLM@ADAL in the macro library.) To create load modules from the generated object files, the following is required:
    - You must create two separate generic architecture members: one for each link. The example below shows you how the lists should appear:

```
* GENERIC LEC TO BUILD AN MLM OBJECT MODULE
CMD   ALIAS XXXXXXXX
SINC YYYYYYYY   AAAAAAAA   * STUB USED TO DEFINE THE LANGUAGE
SINC ZZZZZZZZ   BBBBBBBB   * INPUT TO LE370
OUT1 XXXXXXXX   CCCCCCCC   * BELOW THE LINE LOAD MODULE
OUT2 XXXXXXXX   DDDDDDDD   * LINK MAP
```

      where:

      XXXXXXXX is the member name for the output generated by the build processor.

      YYYYYYYY is a one line stub member with language from FLM@ADAL that must contain the characters SETSSI 00000000 (starting in column 2).

      ZZZZZZZZ is the name of the object module that is to be input to the linkage editor.

      AAAAAAAA is the type name for the associated member name.

      BBBBBBBB is the type name for the associated member name.

      CCCCCCCC is the type name for the associated member name.

      DDDDDDDD is the type name for the associated member name.

    - You should build the generic architecture members to generate the load modules. You can use a high-level architecture member to control these architecture members (along with the compile and bind operations for the main unit). The following example shows a high-level architecture member:

```
* HIGH LEVEL LIST FOR GENERATION OF MULTIPLE LOAD MODULES
INCLD XXXXXXXX  AAAAAAAA   * BIND MEMBER
INCL  YYYYYYYY  BBBBBBBB   * GENERIC FOR LOAD MODULE GENERATION
INCL  YYYYYYYY  CCCCCCCC   * GENERIC FOR LOAD MODULE GENERATION
```

where:

XXXXXXXX is the member name used to identify the main unit to be "bound."

YYYYYYYY is the member name for the generic architecture member.

AAAAAAAA is the type name for the associated member.

BBBBBBBB is the type name for the associated member.

CCCCCCCC is the type name for the associated member.

DDDDDDDD is the type name for the associated member.

- You should define two separate object output types for the user environment: one for *above-the-16-megabyte-line* object modules and one for *below-the-16-megabyte-line* object modules. The first object data set you specify in the language macro (referenced by a DFLTTYP) is used to store below-the-line object modules. The second object data set is used to store above-the-line object modules.
- You should define two separate load output types for the user environment: one for above-the-line load modules and one for below-the-line load modules. The architecture members define these load modules as described in the previous step.
- Finally, you must have two unique language definitions: one for single object and load modules, and one for multiple object and load modules.

The language versions in the architecture definition language definition and FLM@ADAL must be either nonexistent or equal.

Depending on which language definition you use to compile and bind the Ada code, you can choose one or two object and load modules.

Optimizer Support

The IBM Ada compiler has optimization support. The optimizer effectively changes the dependencies between compilation units in a way that SCLM cannot predict. Some Ada compiles using the optimizer might fail due to these changes. If you have dependency problems while using the optimizer, perform a forced build in extended scope to correct the problem. If dependency problems are widespread, consider limiting use of the optimizer to the final stages of the development cycle—such as integration test—if possible. You can then restrict the forced builds to the group used to perform the testing and perform them only after promotions to that group are complete.

When using the IBM Ada compiler, generated load module names must match the corresponding compilation unit names (excluding the underscore, and only for the first eight characters). For example:

```
cu_name = MAIN_PROCEDURE
load module name  = MAINPROC
```

When defining LEC architecture members, the load module name must follow the above naming conventions.

If the above restriction is not acceptable, the generic method described in "Multiple Load Module Support" on page 287 can be used to create an alias load module name.

Part 3. Messages and Codes

© Copyright IBM Corp. 1989, 1990

Chapter 12. Messages and Codes

This chapter provides a complete listing and description of messages and return codes issued by the SCLM functions. ABEND codes are issued with associated error messages. This chapter also lists the codes returned by the SCLM command processor and translators.

Message Organization

The messages are organized alphabetically and ordered by message number. The message descriptions are composed of the following:

1. A message number, in the format FLM*nnnnn*, where *nnnnn* is a numerical identifier.
2. A message explanation, listing causes of printed messages. For error messages, it describes probable causes of the errors. For warning messages, it explains the warnings given. For information messages, messages that are not error or warning messages, the message explanation elaborates on the information presented in the message.
3. A programmer response, which gives return codes, describes possible causes of problems, and discusses how to correct problems mentioned in the message explanation.
4. A database administrator response, which discusses additional ways that a project administrator rather than a programmer can correct problems mentioned in the message explanation.

© Copyright IBM Corp. 1989, 1990

Message Variables

Messages can contain one or more variables, which identify specific components that cause SCLM to generate a message. For example, these variables can be the name of a member, group, or type. Variables with a length of more than eight characters are truncated to three characters and the maximum length is indicated in parentheses. Table 7 lists the maximum length for each variable.

Table 7. Message Variables

Variable	Maximum Length
ACCESS KEY	16
ACCOUNTING GROUP	8
ARCHITECTURE MEMBER	8
AUTHORIZATION CODE	8
BUILD MAP	8
CHANGE CODE	8
CODE	3 or 4
COMMAND	60
COMPOOL NAME	8
COPYLIB NAME	44
CU NAME	55
CU QUALIFIER	8
CU TYPE	4
DATA SET NAME	8 or 26
DATA TYPE	10
DATE	8
DDNAME	8
DISP	4
DEPEND NAME	55
DSNAME	44
EXIT	2
FILE NUMBER	2
FLAG	4
GROUP	8
ID or USER ID	8
INCLUDE NAME	8

Table 7. Message Variables

Variable	Maximum Length
KEYREF	8
KIND	4
LANGUAGE	8
LENGTH	3
LINE	4
LRECL	8
MEMBER	8
MODULE	8
NUMBER	3 or 10
NUMREC	6
PARAMETER	24
PREFIX USER ID	17
PROJECT DEFINITION NAME	8
QNAME	8
RECFM	8
REPORT	21
RETURN CODE	8 or 26
ROUTINE NAME	8, 16, or 40
RNAME	60
SERVICE NAME	9
STATUS	8
TIME	8
TRANSLATOR	8 or 16
TYPE	8
VERSION	8

Messages

FLM00101 MEMBER NAME IS BLANK

Explanation: You left the MEMBER field blank.

Programmer Response: Verify that the member parameter was specified and is in the correct position.

Project Administrator Response: None.

FLM00102 SEVERE ERROR OCCURRED AT *aaa(40)*
CODE: *bbb*

Explanation: The message identifies the name of the SCLM routine, which failed unexpectedly, and the return code.

Programmer Response: None.

Project Administrator Response: Contact SCLM Program Support.

FLM01001 ERROR RETRIEVING ACCOUNTING INFORMATION,

CODE: *aaa* **GROUP:** *bbbbbbbb*
TYPE: *cccccccc* **MEMBER:** *ddddddd*

Explanation: No accounting record exists or could be retrieved for the specified member within the hierarchy beginning in the specified group.

Programmer Response: Possible return codes are:

8 SCLM did not find the member's accounting information. Register the member with SCLM using the SCLM editor, migration utility, or the SAVE service. Run the processor again.

12 SCLM successfully retrieved the member's accounting and dependency information; however, some of the dependency information failed a verification check.

To determine the nature of the verification error, use the library utility to browse the member's accounting and dependency information. The utility performs this check and displays the fields being validated.

To correct the problem, you may need to edit and save the member.

16 SCLM found an invalid group in the project definition. Contact the project administrator.

20 A severe I/O error occurred. Contact the project administrator.

Project Administrator Response: Run IDCAMS against the accounting data set to determine the problem.

FLM01002 ERROR UPDATING ACCOUNTING INFORMATION

CODE: *aaa* **GROUP:** *bbbbbbbb*
TYPE: *cccccccc* **MEMBER:** *ddddddd*

Explanation: An error occurred while attempting to write a member's accounting information. The error code associated with the error message provides specifics regarding the nature of the error.

Programmer Response: Possible return codes are:

4 An I/O error occurred while writing the member's accounting information to the secondary accounting data set. Because the primary accounting data set was correctly updated, SCLM will use the correct information for all references. However, the two accounting data sets are no longer identical. Contact the project administrator.

8 The number of dependent members (compools, included members, and/or compilation units) referenced in the source member exceeds the maximum allowed by SCLM. Consequently, the accounting information was not written.

Change the member so that the number of referenced dependents is decreased below the maximum supported. Delete unnecessary change codes and user data.

12 Contact SCLM Program Support.

20 An I/O error occurred while writing the member's accounting information to the primary accounting data set. The failure to create accounting information implies that SCLM will not be able to track the member. Resubmit the job and if the error recurs, contact the project administrator.

Project Administrator Response: Run IDCAMS against the accounting data set to determine the problem. If the secondary accounting data set has been damaged, reallocate it and initialize with data from the primary accounting data set.

FLM01003 ERROR PURGING ACCOUNTING INFORMATION,

CODE: *aaa* **GROUP:** *bbbbbbbb*
TYPE: *cccccccc* **MEMBER:** *ddddddd*

Explanation: A VSAM error occurred while the system was deleting the accounting record specified.

Programmer Response: Resubmit the job. If the problem recurs, contact the project administrator.

Project Administrator Response: Run IDCAMS against the accounting data set to determine the problem.

FLM01004 ERROR RETRIEVING ACCOUNTING INFORMATION,

CODE: *aaa* **GROUP:** *bbbbbbbb*
TYPE: *cccccccc* **MEMBER:** *ddddddd*

Explanation: No accounting record exists or could be retrieved for the specified member in the given group.

Programmer Response: Possible return codes are:

8 The member's accounting information was not found. Introduce the member to SCLM using the SCLM editor, migration utility, or SAVE service. Run the processor again.

12 The member's accounting and dependency information was successfully retrieved; however, some of the dependency information failed a verification check. To determine the nature of the verification error, browse the member's accounting and dependency information using the SCLM library utility. The utility performs this check and displays the fields being validated. To correct the

© Copyright IBM Corp. 1989, 1990

problem, you may need to edit and save the member.

20 A severe I/O error occurred. Contact the project administrator.

Project Administrator Response: Run IDCAMS against the accounting data set to determine the problem.

FLM01011 ERROR RETRIEVING ACCOUNTING OR CROSS-REFERENCE INFORMATION

CODE: *aaaaaaaa* **ERROR GROUP:** *bbbbbbbb*
TYPE: *cccccccc* **MEMBER:** *dddddddd*

Explanation: An error occurred while attempting to retrieve a member's accounting or dependency information.

Programmer Response: Possible return codes are:

8 The member's accounting information was not found. Introduce the member to SCLM using the SCLM editor, migration utility, or SAVE service. Run the processor again.

12 The member's accounting and dependency information was successfully retrieved; however, some of the dependency information failed a verification check. To determine the nature of the verification error, browse the member's accounting and dependency information using the SCLM library utility. The utility performs this check and displays the fields being validated. To correct the problem, you may need to edit and save the member.

16 SCLM found an invalid group in the project definition. Contact the project administrator.

20 A severe I/O error occurred. Contact the project administrator.

Project Administrator Response: Run IDCAMS against the accounting data set to determine the problem.

FLM01012 ERROR UPDATING ACCOUNTING OR CROSS-REFERENCE DATA SET INFORMATION

CODE: *aaa* **ERROR GROUP:** *bbbbbbbb*
TYPE: *cccccccc* **MEMBER:** *dddddddd*

Explanation: An error occurred while attempting to write a member's accounting and dependency information.

Programmer Response: Possible return codes are:

8 An I/O error occurred while writing the member's accounting information and no attempt was made to write the dependency information. Errors may occur if SCLM attempts to reference this member. Resubmit the job, and if the error recurs, contact the project administrator.

12 An I/O error occurred while writing dependency information for a compilation unit. Errors may occur if SCLM attempts to reference this member. Resubmit the job, and if the error recurs, contact the project administrator.

Project Administrator Response: Run IDCAMS against the accounting and cross-reference data sets to determine the problem.

FLM01013 ERROR PURGING ACCOUNTING OR CROSS-REFERENCE DATA SET INFORMATION

CODE: *aaa* **ERROR GROUP:** *bbbbbbbb*
TYPE: *cccccccc* **MEMBER:** *dddddddd*

Explanation: A VSAM error occurred while the system was deleting the accounting record specified.

Programmer Response: Resubmit the job. If the problem recurs, contact the project administrator.

Project Administrator Response: Run IDCAMS against the accounting data set to determine the problem.

FLM01501 ERROR RETRIEVING BUILD MAP INFORMATION

CODE: *aaa* **GROUP:** *bbbbbbbb*
TYPE: *cccccccc* **MEMBER:** *dddddddd*

Explanation: No build map record could be retrieved for the specified member.

Programmer Response: Possible return codes are:

8 The specified build map record does not exist. Build the appropriate architecture member. Invoke the processor again.

12 The format of the data retrieved was incorrect. Delete the build map and build again to regenerate it.

16 An invalid group was found in the project definition. Contact the project administrator.

20 A severe I/O error occurred. Contact the project administrator.

Project Administrator Response: If the return code is:

16 Reassemble the project definition. Verify that no errors occurred. Relink the project definition. For more information, see "Step 10: Assemble and Link Project Definition" on page 206.

20 A VSAM error occurred. Run IDCAMS against the accounting data set to determine the problem.

FLM01502 ERROR UPDATING BUILD MAP INFORMATION

CODE: *aaa* **GROUP:** *bbbbbbbb*
TYPE: *cccccccc* **MEMBER:** *dddddddd*

Explanation: An error occurred while attempting to insert or update build map information in the accounting data set.

Programmer Response: Possible return codes are:

4 An I/O error occurred while writing the member's accounting information to the secondary accounting data set. Because the primary accounting data set was correctly updated, SCLM will use the correct information for all references. However, the two accounting data sets are no longer identical. Contact the project administrator.

8 The length of the build map exceeds the maximum size allowed by the accounting data set.

12 Contact SCLM Program Support.

20 A severe I/O error occurred. Contact the project administrator.

Project Administrator Response: Run IDCAMS against the accounting data set to determine the problem. If the return code is 8, contact SCLM Program Support.

FLM01503 ERROR PURGING BUILD MAP INFORMATION

CODE: *aaa* **GROUP:** *bbbbbbbb*
TYPE: *cccccccc* **MEMBER:** *dddddddd*

Explanation: A VSAM error occurred while deleting the accounting record specified.

Programmer Response: Resubmit the job. If the problem recurs, contact the project administrator.

Project Administrator Response: Run IDCAMS against the accounting data set to determine the problem.

FLM03001 ERROR RETRIEVING CROSS-REFERENCE INFORMATION

CODE: *aaa* **CU NAME:** *bbb(55) ccc(55)*
CU TYPE: *dddd* **CU QUALIFIER:** *eeeeeeee*
GROUP: *ffffffff*

Explanation: SCLM could not retrieve a build map record for the specified compilation unit.

Programmer Response: Possible return codes are:

8 The accounting information for the compilation unit was not found. Register the member with SCLM using the SCLM editor, migration utility, or the SAVE service. Run the processor again.

12 The member's accounting and dependency information was successfully retrieved; however, some of the dependency information failed a verification check. To determine the nature of the verification error, browse the accounting and dependency information for the compilation unit using the library utility. The library utility performs a verification check and displays the fields while validating them. To correct the problem, you may need to edit and save the member.

16 SCLM found an invalid group in the project definition. Contact the project administrator.

20 A severe I/O error occurred. Contact the project administrator.

24 The cross-reference data set was not defined in the project definition. Contact the project administrator.

Project Administrator Response: If the return code is:

20 A VSAM error occurred. Run IDCAMS against the cross-reference data set to determine the problem.

24 Identify the cross-reference data set on the FLMCNTRL macro of the project definition. For more information, see "FLMCNTRL Macro" on page 218.

FLM03002 ERROR UPDATING CROSS-REFERENCE INFORMATION

CODE: *aaa* **CU NAME:** *bbb(55) ccc(55)*
CU TYPE: *dddd* **CU QUALIFIER:** *eeeeeeee*
GROUP: *ffffffff*

Explanation: An error occurred while attempting to insert or update information in the cross-reference data set.

Programmer Response: Possible return codes are:

8 The length of the cross-reference information exceeds the maximum size allowed by the cross-reference data set. Reduce the number of dependencies for the compilation unit.

12 SCLM internal error. Contact the project administrator.

20 A severe I/O error occurred. Contact the project administrator.

24 The cross-reference data set was not defined in the project definition. Contact the project administrator.

Project Administrator Response: If the return code is:

12 Contact SCLM Program Support.

20 A VSAM error occurred. Run IDCAMS against the cross-reference data set to determine the problem.

24 Define the cross-reference data set on the FLMCNTRL macro of the project definition. For more information, see "FLMCNTRL Macro" on page 218.

FLM03003 ERROR PURGING CROSS-REFERENCE INFORMATION

CODE: *aaa* **CU NAME:** *bbb(55) ccc(55)*
CU TYPE: *dddd* **CU QUALIFIER:** *eeeeeeee*
GROUP: *ffffffff*

Explanation: An I/O error occurred while deleting cross-reference information for the compilation unit specified.

Programmer Response: Possible return codes are:

8 A severe I/O error occurred. Contact the project administrator.

16 The cross-reference data set is enqueued. Try the job again later.

24 The cross-reference data set was not defined in the project definition. Contact the project administrator.

Project Administrator Response: If the return code is:

8 A VSAM error occurred. Run IDCAMS against the cross-reference data set to determine the problem.

24 Define the cross-reference data set on the FLMCNTRL macro of the project definition. For more information, see "FLMCNTRL Macro" on page 218.

FLM03021 ERROR ACCESSING ACCOUNTING INFORMATION FOR DEPENDENT COMPILATION UNIT

CU NAME: *aaa(55) bbb(55)* **CU TYPE:** *dddd*
CU QUALIFIER: *dddddddd* **CODE:** *eee*

Explanation: No accounting record could be retrieved for the dependent compilation unit due to an I/O error.

Programmer Response: Resubmit the job. If the problem recurs, contact the project administrator.

Project Administrator Response: Run IDCAMS against the cross-reference data set to determine the problem.

FLM03501 **ERROR RETRIEVING ACCOUNTING INFORMATION FOR INTERMEDIATE FORM OF**

CU NAME: *aaa(55) bbb(55)* **CU TYPE:** *cccc*
CU QUALIFIER: *dddddddd* **CODE:** *eee*
GROUP: *ffffffff* **TYPE:** *gggggggg*
MEMBER: *hhhhhhhh*

Explanation: An error occurred while attempting to retrieve accounting information for the specified intermediate form.

Programmer Response: Possible return codes are:

8 The accounting information for the intermediate form of the compilation unit was not found in any group in the hierarchy defined by *group*. This means that the compiled intermediate form is missing or out of date. The member containing the compilation unit needs to undergo an SCLM build.

12 SCLM internal error. Contact the project administrator.

16 An invalid group was found in the project definition. Contact the project administrator.

20 An I/O error occurred while retrieving the accounting information for the intermediate form of the compilation unit. Resubmit the job and if the error recurs, contact the project administrator.

24 The cross-reference data set was not defined in the project definition. Contact the project administrator.

Project Administrator Response: If the return code is:

12 Contact SCLM Program Support.

16 Reassemble the project definition. Verify that no errors occurred. Relink the project definition. For more information, see "Step 10: Assemble and Link Project Definition" on page 206.

20 A VSAM error occurred. Run IDCAMS against the cross-reference data set to determine the problem.

24 Define the cross-reference data set on the FLMCNTRL macro of the project definition. For more information, see "FLMCNTRL Macro" on page 218.

FLM03502 **ERROR UPDATING ACCOUNTING INFORMATION OF INTERMEDIATE FORM FOR**

CU NAME: *aaa(55) bbb(55)* **CU TYPE:** *cccc*
CU QUALIFIER: *dddddddd* **CODE:** *eeeeeeee*
TYPE: *ffffffff* **MEMBER:** *gggggggg*

Explanation: An error occurred while attempting to update accounting information for the specified intermediate form.

Programmer Response: Possible return codes are:

12 The record format of the member's intermediate accounting data is incorrect for the current version of SCLM. Contact the project administrator.

20 An I/O error occurred while updating the member's intermediate accounting data. Resubmit the job, and if the error recurs, contact the project administrator.

24 The cross-reference data set was not defined in the project definition. Contact the project administrator.

Project Administrator Response: If the return code is:

12 Verify that the cross-reference data set is compatible with the current release of SCLM.

20 Run IDCAMS against the cross-reference data set to determine the problem.

24 Define the cross-reference data set on the FLMCNTRL macro of the project definition. For more information, see "FLMCNTRL Macro" on page 218.

FLM03503 **ERROR PURGING ACCOUNTING INFORMATION OF INTERMEDIATE FORM FOR**

CU NAME: *aaa(55) bbb(55)* **CU TYPE:** *cccc*
CU QUALIFIER: *dddddddd* **CODE:** *eee*
GROUP: *ffffffff* **TYPE:** *gggggggg*
MEMBER: *hhhhhhhh*

Explanation: An error occurred while attempting to purge accounting records of intermediate form.

Programmer Response: Possible return codes are:

8 An I/O error occurred while purging. Resubmit the job. Contact the project administrator if the error recurs.

16 Target data set enqueued. Resubmit the job after the data set is no longer exclusively in use by another job.

24 The cross-reference data set was not defined in the project definition. Contact the project administrator.

Project Administrator Response: If the return code is:

8 Run IDCAMS against the cross-reference data set to determine the problem.

24 Define the cross-reference data set on the FLMCNTRL macro of the project definition. For more information, see "FLMCNTRL Macro" on page 218.

FLM03504 **ERROR RETRIEVING ACCOUNTING INFORMATION FOR INTERMEDIATE FORM OF**

CU NAME: *aaa(55) bbb(55)* **CU TYPE:** *cccc*
CU QUALIFIER: *dddddddd* **CODE:** *eee*
GROUP: *ffffffff* **TYPE:** *gggggggg*
MEMBER: *hhhhhhhh*

Explanation: An error occurred while attempting to retrieve accounting information for the specified intermediate form. The error code associated with the error message provides specifics regarding the nature of the error.

Programmer Response: Possible return codes are:

8 The accounting information for the intermediate form of the compilation unit was not found in the specified group. This means that the compiled intermediate form is missing or out of date. The member containing the compilation unit needs to be rebuilt.

12 SCLM internal error. Contact the project administrator.

20 An I/O error occurred while retrieving the accounting information for the intermediate form of the compilation unit. Resubmit the job and if the error recurs, contact the project administrator.

24 The cross-reference data set was not defined in the project definition. Contact the project administrator.

Project Administrator Response: If the return code is:

12 Contact SCLM Program Support.

20 A VSAM error occurred. Run IDCAMS against the cross-reference data set to determine the problem.

24 Define the cross-reference data set on the FLMCNTRL macro of the project definition. For more information, see "FLMCNTRL Macro" on page 218.

FLM04001 GROUP: *aaaaaaaa* **IS NOT DEFINED IN THE PROJECT DEFINITION.**

Explanation: The specified group is not defined to the project definition.

Programmer Response: Verify that *aaaaaaaa* is the intended group. Verify that the correct project definition name was specified. Contact the project administrator.

Project Administrator Response: Add the group to the project definition. For more information, see Chapter 7, "Defining the Project."

FLM04002 SPECIFIED GROUP: *aaaaaaaa* **IS NOT A DEVELOPMENT GROUP**

Explanation: The specified group is not valid for the function requested and must be defined to SCLM as a development group. A development group is a group that no groups can promote into.

Programmer Response: Select a group that is defined in the project definition as a development group.

FLM04003 TYPE: *aaaaaaaa* **IS NOT DEFINED IN THE PROJECT DEFINITION.**

Explanation: The specified type has not been defined in the current project definition.

Programmer Response: Verify that *aaaaaaaa* is a type that is supposed to contain SCLM data. If so, contact the project administrator.

Project Administrator Response: Add the type to the project definition.

FLM04005 AUTHORIZATION CODE: *aaaaaaaa* **IS NOT DEFINED TO GROUP:** *bbbbbbbb*

Explanation: The specified authorization code has not been defined to SCLM as a valid authorization code for the specified group.

Programmer Response: Use an authorization code that has been defined to the specified group. Contact the project administrator for a list of valid authorization codes. If the specified authorization code is valid, contact the project administrator.

Project Administrator Response: Check the project definition that defines the specified group. The valid authorization codes for the group are defined there. If authorization groups are used, you may need to reference the FLMAGRP macros in the project definition as well. If the authorization code is valid, add it to the project definition.

FLM04006 LANGUAGE: *aaaaaaaa* **IS NOT DEFINED IN THE PROJECT DEFINITION**

Explanation: The specified language is not defined in the project definition used.

Programmer Response: Verify that the language of the member is defined in the project definition. Specify a valid language and resubmit. To determine whether a language is defined, type an invalid language on the SCLM Edit Profile panel using the SCLM editor, and then type HELP twice.

Project Administrator Response: None.

FLM04007 LANGUAGE: *aaaaaaaa* **IS NOT DEFINED FOR MEMBER:** *bbbbbbbb* **TYPE:** *cccccccc*

Explanation: The language is not defined in the project definition used. If this message is received for an existing member, the project definition has probably changed since the last time the source member was modified.

Programmer Response: Verify that the language of the member is defined in the project definition. Specify a valid language and resubmit. To find the valid languages defined to SCLM, type an invalid language on the SCLM Edit Profile panel using the SCLM editor, and then type HELP twice.

Project Administrator Response: None.

FLM04008 ACCOUNTING RECORD FOR MEMBER: *aaaaaaaa* **TYPE:** *bbbbbbbb* **IS IN STATE:** *cccccccc*

Explanation: The member has been locked but not parsed or stored. This error can result when you call the LOCK service or when you edit a member but do not save it.

Programmer Response: Use a new member name or have the owner of the specified member free it.

Project Administrator Response: None.

FLM04009 ACCOUNTING RECORD FOR MEMBER: *aaaaaaaa* **TYPE:** *bbbbbbbb* **IS IN STATE:** *cccccccc*

Explanation: The member has been locked.

Programmer Response: You must unlock the member before it can be edited. It must also not exist in another private library with an accounting record.

Project Administrator Response: None.

FLM05001 EXISTING MEMBERS AUTHORIZATION CODE IS NOT DEFINED TO THE GROUP

GROUP: *aaaaaaaa* **TYPE:** *bbbbbbbb*
MEMBER: *cccccccc*
ERROR GROUP: *dddddddd*
AUTHORIZATION CODE: *eeeeeeee*

Explanation: The authorization code is not defined to the specified group. This implies that the member is not authorized to replace the version of the member in the error group.

Programmer Response: It is possible that the function will succeed with a different authorization code. Contact the project administrator for a list of authorization codes that

are valid for the group. If none of the authorization codes defined to the group work, try the same function at a different group. Contact the project administrator if all attempts fail.

Project Administrator Response: The FLMGROUP macro lists the valid authorization codes defined for this group in the project definition. Do not attempt to add authorization codes to the project definition unless you are familiar with risks outlined in "Authorization Code Usage" on page 252.

FLM05002 **PREDECESSOR VERIFICATION FAILED**
INPUT GROUP : *aaaaaaaa* **TYPE:** *bbbbbbbb*
MEMBER: *cccccccc*
ERROR GROUP1: *dddddddd*
DATE: *eeeeeeee* **TIME:** *ffffffff*
ERROR GROUP2: *gggggggg*
DATE: *hhhhhhhh* **TIME:** *iiiiiii*

Explanation: The version of the member in *dddddddd* was not based on the member in *gggggggg*. During a promotion, this usually means that a version of the member between these two groups has been deleted. If the authorization code is being changed, changes to the member in *gggggggg* will be lost if the version in *dddddddd* is promoted.

The predecessor date and time fields in the accounting information for the member in *dddddddd* should contain the last modified date and time fields for the next occurrence of the member within the hierarchy.

See Chapter 4, "SCLM Dialog Interface," for specific contents of the predecessor date and time fields.

For the promote processor, if *gggggggg* is not the group being promoted into, this message is a warning. However, the promote processor, in conditional mode, prevents the member in *aaaaaaaa* from replacing the member in *gggggggg*.

Programmer Response: For the promote processor, verify that the member in *aaaaaaaa* contains all of the required changes present in the member in *gggggggg*. If it does, and no other promote verification errors are present, promote again in unconditional mode. If other promote verification errors are present, either correct the errors or use an architecture member that controls as few members as possible.

If you have tried to change the authorization code, and the member is in a private library, verify that all of the changes from the version in *gggggggg* have been incorporated in *aaaaaaaa*. Then delete and recreate the accounting information for the member using the SCLM editor or the SAVE service. If *aaaaaaaa* is not a private library, the member must be drawn down to a private library, and you must delete the member in *aaaaaaaa* before using the procedure outlined above.

Project Administrator Response: None.

FLM05010 **MEMBER LOCKED AT ANOTHER GROUP**
INPUT GROUP: *aaaaaaaa* **TYPE:** *bbbbbbbb*
MEMBER: *cccccccc*
ERROR GROUP: *dddddddd*
AUTHORIZATION CODE: *eeeeeeee*

Explanation: The member has already been updated in another hierarchy. The changes currently reside in *dddddddd*, which is not in your hierarchy. You are not allowed to update the member because you would not be working with the most current version of the member.

Programmer Response: Have the member promoted into a group that is in your hierarchy (that is, one that appears on the SCLM Edit - Entry panel). If the member cannot be promoted, the member and its accounting information must be deleted in *dddddddd* using the SCLM library utility or the DELETE service.

Project Administrator Response: None.

FLM05020 **ERROR ALLOCATING HIERARCHY VIEW FOR TYPE:** *aaaaaaaa* **FROM GROUP:** *bbbbbbbb* **CODE:** *ccc*

Explanation: One of the following has occurred:

- The specified type does not exist or is not defined.
- One or more data sets are not allocated in the hierarchy or are allocated with different attributes.
- One or more data sets in the hierarchy are allocated exclusively to another job.

Programmer Response: Resubmit the job and check for the following:

1. Check input parameters and verify that the type exists in the project definition.
2. Verify that all data sets in the hierarchy exist for this type and were allocated with the same attributes.
3. Verify that data sets are not allocated exclusively to another job.

Project Administrator Response: None.

FLM05501 **ARCHITECTURE MEMBER:** *aaaaaaaa* **IN TYPE:** *bbbbbbbb* **IS INVALID.**

GROUP: *cccccccc* **CODE:** *ddd*

Explanation: A problem was encountered with the architecture member specified. Use the return code to identify and correct the problem.

Programmer Response: Possible return codes are:

12 An incorrect keyword was found in the specified architecture member. Verify the contents of the architecture member, and resubmit the job.

20 The specified member could not be allocated. Verify that the member exists and that the data set is not allocated exclusively to another job.

24 A subordinate copy member could not be allocated. Verify that all members referenced through the use of the COPY keyword exist and that the members are not allocated exclusively to another job.

28 An error occurred while attempting to allocate the hierarchy for the type starting in *cccccccc*. Verify that the type *bbbbbbbb* exists in the project definition. Verify that all data sets in the hierarchy exist for this type and were allocated with the same attributes. Verify that data sets are not allocated exclusively to another job.

32 In processing the given architecture member, either the nesting limit of copied architecture members was exceeded, or the architecture definition has a recursive copy and the recursion limit was exceeded. The maximum number of nested copies currently allowed is 75. Modify the architecture definition and resubmit the job.

Other The specified member has an invalid keyword or is not an architecture member. Verify that the specified member is an architecture member and resubmit the job.

Project Administrator Response: None.

FLM06501 TRANSLATOR RETURN CODE FROM ===> *aaa(16)* ===> *bbb*

Explanation: This message identifies the return code received from the specified translator. . If the return code indicates success as defined on the FLMTRNSL macro, all output is saved in the hierarchy and no response is necessary. If the return code from the translator did not meet the GOODRC specified for the translator, SCLM saves translator output, such as compiler listings, in the listings data set for the processor if requested in the language definition.

Programmer Response: Use the listings data set to locate and correct all errors identified by the translator. If the return code from the translator is acceptable and build indicated that the translator failed, contact the project administrator.

Project Administrator Response: Change the GOODRC parameter of the FLMTRNSL macro, which is defined in the project definition.

FLM06502 ERROR INVOKING TRANSLATOR: *aaaaaaaa*, CODE: *bbb*

Explanation: The translator could not be invoked by SCLM. The load module containing the translator may be allocated exclusively to another job. There is a possible error in the language definition that defines the translator.

Programmer Response: If the translator has been used successfully in the past and no changes were anticipated (for example, a new compiler release), invoke the processor again. If the translator is new or the problem recurs, contact the project administrator.

Project Administrator Response: Verify that the parameters of the FLMTRNSL macro, which are defined in the project definition, are correct. For more information, see "FLMTRNSL Macro" on page 228.

FLM06503 PROBABLE SYSTEM/USER ABEND FOR TRANSLATOR: *aaaaaaaa* HEXADECIMAL VALUE OF RETURN CODE: *bbbbbbbb*

Explanation: SCLM issues this message when an ABEND occurs. A translator return code over 4096 indicates an ABEND. SCLM also provides the hexadecimal value of the translator return code.

Programmer Response: Use the information provided in this message to correct the cause of the ABEND, and resubmit the job.

Project Administrator Response: None.

FLM06511 ERROR INVOKING USER EXIT ROUTINE: *aaa(16)*, CODE: *bbb*

Explanation: SCLM could not invoke the user exit. The load module containing the user exit may be allocated exclusively to another job. There is a possible error in the project definition that defines the user exit.

Programmer Response: If the user exit has been used successfully in the past, run the job again. If the user exit is new or the problem recurs, contact the project administrator.

Project Administrator Response: Verify that the user exit executes correctly outside of SCLM. Verify that the user exit is defined correctly in the project definition. For more information on user exits, see "Build and Promote User Exit Routines" on page 263.

FLM06512 VERIFICATION ERROR FROM USER EXIT ROUTINE: *aaa(16)*, CODE: *bbbb*

Explanation: The return code from the user exit invoked did not meet the acceptable criteria specified for the user exit. Output produced will depend on the user exit routine.

Programmer Response: Review the local software configuration management for information about the user exit.

Project Administrator Response: For more information on user exits, see "Build and Promote User Exit Routines" on page 263.

FLM06513 PROBABLE SYSTEM/USER ABEND FOR USER EXIT ROUTINE: *aaa(16)* HEXADECIMAL VALUE OF RETURN CODE: *bbbbbbbb*

Explanation: SCLM issues this message when an ABEND (user exit return code greater than 4096) occurs. SCLM also provides the hexadecimal value of the user exit return code.

Programmer Response: Use the information provided in this message to correct the cause of the ABEND.

Project Administrator Response: None.

FLM07001 AUTHORITY CODE: *aaa* ON DATA SET: *bbb(44)* RESULTED FROM ATTEMPT TO UPDATE DATA. ATTR: *c* MACRO RC: *ddd* EXIT RC: *eee* EXIT REASON: *fff*

Explanation: An attempt was made to perform an SCLM function without the proper authority. Programmers cannot update SCLM internal data, using SCLM functions, unless they have the authority to update the data set to which the internal data is related.

Possible return codes are:

8 LOCATE macro failed.

12 RACROUTE macro failed.

DSNAME
Data set being accessed.

ATTR R: READ, U: UPDATE, C: CONTROL, A: ALTER.

MACRO RC
For a return code of 8, this value contains the return code from the LOCATE macro. For a return code of 12, this value contains the return code from the RACROUTE macro. Otherwise it is set to zero.

EXIT RC
For a return code of 12 and MACRO RC of 8, this value contains the return code from RACF or the SAF router exit routine. For RACF, this is the RACHECK return code. Otherwise it is set to zero.

Note: For the INIT service call using a program, only the first line will appear, indicating the user does not have READ access to the project definition data set.

Programmer Response: Verify that you specified the correct group and type for the function you are requesting.

If the request was valid, get update authority to the data set identified in the message.

Project Administrator Response: None.

FLM07002 ERROR PERFORMING AN ENQUEUE

CODE: *aaa* **QNAME:** *bbbbbbbb* **RNAME LENGTH:** *ccc* **RNAME:** *ddd(60)*

Explanation: The requested resource was enqueued by another job. The enqueued resource is identified by the RNAME. RNAME is usually a data set. RNAME LENGTH identifies the size of RNAME in bytes because RNAME may contain trailing blanks. QNAME is the name of the queue used for the enqueue operation.

Programmer Response: Try the job again later.

Project Administrator Response: None.

FLM07003 ERROR PURGING MEMBER

CODE: *aaa* **ERROR GROUP:** *bbbbbbbb* **TYPE:** *cccccccc* **MEMBER:** *dddddddd*

Explanation: An error occurred while attempting to delete the specified member.

Possible return codes are:

4 The specified member does not exist.
8 The target data set is enqueued.
12 An I/O error exists in the target data set.
16 SCLM is unable to allocate the data set.
20 An internal error exists.

Programmer Response: Verify that the data set exists and that it is not allocated exclusively to another job. Resubmit the job.

Project Administrator Response: None.

FLM07004 ERROR ALLOCATING A TEMPORARY DATA SET

CODE: *aaaa* **DDNAME:** *bbbbbbbb* **LRECL:** *cccccccc* **RECFM:** *dddddddd* **NUMRECS:** *eeeeee* **DISP:** *ffff* **DSNAME:** *ggg(44)*

Explanation: An error occurred in attempting to allocate a temporary data set. The file number identifies the unallocated data set for the translator being invoked.

Possible return codes are:

8 SVC 99 error.

12 SCLM internal error. Contact SCLM Program Support.

16 Missing or incorrect data set name.

20 Invalid file attribute specified.

24 A member of a PDS was requested but the data set is not partitioned.

28 The requested member could not be found.

32 The requested member was not available.

>64 SVC 99 error, reason code (decimal):

528 Data set cannot be exclusively WRITE allocated.

5896 Data set does not exist.

Programmer Response: Resubmit the job. If the error recurs, contact SCLM Program Support.

Project Administrator Response: None.

FLM07005 ERROR RETRIEVING DIRECTORY INFORMATION FOR TYPE: *aaaaaaaa* **CODE:** *bbb*

Explanation: One or more data sets in the hierarchy are allocated exclusively to another job.

Programmer Response: Verify that data sets are not allocated exclusively to another job.

Project Administrator Response: None.

FLM07006 ERROR ACCESSING MEMBER: *aaaaaaaa* **TYPE:** *bbbbbbbb*, **CODE:** *ccc*

Explanation: One of the following has occurred:

- The specified type does not exist or is not defined.
- One or more data sets in the hierarchy are allocated exclusively to another job.

Possible return codes are:

8 The member is not registered with SCLM or is not allocated.

16 SCLM cannot retrieve the directory for the member.

Programmer Response: Check for the following and resubmit the job:

- Check input parameters and verify that the type exists in the project definition.
- Verify that data sets are not allocated exclusively to another job.

Project Administrator Response: None.

FLM07007 ACCOUNTING INFORMATION IS NOT ACCURATE FOR MEMBER: *aaaaaaaa* **TYPE:** *bbbbbbbb* **ACCOUNTING GROUP:** *cccccccc* **MEMBER GROUP:** *dddddddd*

Explanation: The accounting information for the member does not match the contents of the member. If neither group is a private library, it is possible that the member has been updated outside of SCLM control.

Programmer Response: If the member is editable, register the member with SCLM using the SCLM editor, migration utility, or the SAVE service. If the member is non-editable, delete the member with the SCLM library utility or the DELETE service and regenerate the member with the SCLM build function.

Project Administrator Response: None.

FLM07008 ERROR ACCESSING MEMBER: *aaaaaaaa* **TYPE:** *bbbbbbbb*, **CODE:** *ccc*

Explanation: The specified member could not be allocated or opened.

Programmer Response: Verify that the member exists and that the data set is not allocated exclusively to another job. Resubmit the job.

Project Administrator Response: None.

FLM07009 **ERROR ACCESSING MEMBER:** *aaaaaaaa* **TYPE:** *bbbbbbbb*, **CODE:** *ccc*

Explanation: The message indicates that a possible I/O error occurred in accessing the member.

Programmer Response: Verify that the member exists and that the data set is not allocated exclusively to another job. Resubmit the job.

Project Administrator Response: None.

FLM07010 **ERROR UPDATING DIRECTORY INFORMATION AT GROUP:** *aaaaaaaa* **TYPE:** *bbbbbbbb* **MEMBER:** *cccccccc* **CODE:** *ddd*

Explanation: SCLM could not update the data set directory for this member.

Programmer Response: Reallocate the data set with more directory blocks.

Project Administrator Response: None.

FLM07011 **ERROR ALLOCATING DATA SET FOR TRANSLATOR:** *aaa(16)*

(FILE NUMBER: *bbb*) **CODE:** *ccc*

Explanation: An error occurred in allocating a temporary data set for the specified translator. The file number identifies the unallocated data set for that translator.

Possible return codes are:

8 SVC 99 error.

12 SCLM internal error. Contact SCLM Program Support.

16 Missing or incorrect data set name.

20 Invalid file attribute specified.

24 A member of a PDS was requested but the data set is not partitioned.

28 The requested member could not be found.

32 The requested member was not available.

>64 SVC 99 error, reason code (decimal).

Programmer Response: Resubmit the job. If the error recurs, contact SCLM Program Support.

Project Administrator Response: None.

FLM09002 **THE REPORT WILL APPEAR IN** *aaa(26)*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM09004 **THE MESSAGES WILL APPEAR IN** *aaa(26)*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM09006 **THE LISTING WILL APPEAR IN** *aaa(26)*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM09008 **RETURN CODE** = *aaaaaaaa*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM20001 **IF PARSER LISTINGS WERE CREATED, THEY WILL APPEAR IN DSN:** *aaa(44)*

Explanation: If the dsname value is blank assure that the language macro asks for an error file to be allocated via the Print = I or Print = Y option.

Programmer Response: None.

Project Administrator Response: None.

FLM32101 **MIGRATION UTILITY INITIATED -** *aaaaaaaa* **ON** *bbbbbbbb*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM32201 **UNABLE TO READ DIRECTORY FOR DATA SET**

CODE: *aaa*

Explanation: An error occurred while attempting to read the directory of the data set to be migrated.

Possible return codes are:

16 SCLM is unable to open the data set.

20 A severe error occurred attempting to read the data set directory.

Programmer Response: Verify that the data set directory can be accessed by using the SCLM editor to browse the data set. If you cannot browse the data set, correct the problem and resubmit the job. Possible problems are that the data set is enqueued or the data set does not contain a valid directory.

Project Administrator Response: None.

FLM32303 **NO MEMBERS MATCHING SELECTION CRITERIA NEED MIGRATION**

Explanation: The migration utility did not attempt to migrate any members into SCLM control, because there are no members that are not under SCLM control which match the PROJECT, GROUP, TYPE, and MEMBER parameters. Members are considered under SCLM control if SCLM has accurate accounting information for them.

Programmer Response: Verify that the members to be migrated are not already under SCLM control and that they match the PROJECT, GROUP, TYPE, and MEMBER parameters.

Project Administrator Response: None.

FLM32310 USER DEFINED DDNAME: *aaaaaaaa* **FOR MIGRATION MESSAGES IS NOT ALLOCATED**

Explanation: The ddname specified for the migration messages was not allocated. If the migration function is invoked via the services, the ddname for the migration messages is optional. If not specified, the migration report is defaulted to the terminal. If the ddname is specified it must be allocated.

Programmer Response: Verify that the user-supplied ddname for the migration messages is allocated. Resubmit the job.

Project Administrator Response: None.

FLM32320 USER DEFINED DDNAME: *aaaaaaaa* **FOR MIGRATION LISTING IS NOT ALLOCATED**

Explanation: The ddname specified for the migration listing was not allocated. If the migration function is invoked via the services, the ddname for the migration listing is optional. If not specified, the migration listing is defaulted to the terminal. If a ddname is specified it must be allocated.

Programmer Response: Verify that the user-supplied ddname for migration listing is allocated. Resubmit the job.

Project Administrator Response: None.

FLM32401 MIGRATION UTILITY COMPLETED

Explanation: The migration utility finished processing.

Programmer Response: See the accompanying messages that appear with this message on your screen for additional information regarding the status of this report.

Project Administrator Response: None.

FLM32501 INVOKING MIGRATION UTILITY

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM40501 NO TRANSLATOR INVOKED FOR LANGUAGE: *aaaaaaaa*

Explanation: No translator was invoked for the specified language.

Programmer Response: If a translation was expected, contact the project administrator.

Project Administrator Response: Verify that a translation was expected.

FLM40507 ERROR ALLOCATING DATA SET: *aaa(44)* **FOR TRANSLATOR:** *bbb(16)* **(FILE NUMBER:** *ccc* **CODE:** *ddd*

Explanation: An error occurred in allocating the data set for the translator. The data set is being allocated for the file number.

Possible problems:

- More than one IOTYPE=I may have been specified in the FLMALLOC list for a translator.

Possible return codes are:

8 SVC 99 error.
12 SCLM internal error. Contact SCLM Program Support.
16 Missing or incorrect data set name.
20 Invalid file attribute specified.
24 A member of a PDS was requested but the data set is not partitioned.
28 The requested member could not be found.
32 The requested member was not available.
>64 SVC 99 error, reason code (decimal).

Programmer Response: Resubmit the job. If the error recurs, contact SCLM Program Support.

Project Administrator Response: None.

FLM40510 ERROR ALLOCATING DATA SET: *aaa(44)* **FOR TRANSLATOR:** *bbb(16)* **CODE:** *ccc*

Explanation: An error occurred in allocating the data set for the translator. This data set should contain the translator to be invoked.

Possible return codes are:

8 SVC 99 error.
12 SCLM internal error. Contact SCLM Program Support.
16 Missing or incorrect data set name.
20 Invalid file attribute specified.
24 A member of a PDS was requested but the data set is not partitioned.
28 The requested member could not be found.
32 The requested member was not available.
>64 SVC 99 error, reason code (decimal).

Programmer Response: Resubmit the job. If the error recurs, contact the project administrator.

Project Administrator Response: Verify that the data set in error exists and is specified in the language definition for that translator (DSNAME parameter on the FLMTRNSL macro).

FLM40516 MEMBER: *aaaaaaaa* **TYPE:** *bbbbbbbb* **WAS UPDATED DURING THE BUILD**

Explanation: The SCLM editor updated the specified member during processing of the build. SCLM does not save translator output because it may have been created from the previous version of the member.

Programmer Response: Resubmit the job.

Project Administrator Response: None.

FLM40517 DUPLICATE FLMALLOC KEYREF=*aaaaaaaa* **NOT ALLOWED FOR TRANSLATOR OUTPUT**

Explanation: The translator invoked has two temporary output data sets, allocated with either IOTYPE=O or P, both targeted to the same output member (with the KEYREF parameter). The build processor cannot copy multiple output data sets produced by the translator to a single targeted member.

Programmer Response: See the project administrator.

Project Administrator Response: Verify in the language definition that no two FLMALLOC macro calls with either IOTYPE=O or P have the same KEYREF value.

FLM40519 NUMBER OF ALLOCATED DATA SETS FOR HIERARCHY SEARCH AS SPECIFIED IN DATA SET *ccc* FOR TRANSLATOR *ddd(16)* HAS BEEN EXCEEDED (*aaa* DATA SETS WERE ALLOCATED, MAXIMUM ALLOWED IS *bbb*)

Explanation: The number of data sets allocated for the translator hierarchy search has exceeded the maximum value for the system. This message is preceded with the call name of the translator in question. The ddname allocated for hierarchy search is specified by the FLMALLOC macro with IOTYPE=I.

Programmer Response: For the translator in question, verify that all FLMALLOC macros with IOTYPE=I do not exceed the system limit for allocating data sets to a ddname. This error can be caused by the following:

- Defining too many groups for the project
- Using the extended type option (field EXTEND on the FLMTYPE macro)
- Specifying too many FLMCPYLBs for the ddname.

Project Administrator Response: None.

FLM41002 ERROR OCCURRED DURING INITIALIZATION

Explanation: An error occurred during the initialization phase of the build processor.

Programmer Response: See the message data set for all the messages related to this error.

Project Administrator Response: None.

FLM41005 ERROR ALLOCATING DATA SET: *aaa(44)* **FOR USER EXIT:** *#bb* **CODE:** *ccc*

Explanation: An error occurred in allocating the data set for the user exit. This data set should contain the user exit routine to be invoked.

Possible return codes are:

8 SVC 99 error.
12 SCLM internal error. Contact SCLM Program Support.
16 Missing or incorrect data set name.
20 Invalid file attribute specified.
24 A member of a PDS was requested but the data set is not partitioned.
28 The requested member could not be found.
32 The requested member was not available.
>64 SVC 99 error, reason code (decimal).

Programmer Response: Resubmit the job. If the error recurs, contact the project administrator.

Project Administrator Response: Verify that the data set in error exists and is specified in the control type in the project definition (FLMCNTRL macro) for that user exit.

FLM42000 BUILD PROCESSOR INITIATED - *aaaaaaaa* **ON** *bbbbbbbb*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM42004 INVALID INPUT PARAMETER

GROUP	*aaaaaaaa*
TYPE	*bbbbbbbb*
MEMBER	*cccccccc*
USER ID	*ddddddd*
BUILD MODE	*e*
BUILD SCOPE	*f*
ERROR LISTINGS ONLY	*g*
REPORT REQUEST	*h*
PREFIX USER ID	*iii(17)*

Explanation: You specified an invalid input parameter to the build processor. The values of the parameters are listed. Only the first character is listed for build mode and build scope.

Valid values for build mode are CONDITIONAL, UNCONDITIONAL, REPORT, and FORCED. Valid values for build scope are LIMITED, NORMAL, SUBUNIT, and EXTENDED. Valid values for report request are Y and N.

If the build processor was invoked through the SCLM dialog, SCLM retrieves the user ID and prefix user ID input parameters from the ISPF/PDF shared and profile pools, respectively.

Programmer Response: Verify that all input parameters are specified correctly. Resubmit the job, and if the problem recurs, contact SCLM Program Support.

Project Administrator Response: None.

FLM42100 USER DEFINED DDNAME: *aaaaaaaa* **FOR BUILD MESSAGE NOT ALLOCATED**

Explanation: The ddname specified for the build messages was not allocated. If the build function is called through the SCLM services, the ddname for the build messages is optional. If not specified, the build messages are defaulted to the terminal. If a ddname is specified, it must be allocated.

Programmer Response: Verify that the user-supplied ddname for build messages is allocated. Resubmit the job.

Project Administrator Response: None.

FLM42104 USER DEFINED DDNAME: *aaaaaaaa* **FOR BUILD REPORT NOT ALLOCATED**

Explanation: The ddname specified for the build report was not allocated. If the build function is invoked through the SCLM services, the ddname for the build report is optional. If not specified, the build report is defaulted to the terminal. If a ddname is specified, it must be allocated.

Programmer Response: Verify that the user-supplied ddname for build report is allocated. Resubmit the job.

Project Administrator Response: None.

FLM42106 USER DEFINED DDNAME: *aaaaaaaa* **FOR BUILD LISTING NOT ALLOCATED**

Explanation: The ddname specified for the build listing was not allocated. If the build function is invoked through the SCLM services, the ddname for the build listing is optional. If not specified, the build listing is defaulted to the terminal. If the ddname is specified, it must be allocated.

Programmer Response: Verify that the user-supplied ddname for build listing is allocated. Resubmit the job.

Project Administrator Response: None.

FLM42108 USER DEFINED DDNAME: *aaaaaaaa* **FOR USER EXIT DATA SET NOT ALLOCATED**

Explanation: The ddname specified for the user exit data set was not allocated. If the build function is invoked through the SCLM services, the ddname for the user exit data set must be specified if a user exit routine has been specified. Otherwise the ddname is optional. If not specified, a user exit data set is allocated to NULLFILE. If a ddname is specified, it must be allocated.

Programmer Response: Verify that the user-supplied ddname for user exit file is allocated. Resubmit the job.

Project Administrator Response: None.

FLM43001 ERROR RETRIEVING ACCOUNTING INFORMATION

CODE: *aaa* **GROUP:** *bbbbbbbb* **TYPE:** *cccccccc* **MEMBER:** *dddddddd* **(REFERENCED BY MEMBER:** *eeeeeeee* **TYPE:** *ffffffff***)**

Explanation: No accounting information exists or could be retrieved for the specified member. If the member is an include, then the referencing member is provided. If the member is a compilable member, then the referencing member is the name of the build map.

Possible problems:

- Member was moved to another type after a previous successful build.
- Member referenced in an architecture definition does not exist.
- FLMSYSLB macros have been added or removed from the project definition since the source members being built were last parsed or migrated.

Programmer Response: Verify that the accounting database is correct, or register both the specified member and the referencing source member with SCLM using the SCLM editor, SAVE service, or migration utility. Then resubmit the job.

Project Administrator Response: None.

FLM43007 LANGUAGE SCOPE: *a* **FOR MEMBER:** *bbbbbbbb* **TYPE:** *cccccccc* **CONFLICTS WITH BUILD SCOPE SPECIFIED**

Explanation: The scope specified in the project definition for the specified member is of greater range than the scope specified on the build panel. The first letter of the scope defined in the project definition is listed.

Programmer Response: You can specify the following four scopes (in ascending order): LIMITED, NORMAL, SUBUNIT, and EXTENDED. Verify that the range specified as input to the build processor is of equal or greater range than the scope specified in the project definition for the source member being built.

Project Administrator Response: None.

FLM43008 ERROR PROCESSING DEPENDENCIES FOR MEMBER: *aaaaaaaa* **TYPE:** *bbbbbbbb*

Explanation: Errors occurred while processing the dependencies for the specified member. Other messages preceding this one provide more detail on the exact errors that occurred.

Programmer Response: See the message data set for all the messages related to this error.

Project Administrator Response: None.

FLM43109 NO ACCOUNTING INFORMATION EXISTS FOR COMPILATION UNIT

CU NAME: *aaa(55) bbb(55)* **CU TYPE:** *cccc* **CU QUALIFIER:** *dddddddd* **(ACCOUNTING INFORMATION EXISTS FOR INTERMEDIATE FORM)**

Explanation: Accounting information does not exist for the compilation unit; however, accounting information does exist for the associated intermediate form.

This error can be caused when you delete a source member (using the library utility) but forget to delete the intermediate forms produced by the compiler for those compilation units contained in the deleted source member.

Programmer Response: Delete the intermediate form from all groups in the hierarchy used in the build.

Project Administrator Response: None.

FLM43111 SPECIFICATION MISSING FOR COMPILATION UNIT

CU NAME: *aaa(55) bbb(55)* **CU TYPE:** *cccc* **CU QUALIFIER:** *dddddddd*

Explanation: The specified compilation unit has a dependency on an implicit specification. Implicit specifications are not allowed.

Programmer Response: Create a specification for the compilation unit.

Project Administrator Response: None.

FLM43119 VERIFICATION ERROR OCCURRED FOR COMPILATION UNIT

CU NAME: *aaa(55) bbb(55)* **CU TYPE:** *cccc* **CU QUALIFIER:** *dddddddd*

Explanation: Accounting information for the specified compilation unit does not match accounting information for the member that contains the source for the compilation unit. The member in question is identified in the messages that appear on your screen after this message.

Programmer Response: Register the member with SCLM, using the SCLM editor, migration utility, or SAVE service, and resubmit the job.

Project Administrator Response: None.

FLM43120 ERROR PROCESSING DEPENDENCIES FOR COMPILATION UNIT:

CU NAME: *aaa(55) bbb(55)* **CU TYPE:** *cccc* **CU QUALIFIER:** *dddddddd*

Explanation: Errors occurred while processing the dependencies for the specified compilation unit.

Programmer Response: See the messages that appeared

before this message on your screen for additional information.

Project Administrator Response: None.

FLM44001 **MEMBER:** *aaaaaaaa* **TYPE:** *bbbbbbbb* **REFERENCES MEMBER:** *cccccccc* **TYPE:** *dddddddd* **- LOOP FOUND**

Explanation: SCLM found a loop in the specified reference. A loop is a recursive reference. For example, if member X includes member Y and member Y includes member X, a loop results.

Programmer Response: Verify that the dependency structure of the member being built does not contain any loops. Resubmit the job.

Project Administrator Response: None.

FLM44032 **WARNING, "COMP" KEYWORD NOT SPECIFIED FOR MEMBER:** *aaaaaaaa* **TYPE:** *bbbbbbbb*

Explanation: The message is a warning indicating that you have not specified a COMP keyword for a JOVIAL compool. This missing keyword (COMP) will result in the data dictionary not being updated for the compool.

Programmer Response: Verify that the translator to be invoked for the member contains an FLMALLOC macro with IOTYPE = O and KEYREF = COMP. If the specified member is an architecture member, add a COMP keyword.

Project Administrator Response: None.

FLM44035 **FLMALLOC MACRO WITH KEYREF =** *aaaaaaaa* **DOES NOT EXIST FOR LANGUAGE** *bbbbbbbb*

Explanation: The specified language does not contain an FLMALLOC macro with KEYREF *aaaaaaaa*. An architecture member contains the keyword *aaaaaaaa* and controls invocation of the translators for the language *bbbbbbbb*.

Programmer Response: Verify that an FLMALLOC macro with a KEYREF = *aaaaaaaa* parameter exists for the language; otherwise, remove the keyword from the architecture member.

Project Administrator Response: None.

FLM44037 **ERROR PROCESSING MEMBER:** *aaaaaaaa* **TYPE:** *bbbbbbbb*

Explanation: An error occurred while processing the specified member. The member may either be a source member or an architecture member. Other error messages are generated that describe the exact problem.

Programmer Response: See the message data set for all the messages related to this error.

Project Administrator Response: None.

FLM44039 **MULTIPLE "SINC" KEYWORDS MUST REFERENCE THE SAME TYPE SINCE COMPILATION UNIT DEPENDENCIES ARE PRESENT**

Explanation: Multiple SINC statements with different types were specified in the architecture member in which compilation dependencies existed for the members specified on the SINC statement. SCLM requires that all source members referenced with the SINC keyword reside in the same type if any of the members contain compilation units. The message that appears after this message identifies the architecture member in question.

Programmer Response: If you specify multiple source inputs with the SINC keyword, verify that they reside in the same type.

Project Administrator Response: None.

FLM44101 **ARCHITECTURE MEMBER:** *aaaaaaaa* **TYPE:** *bbbbbbbb* **NOT FOUND WITHIN SCOPE OF ARCHITECTURE DEFINITION BEING BUILT**

Explanation: The specified member is being referenced during the build; however, it was not predefined by build to be within the scope of processing. This error may occur if, for example, during the building of a system, a subsystem of that system is rebuilt by another build or promoted into the hierarchy (perhaps by another user).

The rebuilding of the subsystem may increase the scope of the build for the system. The building of the system may have proceeded too far to identify any more members within the scope.

Programmer Response: Verify that no other builds or promotes are occurring within your hierarchy and resubmit the job.

Project Administrator Response: None.

FLM44201 **ARCHITECTURE MEMBER:** *aaaaaaaa* **TYPE:** *bbbbbbbb* **HAS INVALID SYNTAX**

Explanation: The indicated architecture member is not correct. Additional messages are generated that describe the exact problem with this architecture member.

Programmer Response: See the message data set for all the messages related to this error.

Project Administrator Response: None.

FLM44203 **MEMBER:** *aaaaaaaa* **TYPE:** *bbbbbbbb* **IS INCORRECTLY REFERENCED BY MEMBER:** *cccccccc* **TYPE:** *dddddddd*

Explanation: An incorrect dependency exists when this reference occurs. This error occurs when an LEC architecture member references a member that does not produce either an object module or a load module when built. Processing of member *cccccccc* cannot continue.

Programmer Response: Verify that member *aaaaaaaa* in type *bbbbbbbb* produces an object or load module. An object module must be identified by the OBJ keyword. A load module must be identified by the LOAD keyword.

Project Administrator Response: None.

FLM44230 **ERROR RETRIEVING ACCOUNTING INFORMATION**

CODE: *aaa* **GROUP:** *bbbbbbbb* **TYPE:** *ccccccc* **LOAD MODULE:** *ddddddd*

Explanation: No accounting information exists or could be retrieved for the specified member within the hierarchy beginning in the specified group. The member in question is a load module being referenced by the LINK keyword.

Programmer Response: Possible return codes are:

8 The member's accounting information was not found. Build the architecture member that creates the load module.

20 A severe I/O error occurred. Contact the project administrator.

Project Administrator Response: Run IDCAMS against the accounting data set to determine the problem.

FLM44231 INVALID REFERENCE TO LOAD MODULE: *aaaaaaaa* **TYPE:** *bbbbbbbb*

Explanation: The load module referenced with the LINK keyword is a member that can be edited.

Programmer Response: For the architecture member that contains the error, verify that the LINK keyword specifies a load module and not the architecture member that creates the link load module.

Project Administrator Response: None.

FLM44280 SYNTAX ERROR IN ARCHITECTURE MEMBER: *aaaaaaaa* **TYPE:** *bbbbbbbb* **CODE:** *ccc*

Explanation: Possible return codes are:

4 The specified architecture member contains keywords that identify it as both an LEC and a CC or a generic architecture member.

8 Depending on the architecture member that you were processing, a SINC keyword was not specified.

12 The architecture member was identified as a CC or generic architecture member (contains the SINC keyword). However, the architecture member contains a keyword used only for HL and LEC architecture members (INCL, INCLD, and LINK keywords).

16 The architecture member was identified as an LEC (contains the ALIAS, LOAD, LKED, LMAP, or any combination of these keywords). However, the architecture member does not contain a keyword to identify the inputs (INCL, INCLD, and LINK keywords).

20 The architecture member was identified as an HL. However, the architecture member contains either a PARM, PARMx, or CMD keyword. These keywords are not allowed for an HL architecture member.

24 The architecture member was identified as an LEC but is missing the keyword LOAD.

Programmer Response: If the return code is:

4 If the architecture member is to be used to create a load module, use only the LOAD keyword in it. Otherwise, use only OBJ keywords for architecture members intended to create object modules and OUTx keywords for architecture members intended to create other generic output. Check the architecture member for a SINC keyword and resubmit.

8 Check the architecture member for a SINC keyword and resubmit.

12 See "Statement Uses" on page 27 for valid keywords in architecture members.

16 See "Statement Uses" on page 27 for valid keywords in architecture members.

20 See "Statement Uses" on page 27 for valid keywords in architecture members.

24 SCLM identified the architecture member as an LEC, but it does not have a LOAD keyword.

Project Administrator Response: None.

FLM44304 COMPOOL DEPENDENCY TYPE WAS NOT SPECIFIED FOR MEMBER: *aaaaaaaa* **TYPE:** *bbbbbbbb*

Explanation: Build processor could not find the CREF type for the indicated member.

Programmer Response: If the member is an architecture member then verify that a CREF keyword exists. If the member is source, verify that the language definition (FLMLANGL macro) specifies the DFLTCRF parameter.

Project Administrator Response: None.

FLM44306 ERROR RETRIEVING ACCOUNTING INFORMATION

CODE: *aaa* **GROUP:** *bbbbbbbb* **TYPE:** *cccccccc* **COMPOOL:** *dddddddd*

Explanation: No accounting information exists or could be retrieved for the specified member within the hierarchy beginning in group *bbbbbbbb*. The member in question is a compool reference where type is defined either by the CREF keyword in an architecture member or the DFLTCRF parameter (FLMLANGL macro) in the language definition.

Programmer Response: Possible return codes are:

8 The member's accounting information was not found. Build the member that creates the specified compool.

20 A severe I/O error occurred. Contact the project administrator.

Project Administrator Response: Run IDCAMS against the accounting data set to determine the problem.

FLM44307 ERROR REFERENCING COMPOOL: *aaaaaaaa* **TYPE:** *bbbbbbbb*

Explanation: The referenced compool is a member that can be edited. It must be a member that was created by the build function.

Programmer Response: Delete the compool from the hierarchy. Rebuild the member that created the compool member. Resubmit the job.

Project Administrator Response: None.

FLM44309 MEMBER: *aaaaaaaa* **TYPE:** *bbbbbbbb* **WAS FOUND AT GROUP:** *cccccccc* **BUT IS BEING CROSS REFERENCED AT GROUP:** *dddddddd*

Explanation: A reference was made to a compilation unit contained in member *aaaaaaaa* in type *bbbbbbbb* at group *dddddddd*. However, a more current version of the member exists at group *cccccccc*. The member at group *cccccccc* does not contain the compilation unit nor does any other member in the hierarchy below group *dddddddd*. This problem can occur when the language of member *aaaaaaaa* is changed to one that has no compilation units or uses a different CUQUAL. Use the sublibrary utility to purge the intermediate form. Check error messages for additional information. If the intermediate form is deleted outside of SCLM control, use the sublibrary utility to delete accounting information for the intermediate form.

Programmer Response: Change the language of member *aaaaaaaa* if appropriate. Add a new member with the compilation units contained in member *aaaaaaaa* at group *ddddddd*. Remove the references to the compilation units in member *aaaaaaaa* at group *ddddddd*.

Project Administrator Response: None.

FLM44311 ERROR PROCESSING COMPILATION UNITS FOR MEMBER: *aaaaaaaa* **TYPE:** *bbbbbbbb*

Explanation: An error occurred during processing of the compilation units for the specified member. Other messages are generated that describe the exact errors that occurred.

Programmer Response: See the message data set for all the messages related to this error.

Project Administrator Response: None.

FLM44315 INVOKE PURGE ROUTINE FOR INTERMEDIATE FORM

CU NAME: *aaa(55) bbb(55)* **CU TYPE:** *cccc*
CU QUALIFIER: *ddddddd*
OLD RECORD - TYPE: *eeeeeeee*
MEMBER: *fffffff* **LANG:** *gggggggg*
NEW RECORD - TYPE: *hhhhhhhh*
MEMBER: *iiiiiii* **LANG:** *jjjjjjj*

Explanation: The intermediate form is no longer valid. It was previously created by another source member, or the language of the source member was changed. A routine is being invoked to purge the intermediate form. This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM44319 UNABLE TO PURGE INTERMEDIATE FORM

CODE: *aaaa*

Explanation: The purge of the intermediate form was not successful.

Programmer Response: Resubmit the job. If the error recurs, contact the project administrator.

Project Administrator Response: Verify that the sublibrary containing the intermediate form is not corrupted.

FLM44410 MEMBER: *aaaaaaaa* **TYPE:** *bbbbbbbb* **HAS DEPENDENCY ON INLINE/GENERIC MEMBER:** *cccccccc* **TYPE:** *ddddddd* **-- LOOP NOT ALLOWED.**
TRACE BACK OF DEPENDENCIES:
MEMBER: *cccccccc...aaaaaaaa*
TYPE:*bbbbbbbb...ddddddd*

Explanation: A dependency loop involving a generic or inline member was detected when member *cccccccc* referenced member *aaaaaaaa*. SCLM does not allow this structure. See "Generic/INLINE Recursive Dependencies" on page 281.

Programmer Response: Edit the necessary source members to remove the dependency loop back to the inline/generic member. See "Ada Language Restrictions" on page 280 for alternate responses.

Project Administrator Response: None.

FLM44500 > > > > > INVOKE TRANSLATOR(S) FOR TYPE: *aaaaaaaa* **MEMBER:** *bbbbbbbb*

Explanation: Translators are being invoked for the specified member. This member may be either a source member or an architecture member. This message is provided for information only.

Programmer Response: None

Project Administrator Response: None.

FLM44501 REPORT: INVOKE TRANSLATOR(S) FOR TYPE: *aaaaaaaa* **MEMBER:** *bbbbbbbb*

Explanation: The translators would be invoked for the specified member if the build mode were not report-only.

Programmer Response: None

Project Administrator Response: None.

FLM44504 ERROR PRINTING TO BUILD LISTING DATA SET FOR FILE NUMBER *aa* **IN TRANSLATOR:** *bbb(16)* **CODE:** *ccc*

Explanation: An error occurred during the printing of a translator data set to the build listing data set. The file number identifies the relative position of the FLMALLOC macro used to allocate the data set for that translator.

Note: Only data sets allocated with IOTYPE=O, W, and S can be printed to the build listing data set.

Programmer Response: Contact the project administrator.

Project Administrator Response: If the return code is:

12 The ddnames are not allocated properly. Verify that the build listing data set and the translator data set are allocated. The problem may be due to conflicting attributes between the two data sets.

16 The build listing data set is full. Reallocate the data set with more storage.

20 Data access failed or SCLM did not find the input member. Verify that the type of access (RACF) is allowed. Verify that the translator data set still exists after all translator steps in the language definition have been completed. A user-created translator may have purposely deallocated the data set.

An ABEND may occur during the printing if the translator data set is allocated with PRINT=Y on the FLMALLOC macro and the data set is never opened by the translator. In such cases, specify PRINT=I on the FLMALLOC macro. This attribute forces the data set to be opened before the translator is invoked, and the data set will be targeted for printing to the build listing data set.

FLM44506 ERROR SAVING FILE NUMBER *aa* **FOR TRANSLATOR:** *bbb(16)* **TO MEMBER:** *cccccccc* **TYPE:** *ddddddd*, **CODE:** *eeeeeeee*

Explanation: An error occurred during the copying of a translator data set to the member. The file number identifies the relative position of the FLMALLOC macro used to allocate the data set for that translator (the data set is a sequential data set allocated with IOTYPE=O).

Programmer Response: Contact the project administrator.

Project Administrator Response: Possible return codes are:

12 The ddnames are not allocated properly. Verify that the type is allocated properly. The problem may be due to conflicting attributes between the translator data set and the type.

16 The target type may be full. Compress the type or reallocate with more space or directory.

20 Data access failed or SCLM did not find the input member. Verify that the type of access (RACF) is allowed. Verify that the type is not allocated exclusively to another job.

28 The member entry could not be created because the input member is an alias or has TTR notes. Verify that the member is not an alias or does not have TTR notes.

FLM44507 ERROR SAVING FILE NUMBER *aa* **FOR TRANSLATOR:** *bbb(16)* **TO MEMBER:** *ccccccc* **TYPE:** *ddddddd*, **CODE:** *eeeeeeee*

Explanation: An error occurred while copying a translator data set to the member. The file number identifies the relative position of the FLMALLOC macro used to allocate the data set for that translator. The data set is a PDS allocated with IOTYPE=P.

Programmer Response: Possible return codes are:

4 Copy error. The translator data set may be empty. The member created in the PDS may not match the name specified in the architecture member.

8 Enqueue error. Resubmit the job, and if the error recurs, verify that the data set is not allocated exclusively to another job.

12 SCLM is unable to allocate the target member. Verify that the type of access (RACF) is allowed. Verify that the type is not allocated exclusively to another job.

16 Error allocating temporary data sets for IEBCOPY. Resubmit the job.

20 Error opening target output data set. Resubmit the job.

>20 IEBCOPY completion code in decimal.

14905344 (E37)
The directory is full; the maximum of 16 extents is exceeded, or the volume/VTOC that the target data set resides on is full and no secondary volumes are available.

Project Administrator Response: None.

FLM44513 TRANSLATOR ERROR FOR MEMBER: *aaaaaaaa* **TYPE:** *bbbbbbbb*

Explanation: A translator error occurred for the specified member. The return code from the translator was not considered acceptable. The acceptable return codes are specified on the FLMLANGL macro with the GOODRC parameter.

Programmer Response: Check listing for translator errors.

Project Administrator Response: None.

FLM44514 TARGET OUTPUT MEMBER: *aaaaaaaa* **TYPE:** *bbbbbbbb* **IS EDITABLE**

Explanation: The build processor cannot copy the translator output data set. The specified member was created by the SCLM editor or registered with the migration utility or SAVE service. The build processor only updates members that were created through the build process (non-editables).

Programmer Response: If the specified member is no longer to be used as an editable component in the system, delete it from the hierarchy. Otherwise, specify a new target member. Resubmit the job.

Project Administrator Response: None.

FLM46000 BUILD PROCESSOR COMPLETED - *aaaaaaaa* **ON** *bbbbbbbb*

Explanation: The build processor completed.

Programmer Response: See the message data set for all the messages regarding the outcome of this build.

Project Administrator Response: None.

FLM49000 INVOKING BUILD PROCESSOR

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM51000 PROMOTE PROCESSOR INITIATED - *aaaaaaaa* **ON** *bbbbbbbb*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM51001 BLANK USER ID IS SPECIFIED AS AN INPUT TO THE PROMOTE PROCESSOR

Explanation: A blank user ID was specified as an input parameter. If the processor was invoked through the SCLM Promote panel, SCLM retrieves the user ID from the ISPF/PDF variable pool.

Programmer Response: Verify that the user ID specified in the input parameter is correct and non-blank. See "PROMOTE—Promote a Member from One Library to Another" on page 145 for more information about the promote input parameters.

Project Administrator Response: None.

FLM51002 INVALID SCOPE SPECIFIED

Explanation: The promote scope specified is invalid. Valid promote scopes are NORMAL, SUBUNIT, and EXTENDED.

Programmer Response: Verify that the input parameters specified for the promote processor are correct.

Project Administrator Response: None.

FLM51003 INVALID PROMOTE MODE SPECIFIED

Explanation: The promote mode specified is invalid. Valid promote modes are CONDITIONAL, UNCONDITIONAL, and REPORT.

Programmer Response: Verify that the input parameters specified for the promote processor are correct.

Project Administrator Response: None.

FLM51004 GROUP: *aaaaaaaa* IS TOP GROUP - PROMOTE BYPASSED

Explanation: The group specified does not promote to another group for this project definition. The promote report is created as if this were a report-only promote.

Programmer Response: Verify that the group specified as an input parameter to the promote processor is the group containing the data to be promoted. Also verify that you specified the correct project definition as an input to the promote processor.

Project Administrator Response: None.

FLM51006 SPECIFIED GROUP: *aaaaaaaa* IS A PRIMARY NON-KEY GROUP

Explanation: The specified group is a primary non-key group. Promoting from a primary non-key group is not allowed.

Programmer Response: Verify that the group and project definition specified as inputs to the promote processor are correct. See "Primary Non-Key Group Testing Techniques" on page 258 for more information about primary non-key groups.

Project Administrator Response: None.

FLM51008 USER DEFINED DDNAME: *aaaaaaaa* FOR PROMOTE MESSAGES NOT ALLOCATED

Explanation: The ddname specified for the promote messages was not allocated. If the promote function is invoked through the SCLM services, the ddname for the promote messages is optional. If the ddname is not specified, the promote messages are defaulted to the terminal. If the ddname is specified, it must be allocated.

Programmer Response: Verify that the user-supplied ddname for promote messages is allocated. Resubmit the job.

Project Administrator Response: None.

FLM51009 USER DEFINED DDNAME: *aaaaaaaa* FOR PROMOTE REPORT NOT ALLOCATED

Explanation: The ddname specified for the report was not allocated. If the promote function is invoked through SCLM services, the ddname for the promote report is optional. If the ddname is not specified, the promote report is defaulted to the terminal. If the ddname is specified, it must be allocated.

Programmer Response: Verify that the user-supplied ddname for promote report is allocated. Resubmit the job.

Project Administrator Response: None.

FLM51010 USER DEFINED DDNAME: *aaaaaaaa* FOR COPY ERROR MESSAGES NOT ALLOCATED

Explanation: The ddname specified for promote copy error messages was not allocated. If the promote function is invoked through SCLM services, the ddname for the copy error messages is optional. If the ddname is not specified, the copy error messages are defaulted to the terminal. If the ddname is specified, it must be allocated.

Programmer Response: Verify that the user-supplied ddname for copy error messages is allocated. Resubmit the job.

Project Administrator Response: None.

FLM51011 USER DEFINED DDNAME: *aaaaaaaa* FOR USER EXIT FILE NOT ALLOCATED

Explanation: The ddname specified for the user exit data set was not allocated. If the promote function is invoked through the SCLM services, the ddname for the user exit data set is optional. If not specified, a user exit data set is allocated to NULLFILE. If the ddname is specified, it must be allocated.

Programmer Response: Verify that the user-supplied ddname for user exit data set is allocated. Resubmit the job.

Project Administrator Response: None.

FLM51103 NO KEY GROUP EXISTS BELOW GROUP: *aaaaaaaa*

Explanation: No key group exists below the specified group, or the specified group is not defined to SCLM.

Programmer Response: Contact the project administrator.

Project Administrator Response: This condition violates the guideline for a project hierarchy. The lowest groups of the hierarchy must be key groups. Modify the project definition to make the lowest group key and resubmit the job.

FLM52000 INITIATING VERIFICATION PHASE - *aaaaaaaa* ON *bbbbbbbb*

Explanation: Indicates that the promote verification phase has been initiated. In this phase, SCLM verifies all members within the scope of the architecture definition. All members must be up to date (for example, source matches object) and must have correct accounting information.

Programmer Response: None.

Project Administrator Response: None.

FLM52103 ERROR RETRIEVING BUILD MAP INFORMATION

CODE: *aaa* TYPE: *bbbbbbbb*
MEMBER: *cccccccc* REFERENCED BY BUILD MAP AT TYPE: *dddddddd*
MEMBER: *eeeeeeee*

Explanation: SCLM could not retrieve build map information for the specified member.

Programmer Response: Possible return codes are:

8 The specified build map information does not exist. Build the architecture member used as input for this promotion again. Invoke the promote function again, and resubmit the job.

12 The format of the data retrieved was incorrect. Delete the build map and build again to regenerate it.

16 An invalid group was found in the project definition. Contact the project administrator.

20 A severe I/O error occurred. Contact the project administrator.

Project Administrator Response: If the return code is:

16 Reassemble the project definition. Verify that no errors occurred. Relink the project definition. For more information, see "Step 10: Assemble and Link Project Definition" on page 206.

20 A VSAM error occurred. Run IDCAMS against the accounting data set to determine the problem. See "Defining Accounting Data Set (SCLM Internal Data)" on page 194.

FLM52105 **VERSION MISMATCH FOR LANGUAGE:** *aaaaaaaa* **BUILD MAP MEMBER:** *bbbbbbbb* **IN TYPE:** *cccccccc* **LANGUAGE VERSION IN BUILD MAP:** *dddddddd* **LANGUAGE DEFINITION VERSION:** *eeeeeeee*

Explanation: Since the last build, a new version of the translator for the language was installed.

Programmer Response: Rebuild the specified member using the current project definition.

Project Administrator Response: None.

FLM52901 **SCOPE:** *a* **SPECIFIED AS INPUT IS INCOMPATIBLE WITH SCOPE:** *b* **FOR LANGUAGE:** *cccccccc* **OF MEMBER:** *dddddddd* **IN TYPE:** *eeeeeeee*

Explanation: Scope *a* requested for this promote has a smaller range than the scope *b* specified in the project definition for the member's language. Promote accepts three values for scope: NORMAL, SUBUNIT, and EXTENDED. NORMAL has the smallest range; EXTENDED has the greatest range.

Programmer Response: Specify an equal or a larger range scope than the scope of the member's language being promoted. If a non-Ada source is being promoted, NORMAL is usually sufficient for the promote scope. Otherwise, EXTENDED scope is always compatible with the languages. Verify that the architecture definition being promoted has been built with the scope used as input to the promote function.

Project Administrator Response: None.

FLM52902 **WARNING:**
THE FOLLOWING DOWNWARD DEPENDENCY COMPILATION UNIT NOT FOUND

CU NAME: *aaa(55) bbb(55)* **CU TYPE:** *cccc*
CU QUALIFIER: *dddddddd*
OF MEMBER: *eeeeeeee* **TYPE:** *ffffffff*

Explanation: The downward dependency of the member was not found in the hierarchy. This message occurs when an Ada package specification is being promoted, but the body of the compilation unit cannot be found within the hierarchy.

Programmer Response: If the compilation unit identified is a specification that contains only Ada-type declarations or a package body is not required for other reasons, no action is necessary. If a package body was expected for the compilation unit, define the body, rebuild, and promote it.

Project Administrator Response: None.

FLM52904 **ARCHITECTURE MEMBER:** *aaaaaaaa* **IN TYPE:** *bbbbbbbb* **IS NOT CURRENT**
THE SOURCE MEMBER: *cccccccc* **IN TYPE:** *dddddddd* **WAS COMPILED WITHOUT THE DEPENDENT COMPILATION UNIT**

CU NAME: *eee(55) fff(55)* **CU TYPE:** *gggg*
CU QUALIFIER: *hhhhhhhh*

Explanation: The compilation unit identified in this message was added since the last build of the source member.

Programmer Response: Rebuild the architecture definition using the specified scope. Resubmit the job.

Project Administrator Response: None.

FLM52905 **ARCHITECTURE MEMBER:** *aaaaaaaa* **IN TYPE:** *bbbbbbbb* **IS NOT CURRENT**
THE DOWNWARD DEPENDENCY COMPILATION UNIT

CU NAME: *ccc(55) ddd(55)* **CU TYPE:** *eeee*
CU QUALIFIER: *ffffffff*
OF MEMBER: *gggggggg* **TYPE:** *hhhhhhhh*
HAS NOT BEEN BUILT.

Explanation: The specified compilation unit has never been built. This error could occur for one of the following reasons:

- The specified compilation unit was introduced to the product after the architecture member was built.
- The specified member has never been built in EXTENDED scope.

Programmer Response: Rebuild the specified architecture definition in EXTENDED scope, and resubmit the job.

Project Administrator Response: None.

FLM52910 **ARCHITECTURE MEMBER:** *aaaaaaaa* **IN TYPE:** *bbbbbbbb* **IS NOT CURRENT**

Explanation: The architecture member is an architecture member of an Adabind member. One or more compilation units referenced for the Adabind member has been rebuilt since the last time the Adabind member was built.

Programmer Response: Rebuild the architecture member and resubmit the job.

Project Administrator Response: None.

FLM52911 **ARCHITECTURE MEMBER:** *aaaaaaaa* **IN TYPE:** *bbbbbbbb* **DEPENDS ON ARCHITECTURE MEMBER:** *cccccccc* **TYPE:** *dddddddd* **WHICH HAS REBUILT GENERIC/INLINE SUBUNITS**

Explanation: One or more generic or inline compilation units of the specified architecture member has been rebuilt since the last time the architecture member was built.

Programmer Response: Rebuild the architecture member being promoted and resubmit the job.

Project Administrator Response: None.

FLM53005 **ARCHITECTURE MEMBER:** *aaaaaaaa* **IN TYPE:** *bbbbbbbb* **IS NOT CURRENT DATE/TIME MISMATCH ON MEMBER:** *cccccccc* **IN TYPE:** *dddddddd* **BUILD MAP ENTRY DATE/TIME:** *eeeeeeee ffffffff* **ACCOUNTING DATE/TIME:** *gggggggg hhhhhhhh*

Explanation: A change has occurred since the last build of the architecture member. The output of the build does not match the input (for example, source does not match object). Build output for the architecture member was based on a version of the source member. The specified source member has since been updated but the updates have not been built.

Programmer Response: Rebuild the architecture member being promoted and resubmit the job.

Project Administrator Response: None.

FLM53106 **PREDECESSOR VERIFICATION FAILED**

INPUT GROUP : *aaaaaaaa* **TYPE:** *bbbbbbbb*
MEMBER: *cccccccc*
ERROR GROUP1: *dddddddd*
DATE: *eeeeeeee* **TIME:** *ffffffff*
ERROR GROUP2: *gggggggg*
DATE: *hhhhhhhh* **TIME:** *iiiiiiii*

Explanation: The version of the member in *dddddddd* was not based on the member in *gggggggg*. This error usually means that a version of the member between the two groups has been deleted.

The predecessor date and time fields in the accounting information for the member in *dddddddd* should contain the last modified date and time fields for the next occurrence of the member within the hierarchy.

The promote processor, in CONDITIONAL mode, prevents the member in *gggggggg* from being replaced.

Programmer Response: Verify that the member contains all of the required changes present in the member in *gggggggg*. If it does, and no other promote verification errors are present, promote again in UNCONDITIONAL mode.

If other promote verification errors are present, either correct the errors or use an architecture member that controls as few members as possible.

Project Administrator Response: None.

FLM53108 **MEMBER:** *aaaaaaaa* **TYPE:** *bbbbbbbb* **AT GROUP:** *cccccccc* **IS NOT ELIGIBLE FOR PROMOTION**

Explanation: One or more of the accounting information fields for the member has an invalid value, which prevents SCLM from promoting the member. The fields are:

- AUTHORIZATION CODE CHANGE
- ACCESS KEY
- ACCOUNTING RECORD TYPE.

If the AUTHORIZATION CODE CHANGE field is not blank, an attempt was made to change the authorization code of the member, which did not complete successfully.

If the ACCESS KEY field is not blank, the member has been reserved for use outside the project hierarchy or blocked from promotion.

If the ACCOUNTING RECORD TYPE is initial or lockout, a lock has been placed on the member but changes to the member have not been registered with SCLM. The source for the member either does not exist or does not match the accounting information.

Programmer Response: Use the SCLM library utility to review the contents of the specified fields. If the AUTHORIZATION CODE CHANGE field is not blank, verify that the authorization code for the member is correct. If it is, use the update capability of the utility to reset the field. If the field should be changed, use the utility to complete the change in progress or assign a new authcode.

If the ACCESS KEY is not blank, refer to local software configuration management procedures to determine the cause of action based on the values of the access key. If the access key is eligible for removal, use the UNLOCK service to reset the access key to blanks.

If the ACCOUNTING RECORD TYPE is initial or lockout and the member is not present in the group you are promoting from, delete the accounting information using the library utility (or use an equivalent function such as the UNLOCK service).

If the member exists, use the SCLM editor or SAVE service to create correct accounting information. Rebuild the architecture member being promoted after the accounting information has been either deleted or updated.

Project Administrator Response: None.

FLM53109 **WARNING, PREDECESSOR VERIFICATION FAILED**

INPUT GROUP : *aaaaaaaa* **TYPE:** *bbbbbbbb*
MEMBER: *cccccccc*
ERROR GROUP1: *dddddddd*
DATE: *eeeeeeee* **TIME:** *ffffffff*
ERROR GROUP2: *gggggggg*
DATE: *hhhhhhhh* **TIME:** *iiiiiiii*

Explanation: The version of the member in *dddddddd* was not based on the member in *gggggggg*. This error usually means that a version of the member between the two groups has been deleted.

The predecessor date and time fields in the accounting information for the member in *dddddddd* should contain the last modified date and time fields for the next occurrence of the member within the hierarchy.

This message is a warning. However, the promote processor, in CONDITIONAL mode, prevents the member from replacing the member in *gggggggg*.

Programmer Response: For this promote, no action is required. An attempt to promote member *cccccccc* to group *gggggggg* will fail in CONDITIONAL mode.

Project Administrator Response: None.

FLM53901 **ERROR RETRIEVING ACCOUNTING INFORMATION FOR INTERMEDIATE FORM OF: CU NAME:** *aaa(55) bbb(55)*
CU TYPE: *cccc* **CU QUALIFIER:** *dddddddd*
CODE: *eee* **GROUP:** *ffffffff*

Explanation: An error occurred while attempting to retrieve accounting information for the specified intermediate form. The error code associated with the error message provides specifics regarding the nature of the error.

Programmer Response: Possible return codes are:

8 The accounting information for the intermediate form of the compilation unit was not found in the specified group in the hierarchical view. The compiled intermediate form may be missing or out of date. You need to build the member containing the compilation unit.

12 SCLM internal error. Contact the project administrator.

16 SCLM found an invalid group in the project definition. Contact the project administrator.

20 An I/O error occurred while retrieving the accounting information for the intermediate form of the compilation unit. Submit the job again. If the error recurs, contact the project administrator.

24 The cross-reference data set was not defined in the project definition. Contact the project administrator.

Project Administrator Response: If the return code is:

12 Contact SCLM Program Support.

16 Reassemble the project definition. Verify that no errors occurred. Link the project definition again. For more information, see "Step 10: Assemble and Link Project Definition" on page 206.

20 A VSAM error occurred. Run IDCAMS against the cross-reference data set to determine the problem.

24 Identify the cross-reference data set on the FLMCNTRL macro of the project definition. For more information, see "FLMCNTRL Macro" on page 218.

FLM53902 **ARCHITECTURE MEMBER:** *aaaaaaaa* **IN TYPE:** *bbbbbbbb* **IS NOT CURRENT VERIFICATION ERROR FOR COMPILATION UNIT**

CU NAME: *ccc(55) ddd(55)* **CU TYPE:** *eeee* **CU QUALIFIER:** *ffffffff* **BUILD MAP DATE/TIME:** *gggggggg hhhhhhhh* **ACCOUNTING DATE/TIME:** *iiiiiiii jjjjjjjj*

Explanation: A change has occurred since the last build of the architecture member being promoted. The output of the build does not match the input. Build output for the specified compilation unit was based on the build map date and time indicated. The specified compilation unit has since been updated but the updates have not been built.

Programmer Response: Rebuild the architecture member being promoted and resubmit the job.

Project Administrator Response: None.

FLM53903 **WARNING, INTERMEDIATE FORM AND ACCOUNTING INFORMATION FOR THE FOLLOWING COMPILATION UNIT WILL BE PURGED FROM GROUP:** *aaaaaaaa* **CU NAME:** *bbb(55) ccc(55)* **CU TYPE:** *dddd* **CU QUALIFIER:** *eeeeeeee* **FROM-GROUP MEMBER:** *ffffffff* **TYPE:** *gggggggg* **LANGUAGE:** *hhhhhhhh* **ABOVE-GROUP MEMBER:** *iiiiiiii* **TYPE:** *jjjjjjjj* **LANGUAGE:** *kkkkkkkk*

Explanation: The source for the compilation unit identified in the messages was moved to a different member. This move would cause the intermediate form of the compilation unit to exist in more than one sublibrary in the specified group unless the intermediate form is purged. SCLM does not allow multiple copies of a member's compilation unit to exist in one group of the hierarchy; therefore, the old compilation unit is purged.

Programmer Response: No action is necessary unless the promote fails to copy the compilation unit identified. If the copy failed, the group will not contain a copy of the compilation unit until the promote competes successfully.

Project Administrator Response: None.

FLM53905 **ERROR RETRIEVING ACCOUNTING INFORMATION FOR INTERMEDIATE FORM OF: CU NAME:** *aaa(55) bbb(55)* **CU TYPE:** *cccc* **CU QUALIFIER:** *dddddddd* **CODE:** *eee* **GROUP:** *ffffffff*

Explanation: An error occurred while attempting to retrieve accounting information for the specified intermediate form.

Programmer Response: Possible return codes are:

8 The accounting information for the intermediate form of the compilation unit was not found at the specified group. This error indicates that the compiled intermediate form is missing or out of date. You need to build the member containing the compilation unit.

12 SCLM internal error. Contact the project administrator.

16 An invalid group was found in the project definition. Contact the project administrator.

20 An I/O error occurred retrieving the accounting information for the intermediate form of the compilation unit. Resubmit the job, and if the error recurs, contact the project administrator.

24 The cross-reference data set was not defined in the project definition. Contact the project administrator.

Project Administrator Response: If the return code is:

12 Contact SCLM Program Support.

16 Reassemble the project definition. Verify that no errors occurred. Relink the project definition. For more information, see "Step 10: Assemble and Link Project Definition" on page 206.

20 A VSAM error occurred. Run IDCAMS against the cross-reference data set to determine the problem.

24 Identify the cross-reference data set on the FLMCNTRL macro of the project definition. For more information, see "FLMCNTRL Macro" on page 218.

FLM55000 **INITIATING COPY PHASE -** *aaaaaaaa* **ON** *bbbbbbbb*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM55004 COPY FAILED FOR TYPE: *aaaaaaaa* **CODE:** *bbb*

Explanation: One of the following has occurred:

- The target type does not exist or is not allocated with the same attributes as the type specified.
- The target type may be allocated exclusively to another job.
- One or more members were deleted between the promote verification and copy phases.

Programmer Response: Check the following:

- Verify that the target data set exists and is allocated with the same attributes as specified type.
- Verify that the target data set is not allocated exclusively to another job.
- If MVS messages were produced, review those messages for more information.

Project Administrator Response: None.

FLM55104 COPY FAILED FOR TYPE: *aaaaaaaa* **ABEND CODE:** *bbbb*

Explanation: Promote was unable to update the data set associated with the specified type because of an ABEND during copy.

Common ABEND codes and their meanings are:

D37
Primary space is full and secondary space is not requested in the target data set.

B37 or E37
The directory is full; the maximum of 16 extents were exceeded; or the volume/VTOC that the target data set resides on is full and secondary volumes are not available.

Programmer Response: Check for MVS system error messages for detailed information. Resubmit the job after you check the following:

1. Compress target data set or reallocate with more space or directory blocks.
2. Verify that the volume and VTOC of the target data set are not full. Move the data set if they are.

Project Administrator Response: None.

FLM55201 ERROR OCCURRED DELETING ACCOUNTING INFORMATION FOR INTERMEDIATE FORM OF DISCREPANCY ITEMS

Explanation: An error occurred while attempting to purge an intermediate form or intermediate accounting record in the "from" group. The intermediate form's type or member name at the "from" group does not match the "to" group. Check that the source for a compilation unit was not moved to a different member.

Programmer Response: See the message data set for all the messages relating to this error.

Project Administrator Response: None.

FLM55904 COPY OF INTERMEDIATE FORM FAILED FOR LANGUAGE: *aaaaaaaa*

Explanation: An error occurred while copying an intermediate form of a compilation unit in the specified language.

Programmer Response: See the message data set for all the messages relating to this error.

Project Administrator Response: None.

FLM55905 ERROR PURGING CROSS-REFERENCE INFORMATION FOR EXTRA COMPILATION UNIT

CU NAME: *aaa(55) bbb(55)* **CU TYPE:** *cccc*
CU QUALIFIER: *dddddddd* **CODE:** *eee*
GROUP: *ffffffff* **TYPE:** *gggggggg*
MEMBER: *hhhhhhhh*

Explanation: The promote processor deletes all the cross-reference information for extra compilation units in the specified group before it copies new text and accounting records of all the members. An extra compilation unit is a compilation unit that exists in the "to" group but does not exist in the "from" group for a member existing in both groups. This situation occurs when you modify a member with an extra compilation in a private library and then delete the extra compilation unit of the member from the private library.

While deleting the cross-reference information from the group for the compilation unit specified, an error occurred and SCLM issued a return code.

Programmer Response: Possible return codes are:

8 A severe I/O error occurred. Contact the project administrator.

16 The cross-reference data set is enqueued. Try the job again later.

24 The cross-reference data set was not defined in the project definition. Contact the project administrator.

Project Administrator Response: If the return code is:

8 A VSAM error occurred. Run IDCAMS against the cross-reference data set to determine the problem.

24 Identify the cross-reference data set on the FLMCNTRL macro of the project definition. For more information, see "FLMCNTRL Macro" on page 218.

FLM57000 INITIATING PURGE PHASE - *aaaaaaaa* **ON** *bbbbbbbb*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM57001 INITIATING PURGE FROM GROUP: *aaaaaaaa*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM57007 **PURGE FAILED FOR GROUP:** *aaaaaaaa* **TYPE:** *bbbbbbbb* **CODE:** *ccc*

Explanation: An error occurred while purging members from the specified type/group.

Possible return codes are:

8 The target data set is enqueued.
12 An I/O error exists in target data set.
16 SCLM is unable to allocate data set.
20 SCLM internal error.

Programmer Response: Verify that the data set exists and the data set is not allocated exclusively to another job. Resubmit the job.

Project Administrator Response: If the return code = 20, contact SCLM Program Support.

FLM57011 **WARNING, UNABLE TO PURGE MEMBER(S) BECAUSE MEMBER(S) ARE MISSING FROM GROUP:** *aaaaaaaa* **TYPE:** *bbbbbbbb*

Explanation: One or more members in the specified type are missing. Only accounting information exists in the specified group. The accounting information will be deleted.

Programmer Response: Verify that no members involved in the promotion should have existed in the specified group.

Project Administrator Response: None.

FLM57101 **WARNING, ACCOUNTING INFORMATION IS NOT CURRENT FOR GROUP:** *aaaaaaaa* **TYPE:** *bbbbbbbb* **MEMBER:** *cccccccc*

Explanation: The accounting information for the member does not match the contents of the member. It is possible that the member has been updated outside of SCLM control.

Programmer Response: Define the member to SCLM using the SCLM editor or the SAVE service. If the member is not needed, delete it using the SCLM library utility or the DELETE service.

Project Administrator Response: None.

FLM57201 **PURGE OF INTERMEDIATE FORM FAILED FOR GROUP:** *aaaaaaaa*

Explanation: Unable to purge intermediate form from the group.

Programmer Response: See the message data set for all the messages related to this error.

Project Administrator Response: None.

FLM58000 **PROMOTE PROCESSOR COMPLETED** *aaaaaaaa* **ON** *bbbbbbbb*

Explanation: The promote processor completed.

Programmer Response: See the message data set for all the messages related to this error.

Project Administrator Response: None.

FLM59001 **INVOKING PROMOTE PROCESSOR**

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM61001 **THE REPORTS WILL APPEAR IN** *aaa(26)*

Explanation: This message is provided for information only.

Programmer Response: None

Project Administrator Response: None.

FLM61002 **THE MESSAGES WILL APPEAR IN** *aaa(26)*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM61007 **DATABASE CONTENTS UTILITY INITIATED -** *aaaaaaaa* **ON** *bbbbbbbb*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM61008 **NUMBER OF PAGES GENERATED FOR THE REPORT -** *aaa(10)*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM61009 **NUMBER OF PAGES GENERATED FOR THE TAILORED OUTPUT -** *aaa(10)*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM61011 **NO RECORDS FOUND FOR DATA TYPE -** *aaa(10)*

Explanation: SCLM cannot find a list of members for the data type entered.

Programmer Response: None.

Project Administrator Response: None.

FLM61015 **ERROR RETRIEVING ACCOUNTING OR CROSS-REFERENCE INFORMATION**

CODE: *aaa* **ERROR GROUP:** *bbbbbbbb* **TYPE:** *cccccccc* **MEMBER:** *dddddddd*

Explanation: No accounting record exists or could be retrieved for the specified member within the hierarchical view beginning at the group identified in the message.

Programmer Response: Possible return codes are:

8 SCLM did not find the member's accounting information. Register the member with SCLM using the edit function, migration utility, or the SAVE service. Run the processor again.

12 The member's accounting and dependency information was retrieved successfully; however, some of the dependency information failed verification processing. To determine the nature of the verification error, browse the member's accounting and dependency information by using the library utility. The utility performs this verification and displays the fields you want to validate. You may need to edit and then save the member to correct the problem.

16 SCLM found an invalid group in the project definition. Contact the project administrator.

20 A severe I/O error occurred. Contact the project administrator.

Project Administrator Response: Run IDCAMS against the accounting data set to determine the problem.

FLM61020 NO MEMBERS MATCHING SELECTION CRITERIA

Explanation: SCLM could not find a match for project, group, type, and member.

Programmer Response: Verify that the selection criteria are under SCLM control.

Project Administrator Response: None.

FLM61021 DATABASE CONTENTS UTILITY COMPLETED - *aaaaaaaa* ON *bbbbbbbb*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM61025 USER DEFINED DDNAME: *aaaaaaaa* FOR DBUTIL REPORT NOT ALLOCATED

Explanation: The ddname specified for the DBUTIL report was not allocated. If the DBUTIL function is invoked via the services, the ddname for the DBUTIL report is optional. If the ddname is not specified, the DBUTIL report is defaulted to the terminal. If a ddname is specified, it must be allocated.

Programmer Response: Verify that the user-supplied ddname for DBUTIL output is allocated. Resubmit the job.

Project Administrator Response: None.

FLM61028 USER DEFINED DDNAME: *aaaaaaaa* FOR TAILORED OUTPUT NOT ALLOCATED

Explanation: The ddname specified for the tailored output was not allocated. If the DBUTIL function is invoked via the services, the ddname for the tailored output is optional. If not specified, the tailored output is defaulted to the terminal. If a ddname is specified, it must be allocated.

Programmer Response: Verify that the user-supplied ddname for the tailored output is allocated. Resubmit the job.

Project Administrator Response: None.

FLM61030 USER DEFINED DDNAME: *aaaaaaaa* FOR DBUTIL MESSAGES IS NOT ALLOCATED

Explanation: The ddname specified for the messages is not allocated. If the DBUTIL function is invoked via the services, the ddname for the messages is optional. If a ddname is not specified, the messages are defaulted to the terminal. If a ddname is specified, it must be allocated.

Programmer Response: Verify that the user-supplied ddname for the messages is allocated. Resubmit the job.

Project Administrator Response: None.

FLM61035 TAILORED REPORT LINE LENGTH EXCEEDS LIMIT

Explanation: The output line that is written to the tailored file exceeded the 512 character limit.

Programmer Response: Verify that the length of the lines being written as output to the tailored file is not greater than 512. If it is greater than 512, change your formatted report line to contain SCLM variables that write 512 characters or less to the tailored file.

Project Administrator Response: None.

FLM62000 ARCHITECTURE REPORT PROCESSOR INITIATED - *aaaaaaaa* ON *bbbbbbbb*

Explanation: This message is provided for information only.

Programmer Response: None

Project Administrator Response: None.

FLM62001 STARTING ARCHITECTURE MEMBER TYPE EXCEEDS CUTOFF

Explanation: The architecture report could not be generated because the type of architecture member specified exceeded the type of architecture definition given for the cutoff of the report. The report cutoff should be equal to or lower than the architecture member kind.

For information on architecture members, see Chapter 2, "Architecture Definition."

Programmer Response: Specify a lower report cutoff and resubmit the job.

Project Administrator Response: None.

FLM62004 MISSING INPUT PARAMETER

Explanation: The programmer has specified blanks in one of the input parameter fields.

Programmer Response: Check the input parameters and resubmit the job.

Project Administrator Response: None.

FLM62008 INVALID CUTOFF PARAMETER: *aaa(24)*

Explanation: The report cutoff for the architecture report is invalid.

Programmer Response: Verify that the report cutoff parameter specified is:

CC For HL, LEC, and CC architecture members.

GEN For HL, LEC, and generic architecture members.

HL For HL architecture members.

LEC For HL and LEC architecture members.

NONE For all architecture members and source members (no cutoff).

TOP SOURCE
For all top source members and all architecture members.

Project Administrator Response: None.

FLM62024 MAXIMUM RECURSION LIMIT EXCEEDED WHILE PROCESSING MEMBER: *aaaaaaaa*

Explanation: In processing the given member, either the group of included architecture members exceeded the maximum allowed or the architecture member included itself and recursion limit was exceeded.

Programmer Response: Modify the architecture member and resubmit the job.

Project Administrator Response: None.

FLM62025 USER DEFINED DDNAME: *aaaaaaaa* **FOR ARCHITECTURE REPORT NOT ALLOCATED**

Explanation: The ddname specified for the architecture report was not allocated. If the architecture function is invoked via the services, the ddname for the architecture report is optional. If a ddname is not specified, the architecture report is defaulted to the terminal. If a ddname is specified, it must be allocated.

Programmer Response: Verify that the user-supplied ddname for the architecture report is allocated. Resubmit the job.

Project Administrator Response: None.

FLM62030 USER DEFINED DDNAME: *aaaaaaaa* **FOR ARCHITECTURE MESSAGES NOT ALLOCATED**

Explanation: The ddname specified for the architecture messages was not allocated. If the architecture function is invoked via the services, the ddname for the architecture messages is optional. If a ddname is not specified, the architecture messages are defaulted to the terminal. If a ddname is specified, it must be allocated.

Programmer Response: Verify that the user-supplied ddname for the architecture messages is allocated. Resubmit the job.

Project Administrator Response: None.

FLM62104 INVALID STATEMENT IN ARCHITECTURE MEMBER: *aaaaaaaa* **TYPE:** *bbbbbbbb*

Explanation: The specified architecture member contains an invalid statement. The architecture member may contain keywords that are specific to both an LEC and a CC (for example, an OBJ keyword and a LOAD keyword). The architecture member may also have two LMAP statements or a COPY keyword with an LMAP statement in the copied member. Any of these errors will cause this message to occur.

Programmer Response: This error will not affect the architecture report. Before attempting a build, check the specified architecture member for any of the above listed errors. Modify the architecture member

Project Administrator Response: None.

FLM62108 ERROR RETRIEVING ACCOUNTING INFORMATION

CODE: *aaa* **GROUP:** *bbbbbbbb*
TYPE: *cccccccc* **MEMBER:** *dddddddd*

Explanation: No accounting information exists or could be retrieved for the specified member.

Programmer Response: If the report was being run simply to view the high-level architecture of the system, no programmer response is necessary. If you want to see the entire system, including the included source members, you must register all source members with SCLM using the SCLM editor, the migration utility, or the SAVE service.

Project Administrator Response: None.

FLM62900 ARCHITECTURE REPORT PROCESSOR COMPLETED

Explanation: The architecture report processor finished executing.

Programmer Response: See the message data set for all the messages related to this error.

Project Administrator Response: None.

FLM69005 INVOKING ARCHITECTURE REPORT PROCESSOR

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM69010 INVOKING DATABASE CONTENTS UTILITY

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM69015 THE REPORT WILL APPEAR IN *aaa(26)*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM69020 THE MESSAGES WILL APPEAR IN *aaa(26)*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM69025 THE COMMANDS WILL APPEAR IN *aaa(26)*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM69030 DATABASE CONTENTS UTILITY RETURN CODE = *aaa(26)*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM70002 PARAMETER STRING EXCEEDS MAXIMUM SIZE ALLOWED FOR TRANSLATOR *aaaaaaaa*

Explanation: The parameter string for the specified translator is greater than the allowed maximum of 512 characters. The parameter string is formed by concatenating the values of the OPTIONS parameter of the FLMTRNSL macro with the PARM and the PARM*x* architecture member keywords. All SCLM variables in the resulting string are then replaced with the variables' values. Any of the allowable sources for creation of the parameter string could cause the parameter string size to be exceeded.

Programmer Response: Reduce the size of one of the sources for the parameter string.

Project Administrator Response: None.

FLM70003 SUBSTITUTION LIST EXCEEDS MAXIMUM SIZE ALLOWED FOR TRANSLATOR *aaaaaaaa*

Explanation: The ddname substitution list for the translator is greater than the maximum of 512 allowed. Every FLMALLOC macro for the translator causes an eight-character ddname to be put into the ddname substitution list.

Programmer Response: Either reduce the number of FLMALLOC macro invocations for the specified translator or change the PORDER parameter of the FLMTRNSL macro to 0 or 1 so that SCLM will not attempt to pass a ddname substitution list.

Project Administrator Response: None.

FLM70101 INVALID COPYLIB NAME: *aaa(44)*

Explanation: The copylib name is too long.

Programmer Response: Reduce the size of the copylib name. SCLM variables might be causing the name to expand to a larger size than expected.

Project Administrator Response: None.

FLM70212 INVOKING COPY ROUTINES FOR LANGUAGE: *aaaaaaaa*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM70214 INVOKING PURGE ROUTINE FOR LANGUAGE: *aaaaaaaa*.

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM70216 INVOKING TRANSLATORS FOR LANGUAGE: *aaaaaaaa*

Explanation: This message is provided for information only.

Programmer Response: None.

Project Administrator Response: None.

FLM70501 ERROR COPYING ALLOCATION DATA SET *aaa* FOR TRANSLATOR *bbb* TO LISTINGS DATA SET, CODE: *ccc*

Explanation: One of the following may have occurred:

- The listing data set was not allocated.
- The listing data set had an insufficient amount of space allocated.

Possible return codes are:

4 The input data set is empty; output data set is cleared.

8 The data was copied but truncated, or the directory entry data was not copied but an entry was created.

12 The ddnames are not allocated properly due to the following:

- The ddname is not allocated.
- The partitioned data set has no member name.
- The sequential data set member name is incorrect.

16 The output data set is full.

20 Data access failed due to the following:

- RACF protection
- Input member was not found.

24 The input parameter is invalid.

28 The member entry could not be created because the input member is an alias, or it has TTR notes (TTRN).

Programmer Response: Check listings data set and resubmit the job.

Project Administrator Response: None.

FLM70502 LISTINGS NOT COPIED BECAUSE BLANK LISTINGS DDNAME SPECIFIED

Explanation: SCLM did not copy listings to the listings data set because you specified a blank ddname. Therefore, you will not be able to see any listings that the translators produced.

Programmer Response: Specify the listings ddname for the given function and run the function again.

Project Administrator Response: None.

FLM70801 ERROR DEALLOCATING DATA SET NUMBER: *aaa* FOR LANGUAGE: *ccccccc* TRANSLATOR: *bbb*(16) CODE:*ddd*

Explanation: An error occurred while deallocating a data set for the specified translator. The file number identifies the allocated data set for that translator.

Note: For the FREE and END services called using a program, only the first line will appear. This message indicates that a data set for one of the translators defined for the specified language could not be deallocated. Since the condition occurs during cleanup, it can usually be treated as a

warning. Verify that the program has not deallocated the data sets specified for the language.

Possible return codes are:

8 SVC 99 error.

12 SCLM internal error. Contact SCLM Program Support.

16 Missing or incorrect data set name.

20 Invalid file attribute specified.

24 A member of a PDS was requested but the data set was not partitioned.

28 The requested member could not be found.

32 The requested member was not available.

>64 SVC 99 error, reason code (decimal):

528 Data set cannot be exclusively WRITE allocated.

5896 Data set does not exist.

Programmer Response: Resubmit the job. If the error recurs, contact SCLM Program Support.

Project Administrator Response: None.

FLM71002 ERROR INVOKING TRANSLATOR: *aaa(16)*

Explanation: SCLM could not invoke the specified translator. The load module containing the translator may be allocated exclusively to another job, or there is an error in the language definition that defines the translator.

Programmer Response: If the translator has been used successfully in the past and changes were not anticipated (for example, a new compiler release), invoke the processor again. If the translator is new or the problem recurs, contact SCLM Program Support.

Project Administrator Response: None.

FLM71004 TRANSLATOR RETURN CODE FROM *===> aaa(16) ===> bbbb*

Explanation: The return code from the invoked translator did not match the GOODRC parameter specified for the translator. Translator output, such as compiler listings, will be saved in the listings data set for the processor if requested in the language definition.

Programmer Response: Use the listings data set to locate and correct all errors identified by the translator. If the specified return code is acceptable for the translator, contact the project administrator.

Project Administrator Response: Change the GOODRC parameter of the FLMTRNSL macro, which defines the specified return code, in the project definition.

FLM71006 ERROR ALLOCATING DATA SET: *aaa(44)* **DDNAME:** *bbbbbbbb* **CODE:** *ccc*

Explanation: An error occurred while attempting to allocate a data set. Possible return codes are:

8 SVC 99 error.

12 Internal error. Contact SCLM program support.

16 Incorrect data set name.

20 Invalid file attribute specified.

24 A member of a PDS was requested but the data set is not partitioned.

28 The requested member could not be found.

32 The requested member was not available.

>64 SVC 99 error; reason code (decimal):

528 The data set cannot be exclusively WRITE allocated.

5896 The data set does not exist.

Programmer Response: Resubmit the job. If the error recurs, contact the project administrator.

Project Administrator Response: None.

FLM80001 'END' RECORD NOT FOUND IN THE "ACCOUNTING LIST INFO ARRAY"

Explanation: The accounting $list_info array has exceeded its buffer size.

Programmer Response: Contact the project administrator.

Project Administrator Response: Increase the size of the accounting $list_info array defined for the language on the FLMLANGL macro. For more information on the FLMLANGL macro and how to specify the size of the accounting $list_info array, see "FLMLANGL Macro" on page 224.

FLM80002 INVALID RECORD TYPE FOUND IN THE "ACCOUNTING LIST INFO ARRAY" RECORD TYPE: *aaaaaaaa*

Explanation: The record type is unknown.

Programmer Response: Contact the project administrator.

Project Administrator Response: Either the parser created or the user defined an accounting $list_info array that contained an invalid record type. If a parser was used to create the array, check that the passed values are correct. For more information, see Chapter 5, "SCLM Services."

FLM80003 INVALID COMPOOL NAME FOUND IN THE "ACCOUNTING LIST INFO ARRAY" RECORD KIND: *aaaa* **COMPOOL NAME:** *bbbbbbbb*

Explanation: The accounting $list_info array contained an entry for a compool with either an invalid or blank associated compool name.

Programmer Response: If a parser was used, parse the members in question again. If the same error occurs or a parser was not used, contact the project administrator.

Project Administrator Response: Either the parser created or the user defined an accounting $list_info array that contained an invalid or blank compool name. If a parser was used to create the array, check that the passed values are correct. For more information, see Chapter 5, "SCLM Services."

FLM80004 INVALID INCLUDE NAME FOUND IN THE "ACCOUNTING LIST INFO ARRAY" RECORD KIND: *aaaa* **INCLUDE NAME:** *bbbbbbbb*

Explanation: The accounting $list_info array contained an entry for an include with either an invalid or blank associated include name.

Programmer Response: If a parser was used, parse the members in question again. If the same error occurs or a parser was not used, contact the project administrator.

Project Administrator Response: Either the parser created or the user defined an accounting $list_info array that contained an invalid or blank include name. If a parser was used to create the array, check that the passed values are correct. For more information, see Chapter 5, "SCLM Services."

FLM80005 INVALID CU DATA FOUND IN THE "ACCOUNTING LIST INFO ARRAY"

RECORD KIND: *aaaa*
CU NAME: *bbb(55) ccc(55)* **CU TYPE:** *d*
GENERIC FLAG: *e* **DEPEND NAME:** *fff(55) ggg(55)* **DEPEND CU TYPE:** *h* **DEPENDENCY TYPE:** *i*

Explanation: The accounting $list_info array contained invalid data for a CU.

Programmer Response: If you used a parser, parse the members in question again. If the same error occurs or you did not use a parser, contact the project administrator.

Project Administrator Response: Either the parser created or the user defined an accounting $list_info array that contained the invalid CU data. If a parser was used to create the array, check that the passed values are correct. For more information, see Chapter 5, "SCLM Services."

FLM80010 CONFLICTING GENERIC FLAGS FOUND FOR THE SAME CU IN THE "ACCOUNTING LIST INFO ARRAY"

RECORD KIND: *aaaa* **CU NAME:** *bbb(55) ccc(55)* **CU TYPE:** *d*

Explanation: Dependencies for the same CU have different generic flags. The generic flags must always be the same for all dependencies within a CU.

Programmer Response: Contact the project administrator.

Project Administrator Response: Either the parser created or the user defined an accounting $list_info array that contained the invalid CU data. If a parser was used to create the array, check that the passed values are correct. For more information, see Chapter 5, "SCLM Services."

FLM80011 CONFLICTING DEPENDENCY TYPE FLAGS FOUND IN THE "ACCOUNTING LIST INFO ARRAY"

RECORD KIND: *aaaa* **CU NAME:** *bbb(55) ccc(55)* **CU TYPE:** *d* **GENERIC FLAG:** *e* **DEPEND NAME:** *fff(55) ggg(55)* **DEPEND CU TYPE:** *h*

Explanation: Different type flags exist for the same CU within the accounting $list_info array. For example, the same CU is specified as a SPEC in one instance and a BODY in another.

Programmer Response: Contact the project administrator.

Project Administrator Response: Either the parser created or the user defined an accounting $list_info array that contained the invalid CU data. If a parser was used to create the array, check that the passed values are correct. For more information, see Chapter 5, "SCLM Services."

FLM80012 ONLY ONE CU RECORD MAY BE PRESENT IN THE "ACCOUNTING LIST INFO ARRAY"

Explanation: If a CU record with a type of X exists in the accounting $list_info array, it can be the only CU record in the array.

Programmer Response: Contact the project administrator.

Project Administrator Response: Either the parser created or the user defined an accounting $list_info array that contains the invalid CU data. If a parser was used to create the array, check that the passed values are correct. For more information, see Chapter 5, "SCLM Services."

FLM80020 ERROR ALLOCATING THE CHANGE CODE VERIFICATION ROUTINE

DATA SET DSNAME: *aaa(44)*

Explanation: The specified data set could not be allocated. The data set may not exist or it may be allocated exclusively to another user or job.

Programmer Response: Do the following:

- Allocate the required data set and move the change code verification routine into it.
- Free the data set so that it can be allocated in SHR mode.

Project Administrator Response: None.

FLM80021 ERROR INVOKING THE CHANGE CODE VERIFICATION ROUTINE

NAME: *aaaaaaaa* **DSNAME:** *bbb(44)*

Explanation: The change code verification routine could not be invoked. Verify that the routine exists within the specified data set.

Programmer Response: If the routine does not exist, move it into the proper data set. If it does exist, contact SCLM Program Support.

Project Administrator Response: None.

FLM80022 INVALID CHANGE CODE: *aaaaaaaa* **CHANGE CODE VERIFICATION ROUTINE RETURN CODE:** *bbb*

Explanation: The change code verification routine completed with a return code > 0.

Programmer Response: Check change code verification routine for return code explanations. See "Change Code Verification Routine Specification" on page 204 for more information.

Project Administrator Response: None.

FLM80030 THE SIZE OF THE "ACCOUNTING LIST INFO ARRAY" HAS BEEN EXCEEDED

Explanation: The accounting $list_info array has insufficient space to contain the data specified. Dependency information, user data records, and change code information must fit into the array.

Programmer Response: If possible, eliminate unneeded user data and/or change code information from the accounting record using the SCLM library utility. If all of the information is required, contact the project administrator.

Project Administrator Response: Increase the size of the accounting $list_info array defined for the language on the FLMLANGL macro. For more information on the FLMLANGL macro and how to specify the size of the accounting $list_info array, see "FLMLANGL Macro" on page 224.

FLM80031 "ACCOUNTING LIST INFO ARRAY" MUST ONLY CONTAIN CHANGE CODE RECORDS

RECORD KIND: *aaaa*

Explanation: An invalid change code record was found in the accounting $list_info array. Verify that the change codes in the accounting record are correct.

Programmer Response: Contact the project administrator.

Project Administrator Response: Ensure that correct change code values exist in the accounting record. If not, check the parser. If it is correct, increase the size of the accounting $list_info array defined for the language on the FLMLANGL macro. For more information on the FLMLANGL macro and how to specify the size of the accounting $list_info array, see "FLMLANGL Macro" on page 224.

FLM80035 "$LIST_INFO" DOES NOT CONTAIN A CHANGE CODE TO BE VERIFIED

Explanation: The project definition indicates that change code verification is in affect, but there are no change codes in the accounting $list_info array.

Programmer Response: Contact the project administrator.

Project Administrator Response: The "accounting list info array" must exist and contain valid change code information if change code verification is in effect. Determine why the "accounting list info array" pointer is NIL.

FLM80500 ACCESS KEY INCORRECT

ACCESS KEY: *aaa(16)* **GROUP:** *bbbbbbbb*
TYPE: *cccccccc* **MEMBER:** *dddddddd*

Explanation: The access key specified was invalid. The member is currently locked out with an access key. Only if you specify the correct access key can you save the member in the SCLM hierarchy.

Programmer Response: If another user has the member "checked out," wait until it is checked in. Otherwise, specify the correct access key for the member.

Project Administrator Response: None.

FLM81001 INVALID APPLICATION ID: *aaaaaaaa*

Explanation: An INIT or END operation was attempted with an invalid application ID specified.

Programmer Response: Make sure that the application ID passed back from the START function is used in the INIT and END functions.

Project Administrator Response: None.

FLM81201 INVALID PROJECT IDENTIFIER: *aaaaaaaa*

Explanation: An invalid project identifier was passed to an SCLM service. A valid project identifier is required by the SCLM service requested.

Programmer Response: Supply a valid project identifier in the SCLM service parameter list.

Project Administrator Response: None.

FLM81202 INVALID PROJECT DEFINITION NAME: *aaaaaaaa*

Explanation: An invalid project definition name was passed to an SCLM service. A valid project definition name is required by the SCLM service requested.

Programmer Response: Supply a valid project definition name in the SCLM service parameter list.

Project Administrator Response: None.

FLM81203 MAXIMUM SCLM ID LIMIT EXCEEDED

Explanation: No more SCLM IDs are available at this time.

Programmer Response: Free some previously allocated SCLM IDs.

Project Administrator Response: None.

FLM81204 ERROR INITIALIZING THE PROJECT DEFINITION, CODE: *aaa*

Explanation: The return codes are defined below.

Programmer Response: Possible return codes are:

4 The specified project definition load module is not RMODE(24). Generate the project definition load module again and specify the RMODE(24) parameter to the linkage editor.

8 An error occurred while attempting to obtain the specified project definition for the project. Check the project definition and resubmit.

12 The project definition is out of date. Reassemble the project definition with new SCLM macros. Resubmit the job.

16 The project name specified does not match the project name in project definition. Verify that the project name (on the FLMABEG macro) specified and the project name in the project definition are the same.

20 An attempt to open or close the project definition failed.

24 The project definition data set could not be allocated.

Project Administrator Response: None.

FLM81205 ERROR ACCESSING INTERNAL DATA DATA SET(S) FOR THE PROJECT, CODE: *aaa*

Explanation: An error occurred while attempting to access internal data.

Programmer Response: Possible return codes are:

8 The cross-reference data set could not be opened. Verify that the user is authorized to update the cross-reference data set. Verify that the cross-reference data set is intact. Resubmit the job. Contact the project administrator if the problem recurs.

12 The cross-reference data set could not be allocated. Check the project definition for this project to ensure that the correct cross-reference data set has been specified. If it is, verify that the data set is not allocated exclusively to another job. Check the input parameters and resubmit the job. If the problem recurs, contact the project administrator.

16 The backup accounting data set could not be opened. Verify that the user is authorized to update the backup accounting data set. Verify that the backup accounting data set is intact. Resubmit the job. If the problem recurs, contact the project administrator.

20 The backup accounting data set could not be allocated. Check the project definition for this project to ensure that the correct backup accounting data set has been specified. If it is correct, verify that the data set is not allocated exclusively to another job. Check input parameters, and resubmit the job. If the problem recurs, contact the project administrator.

24 The accounting data set could not be opened. Verify that the user is authorized to update the accounting data set. Verify that the accounting data set is intact. Resubmit the job. If the problem recurs, contact the project administrator.

28 The accounting data set could not be allocated. Check the project definition for this project to ensure that the correct accounting data set has been specified. If it is, verify that the data set is not allocated exclusively to another job. Check input parameters and resubmit the job. If the problem recurs, contact the project administrator.

Project Administrator Response: None.

FLM81302 ERROR PROCESSING MEMBERS WITH ACCOUNTING INFORMATION TYPE: INITIAL

Explanation: This is a warning message.

Programmer Response: None.

Project Administrator Response: None.

FLM82002 MEMBER IS NON-EDITABLE
GROUP: *aaaaaaaa* **TYPE:** *bbbbbbbb*
MEMBER: *cccccccc*

Explanation: The specified member cannot be edited because it is SCLM output. This member cannot be updated in this way.

Programmer Response: If the user needs to edit a non-editable member, select a new member and copy the non-editable data into the new member.

Project Administrator Response: None.

FLM82003 MEMBER IS NOT LOCKED
GROUP: *aaaaaaaa* **TYPE:** *bbbbbbbb*
MEMBER: *cccccccc*

Explanation: The UNLOCK service was called but the requested member was not locked. If this error occurred while you were using the STORE service, no accounting information was available. If this error occurred while you were in the editor, the accounting information created at the beginning of your edit session was lost.

Programmer Response: For the STORE service, verify that the LOCK service completed successfully before calling the STORE service. For the UNLOCK service, this message is a warning and can be ignored. If you are in an edit session, your data has been saved. However, the accounting information is lost. Cancel this edit session and re-edit the member. Then issue the SAVE command immediately to establish accurate accounting information.

Project Administrator Response: None.

FLM82004 LANGUAGE: *aaaaaaaa* **CANNOT BE USED FOR EDITABLE MEMBERS**

Explanation: The language specified to the PARSE routine is not a valid language. Check the list of valid languages in the project definition.

Programmer Response: Use a language that is in the project definition. If the language that you need is not in the project definition, contact the project administrator.

Project Administrator Response: Add the required language to the project definition in the form of a language definition and reassemble. For more information on language definitions, see "New Language Definitions" on page 234.

FLM82005 INPUT PARAMETER "ERROR_LISTINGS_ONLY" MUST BE 'Y' OR 'N'
ERROR_LISTINGS_ONLY: *a*

Explanation: An invalid parameter was passed to an SCLM service.

Programmer Response: Supply the valid parameter in the SCLM service parameter list.

Project Administrator Response: None.

FLM82006 INPUT PARAMETER "VERIFY_CC" MUST BE 'Y' OR 'N',
VERIFY_CC: *a*

Explanation: An invalid parameter was passed to an SCLM service.

Programmer Response: Supply the valid parameter in the SCLM service parameter list.

Project Administrator Response: None.

FLM82008 INPUT PARAMETER "SUB_DRAWDOWN_MODE" MUST BE 'C' OR 'U'
SUB_DRAWDOWN_MODE: *a*

Explanation: An invalid parameter was passed to an SCLM service.

Programmer Response: Supply the valid parameter in the SCLM service parameter list.

Project Administrator Response: None.

FLM82203 THE MEMBER HAS ACCOUNTING INFORMATION WITH TYPE: EDITABLE AUTHORIZATION CODE CANNOT BE UPDATED

GROUP: *aaaaaaaa* **TYPE:** *bbbbbbbb*
MEMBER: *cccccccc*

Explanation: The authorization code in the service parameter list does not match the authorization code already assigned to the member. Only the library utility is capable of changing an existing member.

Programmer Response: Use the SCLM library utility for this function.

Project Administrator Response: None.

FLM82301 EDITABLE MEMBER'S ACCESS KEY IS BLANK

GROUP: *aaaaaaaa* **TYPE:** *bbbbbbbb*
MEMBER: *cccccccc*

Explanation: You tried to unlock a member that has an editable accounting record and the access key was already blank.

Programmer Response: If you want to unlock the member rather than just reset the access key, use the DELETE service to delete the member's accounting record.

Project Administrator Response: None.

FLM82401 ERROR PROCESSING SYSTEM LIBRARIES FOR PARSING

CODE: *aaa* **LANGUAGE:** *bbbbbbbb* **ERROR DSNAME:** *ccc(44)*

Explanation: SCLM was unable to allocate the system library defined in the language definition for the language specified in the message.

Programmer Response: Check that the system libraries specified in the language definition exist and are not allocated exclusively. If one or more do not exist, contact the project administrator.

Project Administrator Response: Remove the invalid system libraries from the language definition and regenerate the project definition.

FLM82501 EXISTING CU'S AUTHORIZATION CODE NOT DEFINED TO GROUP

CU NAME: *aaa(55) bbb(55)* **CU TYPE:** *cccc*
CU QUALIFIER: *ddddddd*
GROUP: *eeeeeeee*
ERROR GROUP: *ffffffff*
AUTHORIZATION CODE: *gggggggg*

Explanation: The authorization code is not defined to the group. This implies that the CU is not authorized to replace the version of the member in the error group.

Programmer Response: It is possible that the function will succeed with a different authorization code. Contact the project administrator for a list of authorization codes that are valid for this group. If none of the authorization codes defined to the group work, try the same function at a different group. Contact the project administrator if all attempts fail.

Project Administrator Response: The list of valid authorization codes defined for *group* may be found in the project definition on the FLMGROUP macro. Do not attempt to add authorization codes to the project definition unless you are familiar with the risks outlined in "Authorization Code Usage" on page 252.

FLM82502 INPUT PARAMETER "$STATS_INFO" CANNOT BE NIL

Explanation: $STATS_INFO has not been initialized with data. The SCLM service requested must have data in this record.

Programmer Response: Initialize $STATS_INFO and invoke the service again.

Project Administrator Response: None.

FLM82503 DUPLICATE CHANGE CODE RECORDS FOUND IN THE "ACCOUNTING LIST INFO ARRAY"

RECORD KIND: *aaaa*
CHANGE CODE: *bbbbbbbb*

Explanation: The change code was specified multiple times within the same accounting $list_info array. A service call using a user-specified parser had duplicate entries for the change code.

Programmer Response: Remove duplicate entries for the change code in the SCLM services parameter list and call the service again.

Project Administrator Response: Rewrite the involved parser to add logic that will remove duplicate entries. For more information, see "Invoking User-Defined Parsers" on page 239.

FLM82504 DUPLICATE COMPOOL RECORDS FOUND IN THE "ACCOUNTING LIST INFO ARRAY"

RECORD KIND: *aaaa*
COMPOOL NAME: *bbbbbbbb*

Explanation: The compool was specified multiple times within the same accounting $list_info array. A service call to a user-specified parser generated duplicate entries for the compool.

Programmer Response: If the STORE service was called, remove duplicate entries for the compool in the SCLM services parameter list and call the service again. If the SAVE service was called, or data was passed to the STORE service as a result of the PARSE service, contact the project administrator.

Project Administrator Response: Rewrite the involved parser to add logic that will remove duplicate entries. For more information, see "STORE—Store Member Information in an Accounting Record" on page 158.

FLM82505 DUPLICATE INCLUDE RECORDS FOUND IN THE "ACCOUNTING LIST INFO ARRAY"

RECORD KIND: *aaaa*
INCLUDE NAME: *bbbbbbbb*

Explanation: The include was specified multiple times within the same accounting $list_info array. A service call to a user-specified parser generated duplicate entries for the include.

Programmer Response: If the STORE service was called, remove duplicate entries for the include in the SCLM services parameter list and call the service again. If the SAVE service was called, or data was passed to the

STORE service as a result of the PARSE service, contact the project administrator.

Project Administrator Response: Rewrite the involved parser to add logic that will remove duplicate entries. For more information, see "STORE—Store Member Information in an Accounting Record" on page 158.

FLM82506 DUPLICATE CU RECORDS FOUND IN THE "ACCOUNTING LIST INFO ARRAY"

RECORD KIND: *aaaa*
CU NAME: *bbb(55) ccc(55)* **CU TYPE:** *d*
GENERIC FLAG: *e*
DEPEND NAME: *fff(55) ggg(55)*
DEPEND CU TYPE: *h*

Explanation: The CU was specified multiple times within the same accounting $list_info array. A service call to a user-specified parser generated duplicate entries for the CU.

Programmer Response: If the STORE service was called, remove duplicate entries for the CU in the SCLM services parameter list and call the service again. If the SAVE service was called, or data was passed to the STORE service as a result of the PARSE service, contact the project administrator.

Project Administrator Response: Rewrite the involved parser to add logic that will remove duplicate entries. For more information, see "STORE—Store Member Information in an Accounting Record" on page 158.

FLM82507 ERROR ALLOCATING THE SPECIFIED MEMBER

GROUP: *aaaaaaaa* **TYPE:** *bbbbbbbb*
MEMBER: *cccccccc*

Explanation: The member does not exist in the specified group and type.

Programmer Response: Put the member in the hierarchy, or remove the reference to the member source code or member lists.

Project Administrator Response: None.

FLM82508 CU LOCKED ELSEWHERE

CU NAME: *aaa(55) bbb(55)* **CU TYPE:** *cccc*
CU QUALIFIER: *dddddddd*
ERROR GROUP: *eeeeeeee*
TYPE: *ffffffff* **MEMBER:** *gggggggg*
ERROR AUTHORIZATION CODE: *hhhhhhhh*

Explanation: The CU has already been updated in another hierarchical view or the CU is in the current view but in another type. The changes currently reside in the group specified in this message. This group is not in your view of the hierarchy. You cannot update the member because you would not be working with the most current version of the member.

Programmer Response: Promote the member into a group that is in your hierarchy (that is, one that appears on your SCLM Edit - Entry panel). If the member cannot be promoted, you must delete the member and its accounting information in the error group using the SCLM library utility or the DELETE service.

Project Administrator Response: None.

FLM82509 DRAWDOWN VERIFICATION ERROR

CODE: *aaa* **CU NAME:** *bbb(55) ccc(55)*
CU TYPE: *dddd* **CU QUALIFIER:** *eeeeeeee*
GROUP: *ffffffff* **AUTHORIZATION CODE:** *gggggggg*

Explanation: One of the following errors has occurred:

- The group is an invalid SCLM group.
- An I/O error occurred in retrieving the XREF record from the accounting database.

Programmer Response: Correct the group and contact the project administrator.

Project Administrator Response: Verify that the cross-reference record in the accounting database is correct.

FLM82511 CU DRAWN DOWN FROM ANOTHER MEMBER

CU NAME: *aaa(55) bbb(55)* **CU TYPE:** *cccc*
CU QUALIFIER: *dddddddd*
DRAWN DOWN FROM GROUP: *eeeeeeee*
TYPE: *ffffffff* **MEMBER:** *gggggggg*

Explanation: SCLM is now tracking two sessions of this compilation unit in separate members.

Programmer Response: None.

Project Administrator Response: None.

FLM82601 INPUT PARAMETER "PARSE_MODE" MUST BE 'C' OR 'U',

PARSE_MODE: *a*

Explanation: An invalid parameter was passed to an SCLM service.

Programmer Response: Supply the valid parameter in the SCLM service parameter list.

Project Administrator Response: None.

FLM82602 "LANGUAGE" CANNOT BE DEFAULTED AN EDITABLE ACCOUNTING RECORD DOES NOT EXIST FOR THE MEMBER

GROUP: *aaaaaaaa* **TYPE:** *bbbbbbbb*
MEMBER: *cccccccc*

Explanation: A valid language has not been assigned to the member specified in this message. The language is obtained from SCLM accounting information.

Programmer Response: Add a valid language to the service call or command.

Project Administrator Response: None.

FLM82603 WARNING: A PARSER ERROR OCCURRED BUT AN UNCONDITIONAL PARSE WAS REQUESTED

Explanation: An error occurred while parsing the member but you requested an unconditional parse. The SAVE service continues and saves the statistical and dependency information that the parser returned for the member.

Programmer Response: If possible, you should try to correct the parser errors. A parser error can cause incorrect dependency information to be saved for the

member. If the parser error cannot be corrected at this time, call the SAVE service for this member at a later date to correct the parser error.

Project Administrator Response: None.

FLM84101 INPUT PARAMETER "DELETE_FLAG" MUST BE 'BMAP', 'ACCT', OR 'TEXT'

DELETE_FLAG: *aaaa*

Explanation: An invalid parameter was passed to the DELETE service.

Programmer Response: Supply the valid parameter in the SCLM service parameter list. See "DELETE—Delete Database Components" on page 129 for more information.

Project Administrator Response: None.

FLM84200 NEW AUTHORIZATION CODE IS EQUAL TO OLD AUTHORIZATION CODE. NO CHANGE IS REQUIRED FOR: GROUP: *aaaaaaaa* **TYPE:** *bbbbbbbb* **MEMBER:** *cccccccc*

Explanation: The authorization code currently assigned to the specified member is equal to the new authorization code.

Programmer Response: Verify that the new authorization code was specified correctly.

Project Administrator Response: None.

FLM84204 WARNING, MEMBER: *aaaaaaaa* **COULD BE REPLACED BY MEMBER AT GROUP:** *bbbbbbbb* **TYPE:** *cccccccc* **WITH AUTHORIZATION CODE:** *dddddddd*

Explanation: The member that is being updated was found at a lower group. The lower group member is within the promotable hierarchy and could replace the member that is currently being updated. The update is performed as requested.

Programmer Response: Report this situation to the project administrator.

Project Administrator Response: The authorization code for the lower group member should be updated to match the higher group member to avoid overlaying the update made in the higher group.

FLM87100 ERROR, PARAMETER STRING MUST BE SHORTER THAN *aaa* **CHARS LONG**

Explanation: The input parameter string exceeded the maximum length.

Programmer Response: Shorten the input parameter string to a valid length.

Project Administrator Response: None.

FLM87103 RECURSIVE "FILE" COMMAND INVOCATIONS ARE NOT ALLOWED

Explanation: A FILE command cannot be invoked within another FILE.

Programmer Response: Remove the recursive occurrence of the FILE command. The contents of the referenced data set can be copied directly into the original data set if desired.

Project Administrator Response: None.

FLM87105 THE COMMAND IS NOT SUPPORTED

COMMAND: *aaa(60)*

Explanation: The command is not supported by this release of SCLM.

Programmer Response: For a list of and descriptions of valid SCLM service commands, see Chapter 5, "SCLM Services."

Project Administrator Response: None.

FLM87107 *aaaaaaaa bbb(24)* **FOR MEMBER** *cccccccc* **AT** *ddddddd*, **CODE:** *eee*

Explanation: The FLMCMD command termination message (*aaaaaaaa*) represents the command that was executed. *bbb(24)* represents the completion status of the command.

Programmer Response: For information on the return code, see the appropriate service in Chapter 5, "SCLM Services."

FLM87110 ERROR FOUND ON LINE *aaaa* **OF** *bbbbbbbb* **DATA SET**

Explanation: Check to see if another error message was printed. If it was, correct the error indicated by the other error message first. If the error message FLM87110 is the only error message printed, then an error was found on the indicated line number of the data set.

Programmer Response: The error should be corrected in the data set. For more details, see Chapter 5, "SCLM Services."

Project Administrator Response: None.

FLM87115 DBUTIL *aaa(9)* **AT** *bbbbbbbb*, **CODE:** *ccc*

Explanation: Completion message for DBUTIL.

Programmer Response: For information on the return code, see "DBUTIL—Generate a Tailored Data Set and Report" on page 124.

Project Administrator Response: None.

FLM87120 ERROR, *aaa(24)* **PARAMETER IN COLUMN** *bbb* **IS TOO LONG**

Explanation: The parameter is longer than the maximum allowed.

Programmer Response: Shorten the parameter. For more details on the SCLM services, see Chapter 5, "SCLM Services."

Project Administrator Response: None.

FLM87125 ERROR, *aaa(24)* **PARAMETER MUST BE SPECIFIED**

Explanation: The parameter has not been specified.

Programmer Response: Add the parameter to the SCLM service invocation. For more details on the SCLM services, see Chapter 5, "SCLM Services."

Project Administrator Response: None.

FLM87130 INVALID VALUE IN COLUMN *aaa* FOR *bbb(24)* PARAMETER

Explanation: The value for the parameter is invalid. The column number identifies the starting location of the invalid value with the command.

Programmer Response: Correct the value of the parameter in the command data set. For more details on the SCLM services, see Chapter 5, "SCLM Services."

Project Administrator Response: None.

FLM87133 EXTRANEOUS PARAMETER(S) DETECTED IN SERVICE CALL

Explanation: An SCLM service was passed a parameter string that contained more parameters than it requires.

Programmer Response: Remove the extra parameters from the SCLM service invocation. For more details on the SCLM services and their parameters, see Chapter 5, "SCLM Services."

Project Administrator Response: None.

FLM87135 THE DDNAME: *aaaaaaaa* IS ALREADY IN USE

Explanation: The ddname is currently reserved for use by the command processor.

Programmer Response: Change the ddname and resubmit the command.

Project Administrator Response: None.

FLM87140 THE DDNAME: *aaaaaaaa* HAS NOT BEEN ALLOCATED

Explanation: The ddname specified was passed to SCLM as a parameter, but the ddname has not been allocated.

Programmer Response: Allocate the ddname and resubmit the command.

Project Administrator Response: None.

FLM87150 THE COMMAND INVOCATION IS TOO LONG

Explanation: The command invocation statement is longer than the maximum 256 characters allowed.

Programmer Response: Edit the command and resubmit it.

Project Administrator Response: None.

FLM87201 SCLM ID: *aaaaaaaa* IS NOT IN USE

Explanation: The SCLM ID corresponds to members that have been locked but not freed by either the UNLOCK or the STORE service. These will be converted from initial state to lockout. This is just a warning message.

Programmer Response: None.

Project Administrator Response: None.

FLM87202 INVALID SCLM ID: *aaaaaaaa*

Explanation: The syntax of the SCLM ID is not valid. SCLM IDs are generated in the format FLM*ddddd* where *d* represents a digit from 0-9.

Programmer Response: Check the SCLM ID specified for accuracy, or make sure you used the INIT service to generate an SCLM ID before you try to use the FREE service on it.

Project Administrator Response: None.

FLMCMD Return Codes

General-Use Programming Interface

The FLMCMD return codes are a general-use programming interface, which you can use for programming purposes.

The SCLM command processor and translators provide you with return codes. This part of the chapter lists the categories of return codes.

FLMCMD

Each function returns a numeric code, called a return code, that indicates the results of the operation.

0 Successful completion. Messages may or may not be generated.
4 A warning condition. Messages may or may not be generated.
8 An error condition. Messages are generated.
12 Maximum application ID limit exceeded.
16 SCLM table verification failed.
20 NLS table verification failed.
24 Unable to load the SCLM table (FLMTABLE).
28 Unable to load the NLS table or the SLM I/O load module (FLMIO24).

End of General-Use Programming Interface

SCLM Translator Return Codes

General-Use Programming Interface

The translator return codes are a general-use programming interface, which you can use for programming purposes.

The translators return the following codes for the specified functions.

FLMDEBG

0 Indicates a successful completion.
8 Indicates that allocation of a source file failed. Check the input parameter from the translator.

FLMLA

0 Indicates successful completion.
4 Indicates a warning condition. See error messages for details.
8 Indicates a syntax error condition. See error messages for details.
12 Indicates a severe error condition. Messages are not generated.

FLMLABND

0 Indicates successful completion.
8 Indicates an error condition. SCLM found more than one compilation unit.
12 Indicates an error condition. SCLM found an invalid input parameter or could not find an input parameter.
16 Indicates an error condition. SCLM did not find any compilation unit.
20 Indicates an error condition. SCLM found extraneous information after the compilation unit.

FLMLSS

0 Indicates successful completion.
4 Indicates a warning condition. The continuation line limit of 3000 characters was exceeded.
8 Indicates a parser translator error condition.
12 Indicates a parser FLMLI error condition.
16 Indicates an error condition. SCLM found an invalid input parameter or could not find an input parameter.
20 Indicates an error condition. There are too many includes and compools for the size of the $info_list array.

:h3'.FLMS1S

0 Successful completion.
16 SCLM table verification failed.
20 NLS table verification failed.
24 Unable to load SCLM table (FLMTABLE).
28 Unable to load the NLS table or the SCLM I/O load module (FLMIO24).

FLMS7C

0 Successful completion.
4 A warning condition. Messages are generated.
8 An error condition. Messages are generated.
12 Maximum application ID limit exceeded.
16 SCLM table verification failed.
20 NLS table verification failed.
24 Unable to load the SCLM table (FLMTABLE).
28 Unable to load the NLS table or the SCLM I/O load module (FLMIO24).

FLMTALI

8 The input option string is invalid.

FLMTAPM

8 The input parameters are invalid.

FLMTAPV

0 SCLM found a string for every compilation unit.
4 SCLM found the alternate search string at least once. Alternate search string + search string = the number of compilation units passed.
8 SCLM did not find search strings for every compilation unit.

FLMTMSI

0 See the SCRIPT/VS translator code explanation.
4 See the SCRIPT/VS translator code explanation.
8 See the SCRIPT/VS translator code explanation. Messages are generated.
12 See the SCRIPT/VS translator code explanation.
16 See the SCRIPT/VS translator code explanation.
20 See the SCRIPT/VS translator code explanation.
24 SCLM did not allocate TEXTOUT.
28 SCLM did not allocate TEXTIN.
32 SCLM cannot generate a unique name.
36 The user ID was not specified in the input.

FLMTSPLT

0 Indicates a successful completion.
8 Indicates an error in allocating the SCLM hierarchy. This error is based on input from the translator.
12 An "above the 16 megabyte line" object member for the main unit which was created previously is missing. Delete the load module definition for the main unit and resubmit.

End of General-Use Programming Interface

Glossary of SCLM Terms

A

access key. An identifier used to restrict access to a member.

accounting record. Internal data record containing statistical, historical, and dependency information.

application. Software that performs a service for an end user.

architecture. The organization of software components to form integrated applications.

architecture definition. The specification of relationships between software components of an application.

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.

authorization code. An identifier used by SCLM to control authority to update and promote members within a hierarchy.

authorization group. A set of authorization codes.

B

build. To process architecture members through translators defined in the appropriate language definition. During a build, compilers and linkage editors may be invoked.

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. Reason code associated with a software update.

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.

compilation unit. The smallest compilable unit for the Ada language.

complib. A library containing frozen project level compools.

compool. A JOVIAL data mapping structure.

compool reference. A reference to a JOVIAL data mapping structure that SCLM must compile before it can compile the current member.

concurrent updates. Two programmers update the same member in different ways at different development levels.

conditional reference. An include or compool reference construct that depends on information outside the scope of a single line.

copylib. A library containing include referenced source code.

cross-reference record. Internal data record containing Ada compilation unit/member relationship information.

D

data. All information stored in the SCLM database.

database. SCLM-controlled data sets for a project.

database administrator. Individual responsible for customization and maintenance of an SCLM database.

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.

dependency. A software component necessary for the completion of another software component.

development level. Level of an SCLM hierarchy which has no subordinate level.

downward dependency. A dependency indicating a compilation unit which must be compiled after the current compilation unit is compiled.

drawdown. To copy a member or a compilation unit to a development group from its first appearance in a higher key or primary group in the library concatenation.

dynamic include. An include for a source member that cannot be resolved until after the translator invocation.

© Copyright IBM Corp. 1989, 1990

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.

G

group. A set of project data sets with the same middle-level qualifier that contain the different kinds of data maintained for that project.

H

hierarchical view. A concatenation of groups to form a complete project.

hierarchy. The organization of project database groups where each group is subordinate to the one above it.

I

include. A member that is necessary for the proper construction of a module (defined by an architecture definition).

include structure. A generic term for code that you insert when the source member is compiling.

internal data set. Data sets that contain statistical, include, dependency, status and tracking information about all controlled members.

IS SEPARATE declaration. The specification in an Ada compilation unit that a subunit is to be compiled separately.

K

key group. A group that data is moved into (as opposed to copied into) during promotion.

L

language definition. A process or set of processes that source members associated with this language will undergo during Build.

layer. A given tier of the hierarchy, made up of levels of equivalent rank.

level. A (complete) set of libraries forming a single element of a hierarchy.

library (MVS). A partitioned data set.

line commands. Editing commands that are entered directly on the line number of the line or lines to be affected.

lock. To preclude other programmers from updating a member (usually associated with drawdown).

M

maximum promotable level. The topmost level to which a segment can be promoted.

member. The discrete element of an SCLM database, representing a single data type of a software component.

migration. Introduction of software components into an SCLM database.

N

non-key group. A group that data is copied into (as opposed to moved into) during promotion.

O

out-of-scope software component. An architecture that is referenced with a LINK or CREF statement but not with an INCL statement; it is not within the domain of the architecture specified.

P

parse. To scan the source input from an edit session (or migration request) to gather SCLM internal data.

predecessor date/time. The last modified date/time stamp taken from the previous version of the current member.

predecessor verification. The process of verifying that changes to the previous version of a member have not been made.

predecessors. Previous versions of a member existing within the same hierarchical view.

primary commands. Editing commands that are entered on the COMMAND line.

primary group. A group 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. An undertaking with prescribed objectives, magnitude, and duration.

project database. A set of logically ordered MVS partitioned data sets (known as libraries) under a single high-level qualifier.

project definition. A project-specific customization of the SCLM product.

project identifier. The high-level qualifier used by all data sets belonging to a particular project.

promotable hierarchy. A subset of a hierarchical view whose top level is the maximum promotable level.

promote. To move (or copy) members and associated internal data up through the hierarchy one level at a time.

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.

software component. An element of the architecture definition, composed of members of one or more data types.

software configuration management. A common point of integration for all planning, oversight, and implementation activities for a project.

sublibrary. A data set that contains Ada intermediate form in a form that only the Ada compiler can use.

syslib. A library containing frozen project level source code.

T

text. Data present in its natural language form (not translatable).

translator. A software program that transforms data from one representation to another.

type. The low-level qualifier of SCLM-controlled data sets, such as SOURCE, OBJECT, and LOAD, that identifies the kinds of data maintained in a specific group.

U

unlock. To make a member (formerly locked out) available for updating (usually associated with promote).

upward dependency. A dependency indicating a compilation unit that must be compiled before the current compilation unit is compiled.

V

VSAM cluster. A named structure consisting of a group of related components.

Index

A

© Copyright IBM Corp. 1989, 1990

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

X

Special Characters

ISPF/PDF Software Configuration and Library Manager (SCLM) Guide and Reference V3 R2 for MVS

SC34-4254-0

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Possible topics for comments are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

_____

_____

_____

_____

What is your occupation? _____

Number of latest Newsletter associated with this publication: _____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the front cover or title page.)

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

Reader's Comment Form

Fold and tape | **Please Do Not Staple** | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

INTERNATIONAL BUSINESS MACHINES CORPORATION
DEPARTMENT T45 INFORMATION DEVELOPMENT
PO BOX 60000
CARY NC 27511

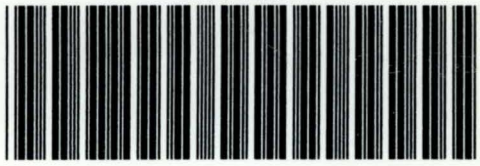
Fold and tape | **Please Do Not Staple** | Fold and tape

IBM®

Program Number
5665-402

File Number
S370/4300-39

SC34-4254-0



Printed in U.S.A.