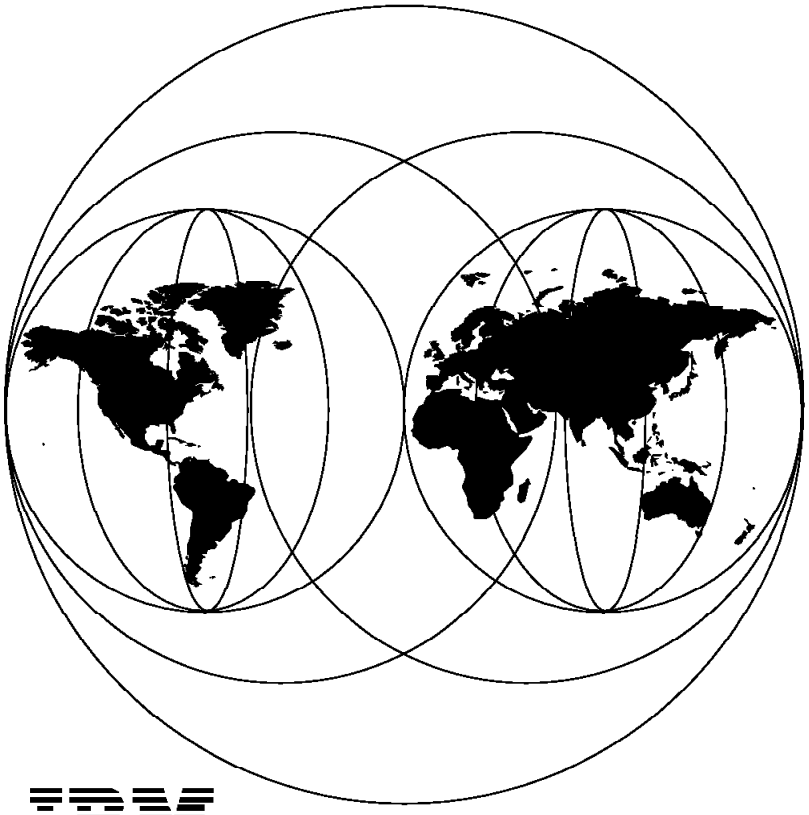
SG24-4843-00

Using ISPF/SCLM
for Automated and Controlled Software Development

October 1996



IBM
International Technical Support Organization
Boeblingen Center

International Technical Support Organization

SG24-4843-00

Using ISPF/SCLM for Automated and Controlled Software Development

October 1996

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix I, “Special Notices” on page 197.

First Edition (October 1996)

This edition applies to Version 4, Release 2 of ISPF/SCLM, program number 5655-042.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. 3222 Building 71032-02
Postfach 1380
71032 Böblingen, Germany

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

Preface

This redbook is intended for management and technical personnel who are responsible for configuration or inventory management of software product development environments. It discusses some of the challenges and problems being faced by many organizations today in software configuration management.

The descriptive part presents an overview of SCLM′s superior capabilities to properly organize the setup and implementation of a complex software development project. It is outlined in detail how the Böblingen Programming Laboratory (BPL), which is responsible for various MVS components and the VSE/ESA operating system, uses the System Configuration and Library Management (SCLM) product to streamline the required processes.

In the appendices diverse examples are provided to illustrate the approach being used by the BPL. Studying and understanding these samples requires some basic knowledge of VSE or MVS.

How This Redbook Is Organized

The redbook is organized as follows:

- Chapter 1, “SCLM - An Overview”

  The chapter provides a short overview of the **S**oftware **C**onfiguration and **L**ibrary **M**anager.

- Chapter 2, “SCLM - Usage by the Böblingen Programming Laboratory”

  This topic covers how SCLM is used to manage the product development in IBM′s Böblingen Laboratory environment.

- Chapter 3, “Concepts and Usage Applied by the Böblingen Programming Laboratory”

  This chapter gives a detailed description of the conceptual and organizational rules implemented by the BPL to efficiently exploit the SCLM product.

- Chapter 4, “SCLM Types”

  Here a brief description of the various SCLM input types as used in the BPL is given.

- Chapter 5, “VSE Related Development Environment Differences”

  Describes the specific rules which have to be followed to develop and integrate VSE code under SCLM.

- Chapter 6, “SCLM Project Characteristics”

  This topic itemizes the SCLM project characteristics as used for the different BPL product development environments.

- Chapter 7, “Development”

  Some highlights are provided about REXX program development and its specific documentation approach.

- Chapter 8, “Processes”

Describes the various development processes being used and automated using SCLM in the BPL environment.

- Appendix A, “SCLM Definitions and Functions for Supported Languages”

  This first appendix presents the language definitions and functions which were used.

- Appendix B, “Translators”

  All translators referred to in appendix A are listed here.

- Appendix C, “Include Library Definitions”

  A summary of all include library definitions is provided in this section.

- Appendix D, “Translator Functions and Utilities”

  Various functions and utilities which were used by the build and promote translators are described in this appendix.

- Appendix E, “VSAM Control File Utility Jobs”

  Lists the detailed VSAM control file utility jobs.

- Appendix F, “VSE Project View under HLQ SCLM2 in Source Edit Format”

  Shows an example of a member in source edit format with BookMaster tags being coded as assembler comments. It is part of an SCLM project definition.

- Appendix G, “Sample Meta Project Documentation (Formatted Output)”

  This appendix demonstrates the formatted output of a sample meta project documentation showing only administration guide information; the implementation section is not shown.

- Appendix H, “Sample Meta Project Implementation”

  A partial view of the sample meta project is given here; by using conditional formatting only the implementation section is shown.

The Team that Wrote this Redbook

This redbook was designed by a group of specialists from the Böblingen Development Laboratory in close cooperation with members of the International Technical Support Organization Böblingen Center. The authors are:

Volker Masen (*IBMNET: DEIBMNUV at IBMMAIL, e-mail:VMASEN@VNET.IBM.COM*), from the BPL Process Engineering, Bldg. B/71032-14, Dept. D/3238, Böblingen, Germany

Johann Pramberger (*IBMNET: DEIBMBUV at IBMMAIL, e-mail:JPRAMBERGER@VNET.IBM.COM*), from the BPL Process Engineering, Bldg. B/71032-14, Dept. D/3238, Böblingen, Germany

Thomas Schwarz (*IBMNET: DEIBMWHV at IBMMAIL, e-mail:TSCHWARZ@VNET.IBM.COM*), from the BPL VSE System Integration, Bldg. B/71032-06, Dept. D/4357, Böblingen, Germany

Comments Welcome

We want our redbooks to be as helpful as possible. Should you have any comments about this or other redbooks, please send us a note at the following address:

redbook@vnet.ibm.com

Your comments are important to us!

Chapter 1. SCLM - An Overview

The IBM Software Configuration and Library Manager (SCLM) is a software complex which aids the development of software applications. It:

- Simplifies and reduces development and maintenance effort
- Controls the movement of application components
- Maintains complete software configuration
- Provides auditability
- Can be customized to an organization's environment.

With the appropriate level of ISPF installed, a customer already has the ability to use SCLM and its facilities; SCLM is available as an integral part of ISPF (from level 3.1).

SCLM has two major functions:

1. Software Configuration

   Using SCLM facilities, it is possible to define how the total application can be built from a number of predefined source libraries. The user can:

   - quantify the impacts of changes to the system
   - verify the completeness of those changes
   - rebuild the system following those changes

   For example, if an include structure is changed currently in a project with no software configurator facilities, all the source libraries should be scanned to see where else that structure is used and what other affected source members need to be rebuilt (recompile or reassemble). In an SCLM environment, this could be under SCLM control, so that when the include structure is changed it is not possible to hand over other source members using that structure without rebuilding them.

   Source modules can be permanently 'tied' to other source modules or generated outputs and SCLM will ensure that when updates are made to these components they will always be promoted together.

2. Library Maintenance

   In conjunction with controlling how the application is built from the source created, SCLM will also control how that source is developed. Once the library structure and development hierarchy have been defined to SCLM, it will control what happens to the new or updated source code during the development.

   For example, when a module is updated for a particular development, SCLM will lock that source member so that any other developer will know that it is being worked on. This helps to avoid developing conflicting copies of the same source or 'losing' pieces of simultaneously developed versions.

1.1 SCLM Concepts and Terminology

This section explains the concepts of and some specific terms associated with SCLM.

The intent of this section is to provide the SCLM information required to understand the use of SCLM throughout our project. This section does not duplicate the information presented in the product manuals. For a more detailed discussion of SCLM, refer to *ISPF/SCLM Project Manager's Guide* and *ISPF/SCLM Developer's Guide*.

1.1.1 Software Configuration

SCLM introduces an object-oriented approach (not to be confused with object-oriented programming) to software development. The objects SCLM can control are members in partitioned data sets. SCLM keeps track of its objects by means of multiple distinct VSAM files containing the accounting information of a member for a specific group.

SCLM can control every object that resides in partitioned data sets.

Through configuration management functions SCLM:

- Controls and enforces the integrity of an object and its associated object modules, load modules and script files containing, for example, the prologue of an associated program source member are examples of associates.
- Provides object versioning capabilities

  Note: Versioning is only implemented for editable members. (Noneditable members can be recreated using the retrieved editable members given that the original compiler/translator is available.)
- Facilitates the accountability of an object and the changes applied to it
- Facilitates a controlled development environment (for example, preventing concurrent updates)

SCLM functions such as promote and build (see below) are, in a way, method implementations. By 'method' we mean applying a predefined action to an object. For example, EDIT is a method of applying changes to an object.

Some of the terms associated with SCLM are defined below.

1.1.2 Library Management

The library management functions of SCLM are intertwined with the configuration management functions.

The movements to and from the various partitioned data sets are examples of library management.

The control over movements to and from the various partitioned data sets are examples of configuration management.

Note: It should be noted that there is no consensus in the literature on the meaning of the terms 'library management' and 'configuration management'.

1.1.3 Project

A project in SCLM is a collection of partitioned data sets with one or more high level qualifiers formed to a common project name. This collection is often referred to as the project database. A project is said to be ′under SCLM control′ if there is a project.PROJDEFS.LOAD data set containing a load module whose name equals the project name. The existence of at least one VSAM file is essential for SCLM to keep (accounting) information about its objects.

1.1.4 Project Definition

The load module project.PROJDEFS.LOAD (project) is referred to as the SCLM project′s main project definition. A project definition represents a view of a project, also referred to as alternate definition. Project definitions are set up using a special set of assembler macros. The project definition has to be assembled using Assembler H and linked into this load module.

1.1.5 Project Hierarchy

SCLM uses hierarchies of groups to control its objects. The hierarchies are defined in the project definition and may conform to an enterprise′s naming convention and Time Sharing Option (TSO) standards.

A group is a collection of partitioned data sets with the same group qualifier.

The last qualifier of the partitioned data sets distinguishes the object types used within a project. (COBOL, LOAD, and COBCOPY are examples of types.)

1.1.6 Language Definition

A language in SCLM is the property of an object (member). Languages have to be defined in the project definition. SCLM keeps track of the language of a member by checking its accounting record. Several language definitions together could be thought of as forming a class within the translate method.

1.1.7 Architecture Definition

SCLM uses Architecture Definition members (ARCHDEFS) to relate dependent modules to each other. There is a special language for Architecture Definition members, describing an application′s configuration.

There are four different Architecture Definitions:

Linkedit Control (LEC)
: Architecture Definition to control the linkedit process

Compilation Control (CC)
: Architecture Definition to control the compile or translate process

High-Level Control (HL)
: Architecture Definition that includes other Architecture Definitions

Generic
: Generic Architecture Definition to control noncompile, nontranslate processors, such as a Program Specification Block (PSB).

1.1.8 Build

To build an object means to apply the member associated language. Thus, to build a member that has COBOL as its language, SCLM compiles the member using the COBOL compiler producing an object module. The SCLM build is ′smart′ enough to know whether a component needs to be reprocessed during a build, thus preventing unnecessary processing. (Many organizations build on Architecture Definitions only.)

1.1.9 Promote

Promote is the SCLM term for moving an object to the next higher level in the hierarchy. Usually, during a promote the object′s occurrence at the lower level is purged. If a member is promoted, all the associated members referred during a build are promoted as well (physical promote).

1.1.10 Scope of Processing

Understanding how SCLM determines its processing scope is important. If the processing scope is too narrow, ′out of sync′ situations can occur.

During development, applications can become out of sync because several individual developers are working with one application. At some point during the development life cycle, all applications should be in sync.

To enforce synchronization, an HL Architecture Definition spanning all applications should be developed. This definition should be used to build and promote all changes into the group level used for the system test and then into the group level used for production. (The promote will fail if there is an out of sync situation, thus preventing the propagation of this situation to a higher level in the hierarchy.)

The scope of processing is determined by the scope of the Architecture Definition that is passed to such SCLM functions as build and promote.

Note: It is possible to build and promote an individual object by passing the object to the various SCLM functions. Because language definitions exist, processing can be done without having Architecture Definitions that define or reference the object.

1.1.11 Versioning

With ISPF Version 3.3, versioning was introduced for SCLM. Versioning can be specified for all editable types on all levels in the hierarchy. In addition to a copy of the last version of a member, the ′versioning PDS′ for a versioning-enabled type holds incremental changes called ′deltas′. A member′s version can be recreated by starting with the last version and applying the appropriate deltas to that version. (Version -1 is the last version with last delta applied to it; version -2 is the last version with last delta and delta -1 applied to it, and so forth.) With versioning, when a member is promoted, all the associated members are promoted as well. An ISPF panel interface is available to retrieve the various versions.

1.2 Component Changes of SCLM 4.1 and 4.2

The ISPF SCLM component contains the following new functions and enhancements:

- SCLM advanced extents support

  The include search capabilities of SCLM have been enhanced to allow more than two types to be searched when finding parsed include dependencies. The types to be searched are determined by the language of the object being processed rather than the type of the object.

  Allocation of FLMSYSLB data sets can be done automatically by Build using the ALCSYSLB parameter on the FLMLANGL macro. This removes the need to specify the FLMSYSLB data sets using the FLMCPYLB macro.

  SCLM can now check for includes in FLMSYSLB data sets at Parse or Build time. Doing the check at Build time allows for movement of the includes in and out of the FLMSYSLB data sets without having to reparse all objects that reference the includes.

- SCLM enhanced member list

  Option 3.1 of SCLM has been enhanced to support View, Edit and Build through member list. The member list can be for a single group or *the entire hierarchy view*. The objects on the member list can be selected and ranked by the availability of their accounting and build map record.

- SCLM multiple output support

  Build has been enhanced to support translators that generate multiple output objects in a partitioned data set.

  The architecture language has been enhance for LEC archdefs to support referencing output keywords other than OBJ and LOAD. LEC archdefs can also include SINC statements to include output that is under SCLM control, but generated outside SCLM.

- SCLM workstation edit and view

  Allow the usage of workstation tools to create, modify and view SCLM controlled members.

- SCLM workstation build for OS/2 and Windows

  Permits the invocation of workstation tools against SCLM controlled members during the BUILD process. Sample language definitions, parsers and SCLM control files are provided to help the user get started in building C++ workstation applications using SCLM.

- Build and promote by change code

  Allows to build and promote only those parts associated with a specific change or group changes. This is accomplished by including one or more CCODE statements in the architecture definition to be build or promoted.

- Enhanced support for versioning

  New parameters to improve versioning, auditing and servicing of members and member groups.

- Additional SCLM service interfaces

  Improved handling of member accounting information such as retrieving, searching and deleting data.

- Options by group

  Provides the capability of having different options for Build translators for a group of SCLM-controlled members.

- Conditional processing of Build translators by group

  Allows to specify by group which Build translators to process and which to stop. One can also refine this by examining return codes from previous Build translators in the language definition much like conditional processing of a step in a batch job.

- Additional language sample definitions

  Additional samples in various languages are provided to assist in the definition and usage of important SCLM functions.

Chapter 2. SCLM - Usage by the Böblingen Programming Laboratory

2.1 Introduction

The Böblingen Programming Laboratory (BPL) is a part of IBM Deutschland Entwicklung GmbH and has responsibilities for /390 system product development such as VSE/ESA, MVS System Management products such as RMF and MVS base products such as TSO, HCD and other /390 Client/Server Business related products. Our main development system has always been MVS, although development on VM has been done extensively in the past.

This document describes how we use SCLM to manage our product development and we hope to provide enough detail so that the reader may see how SCLM as a generic library and build system can be used in a real life environment and customized to fulfill different needs. It should give the reader a feeling of what kinds of environment and processes can be covered using SCLM.

We cannot hope to cover all the related subjects in a great deal of detail in this document. Our emphasis first and foremost is on our complete product development environment and the use of SCLM instead of showing the implementation details which would be too much for the scope of this book. However, if the reader feels that he could benefit from our experience and environment setup and needs more information he is welcome to contact one of the authors listed below. We are more than willing to provide guidance and technical background and implementations if desired. Please also refer to the appropriate documents in the bibliography.

In the following chapters will address the questions:

As already mentioned: we have not included all of the source code and definitions such as REXX programs or SCLM definitions, that we used in our environment because of their size. However, if the reader is interested in the source he may contact one of the authors:

- Volker Masen, *IBMNET: DEIBMNUV at IBMMAIL, e-mail:VMASEN@VNET.IBM.COM*
- Johann Pramberger, *IBMNET: DEIBMBUV at IBMMAIL, e-mail: JPRAMBERGER@VNET.IBM.COM*
- Thomas Schwarz, *IBMNET: DEIBMWHV at IBMMAIL, e-mail: TSCHWARZ@VNET.IBM.COM*

2.2 Reasons to Choose SCLM

2.2.1 Development Process Prior to SCLM

Prior to installing SCLM in our environment we used a tool called CLEAR which was developed and updated as required internally over a period of almost 20 years. Obviously, such a product contains a myriad of stop-gap solutions and deficiencies. The library system was optimized for one standard development process with the focus on a central product driver build.

The process can be described briefly so: The developers use CLEAR only to store their source data and an integration team then creates complete product drivers at certain development cutoff dates. Those are tested by an independent component or system test group. The final driver produced is then the product delivery tape(s) to be shipped to the customer.

The library system CLEAR was not designed to allow the creation of parallel development drivers at any state of development of any developer. It did not support different development models, such as prototyping or iterative development. It was traditionally based on a pure waterfall development process to store the program sources. It was essentially needed for the coding and component and system testing phases.

The format to store parts in CLEAR was a proprietary delta format and therefore only accessible via special tools offered with the CLEAR library. Also the interface to CLEAR was a proprietary language and actions were done by JCL generation and batch job submission.

In practice the developers use the library system as the common storage for the product source, checking out the parts on which they are working. These are kept, with the dependency sources, in private partition data sets. Because of this the processes to create the output for development tests such as compiles, were also handled separately from the CLEAR driver build processes. The developers used and maintained private data sets, CLISTs, JCL, REXX programs and so on.

This technique has the tendency to create problems when parts were checked back in to the library system and then processed there with the CLEAR defined processes. It was therefore not guaranteed that processes and even source code used by the developers outside CLEAR were the same as those defined or stored in CLEAR.

Two cases are outlined below where problems may typically arise due to the fact that source code is checked out of the library and treated outside library control in private data sets.

- To work on source code, a developer has to check it out of the CLEAR library and put it into a private data set. This causes problems if other users rely on this source and something goes wrong. For example, there is no control to ensure that the developer checks the right code back into CLEAR, or if he forgets to do it at all.

  It can also happen that a developer needs to lock his data set with the source code for longer periods of time to ensure that only he does development on this member. Naturally, this can lead to disastrous results in the build process which relies on the correct code being in the CLEAR library.

  These problems can be avoided by having the group hierarchy in SCLM, where the developer never has to have private libraries and all is done within one library. You are allowed to view any code currently managed in any state of the complete development cycle because all code is stored in SCLM and no longer in private data sets. Also the SCLM database report can be used to find parts in the different stages of development.

- Several developers are working on the same source in parallel with the possible chance of overwriting each other's work. The same was true with the include dependencies. If one user has the member locked, another user can check out the source in read mode to work on. As there are no controls over this, it is possible that each assumes he is the only one who worked on the member and replaces it only with his changes.

  SCLM as described below provides some restrictions and reminders by using the access and authorization codes for members and groups.

Due to these potential problems, a lot of manual control and rework was necessary to avoid or clear up discrepancies at CLEAR driver build time.

2.2.2 Today

With SCLM, those problem areas are minimized due to the fact that SCLM does not work with the check-out, check-in technique explicitly (even though it is available as functions to connect edit tools to SCLM). It integrates the parallel editing of sources in its concept, so that the developers do not have to leave the library control at any time. This enables even parallel updates by two developers on the same original source. SCLM allows this by using different access codes where only one is allowed to overwrite the original. So a developer editing a source with a ″temporary″ access code is aware that before he changes to the ″update″ access code he must do a rebase to integrate his changes to the code that has been modified using for example the edit compare function with ISPF 4.2.

2.2.3 Where we Started

Our first experiences with SCLM 3.5 in 1993 as a product release development library were to cover SPE (Small Product Enhancements) which were too small for CLEAR and too large for our service development library (see 8.4, “Service Processes” on page 93).

Based on this experience, in 1994 we started a re-engineering effort initiated by management to look in detail at our development process and environment to:

- Minimize costs
- Reduce redundant work
- Standardize processes
- Have easy to maintain tools
- Address iterative development
- Reduce cycle time
- Reduce error-prone areas
- Have easy to use tools
- Use standard available tools

With this in mind a task force was established at the end of 1994 to do a feasibility study for all our MVS and VSE products to replace the CLEAR, VM and private development environments with SCLM within one year. The task force included the SCLM support team to provide their expertise and the integration teams from MVS and VSE products to discover and define all the critical processes needed to be covered in an SCLM only environment. We also investigated if CLEAR could be used in the same way as SCLM to extend its scope using the library system much earlier (starting with system design) and in a more flexible way.

In comparison to CLEAR we found out that SCLM has one library system right from the system design level stage that allows the developers to work (edit, compile, debug) on their level almost as they had done in the past outside CLEAR. It also enables the integrators to do their driver builds using SCLM as in the CLEAR system previously. It gave us the potential to do the documentation development and other parts maintenance in the same library setup which was not feasible with CLEAR.

With SCLM, the developer never has to leave the control of the library system. He does the editing of input and browsing of output using the same data formats and available tools he was used to because data is stored in PDS or PDSEs. Because of this he can immediately do his testing after a build by concatenating the relevant data sets in the SCLM hierarchy. There is no packaging step or check-outs and check-ins. The benefits of this are obvious, especially as the source code and the output produced are maintained together.

Another benefit is that the definition languages to customize SCLM are standard languages such as Assembler, REXX or C. These are easier to learn when compared to the proprietary languages provided by CLEAR.

The feasibility study recommended that the switch over could be done in one year for all our MVS products and the complete VSE operating system. By that time we had seen that SCLM allows us to do everything we did with CLEAR and more, much more efficiently.

With the move to SCLM we achieved the principles listed in the following box.

Principles of our Development Environment Support

- **one** central tool and user interface **for all**
- **all** processes defined and maintained at a **central place**
- **save costs** because SCLM comes **free with ISPF**, where we had to provide funding for the development and maintenance of the internal CLEAR library
- **defined support** structure for the library system available because we shift to official IBM product where we have the **official support channels** provided (Our customers benefit as we now use the same product as they)
- **lower skill needed** to support the environment setup because only **standard languages** are needed

It was very important that the integration teams and management commit themselves to this project and that the feasibility study was done by the people who afterwards had to do the implementation.

Nevertheless, going from CLEAR to SCLM we lost the packaging feature that was hooked into CLEAR to create SMPE formatted tapes. Therefore, we had to implement a strategy to migrate from CLEAR to SCLM smoothly.

2.2.4 Migration to SCLM

We started by putting all the developers for one product on SCLM but kept CLEAR in the background as a shadow library that checked-in members automatically during a promote from one SCLM group to another. We still had CLEAR and the tape build process working as it was. This is described in 8.3.3, “CLEAR as Shadow Library, SCLM as Master” on page 91.

This means that each SCLM group represented then the CLEAR contents. With this approach we took the processes away from the developers and were able to map the CLEAR processes to the SCLM project and keep both synchronized. The benefit was that we could gain experience with SCLM but still have the CLEAR system and processes working as they were.

When we proved that SCLM could cover all the development build processes provided in CLEAR we only had to move the packaging process from CLEAR to SCLM and then drop the CLEAR system. This approach is described in more detail in 8.3.3, “CLEAR as Shadow Library, SCLM as Master” on page 91 as a demonstration for a possible migration step and the move of the packaging process is described in 8.2.2, “Product Packaging” on page 85

2.3 Conclusions

As we expected we did the migration to SCLM in about one year for all the products and even put a new product on SCLM in two months at the same time. There were two support people involved plus one integrator for each of the MVS products and two people for the VSE operating system. Integrators did the migration in parallel to the ongoing release integration work.

Basically our experience is that once you start using SCLM, understanding it is just a matter of doing and reusing what you already know. So the outcome, based upon our experiences, can be summarized as follows:

Principles when using SCLM

- **Start slowly**, this allows you to act faster and more efficiently later on due to the reuse effect
- Take your time to **understand the SCLM capabilities**
- Take your time to **understand your environment**
- Take your time to **understand your processes**
- Stay with the **standard SCLM user interface** provided as far as possible.

  Do not create new dialogs. Put all functions via a language and build and promote sequences into SCLM. With this approach we have a simple environment which handles everything in almost the same way. The complexity is hidden from the user and put in all the exits and translator functions.
- Have an **SCLM support team independent** to the product development and integration teams to provide the basic setup and do consulting across products
- If something seems as though it cannot be done with SCLM, our experience shows it is very likely that:
  - either we found a problem in our process that cannot be automated or is hard to manage
  - or we probably did not fully understand the capabilities of SCLM in this area. Then it is worth restudying the SCLM functions or get advice from experienced SCLM users (consultants)
  - or, in very rare cases, we found some deficiency in the SCLM mechanism, needing additional efforts to be handled

There are obviously still things that can be improved in SCLM to make processing easier, but we see no real obstacles from an SCLM design point of view.

Chapter 3. Concepts and Usage Applied by the Böblingen Programming Laboratory

3.1 BPL Development Environment to Manage Products

One goal was to have a common process for all /390 products we develop, across teams and the complete development cycle.

Overall our development environment can be generally described as shown in Figure 1 on page 15.

The environment can be grouped into three logical areas:

Release development environment

This covers such things as:

- Release Development
  - Create new parts and modify old parts
  - Create development drivers
  - Fix problems found during the release development cycle
  - Integrate field problems (APAR fixes, PTFs) into the release development cycle. For details see 8.4.1, “Interface to Service Library” on page 93 and 8.1.1, “Rebase of Service to Current Development” on page 73.
- Product Packaging. For more details see 8.2.2, “Product Packaging” on page 85.
- Unit and Function test

Test and Production environment

This covers the areas:

- Component test
- System test
- Production environments at customers′ sites

This area is not described here in detail, because typically the packaged product produced in the release development is installed using MSHP formatted tapes for VSE or SMPE formatted tapes for MVS products and has therefore no relation to SCLM.

One exception is if special test systems are needed for development tests. Then, instead of a tape creation and install:

- a technique is used as described in 8.3.2, “SHIPIT - Externalize Data during Promote Time” on page 88
- or the two systems, the development system and the test system share data sets. The test system has read access to some of the SCLM controlled data sets of the group hierarchy that are needed for the product test.

This means, that after builds, the developers have only to log on to the test system or do a refresh on the test system to use the data set according to the concatenation needed for the test.

Service development environment

This area covers the correction of problems in the field:

- to create APAR fixes
- to create PTFs

Details about this environment are given in 8.4, “Service Processes” on page 93.

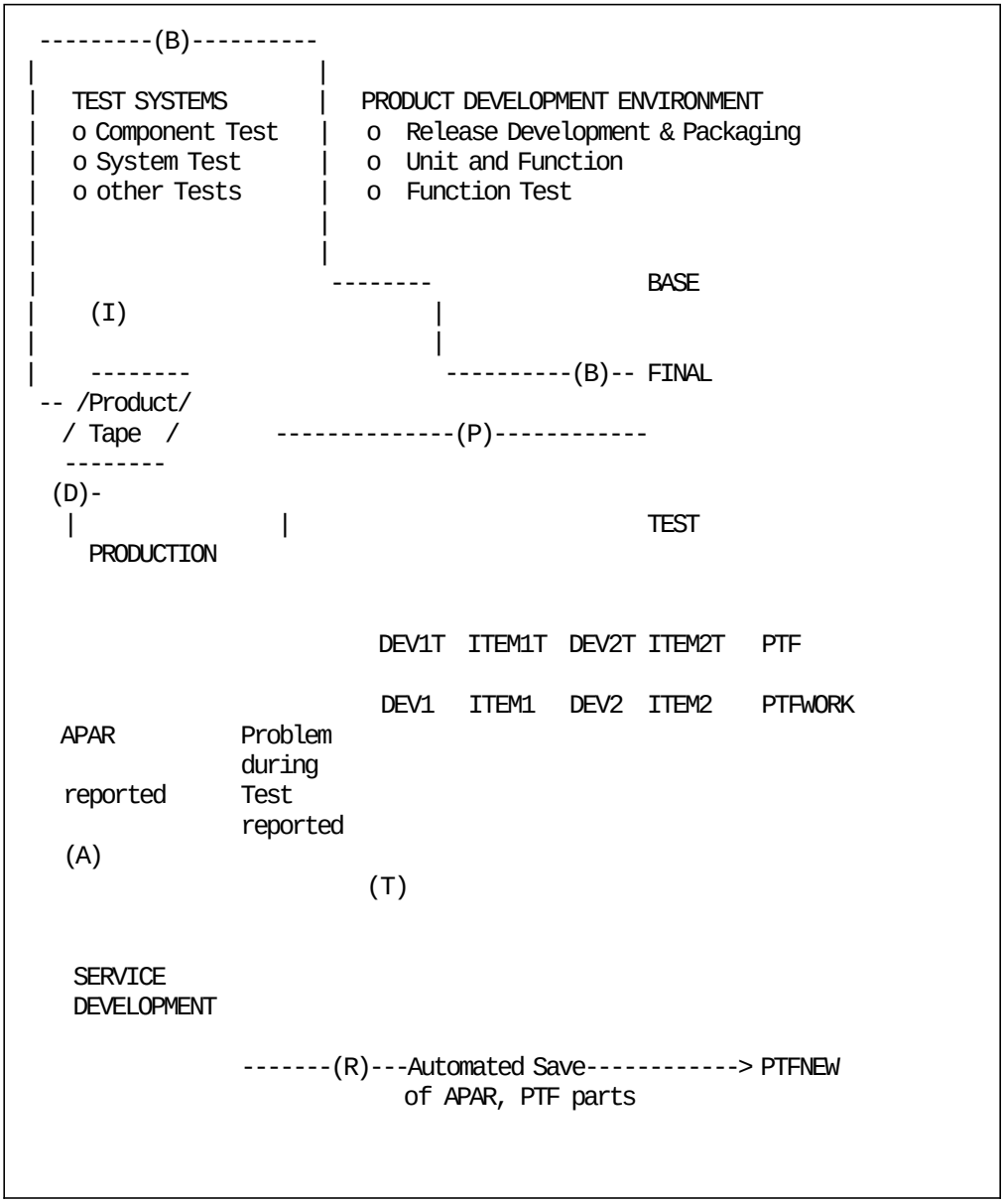


Figure 1. Overview of BPL Product Development Environments.

(B) *Create product tapes from SCLM group hierarchy. See 8.2, "Packaging Processes" on page 76*

(I) *Install product tape*

(D) *Deliver product tape to customer*

(P) *Update external test systems during promote. See 8.3, "Delivery Processes" on page 88*

(A) *Interface to RETAIN to track field problems (APARs)*

(T) *Track problems during release development*

(R) *Provide parts changed due to field problems to reintegrate those changes into release development. See 8.1.1, "Rebase of Service to Current Development" on page 73*

3.1.1 VSE and MVS

The reason we mention VSE and MVS as separate environments in this document is that only some of the processes for VSE can be run on a VSE system whereas all the processes needed for the MVS products can be run on MVS. Because SCLM does not support builds on a VSE system (not even with version 4.2) we had to invent our own that works in a similar way to our old environment with CLEAR. For more details about the VSE environment see Chapter 5, “VSE Related Development Environment Differences” on page 39. However, the overall concepts, apart from the technical implementation, are the same for the MVS and VSE developer or integrator. Also both teams use the same library system and the same user interface.

VSE development environment

Historically we did the VSE development using MVS and VM systems to do the compiles. The links and the packaging were then done on VSE issuing batch jobs to VSE systems.

To use SCLM for the VSE environment we had to create a solution where we initiate batch jobs during the builds to either catalog parts on a target VSE system (such as objects, shippable sources) or do some work on the target VSE system and synchronize the outcome of the batch jobs with the initiated SCLM build. For more details see Chapter 5, “VSE Related Development Environment Differences” on page 39.

MVS development environment

For our MVS products we used MVS not only as the development system but also to do the packaging. For more details see 8.2, “Packaging Processes” on page 76.

Areas common to all SCLM environments

- 3.2, “Roles Needed to Manage Projects” on page 17
- 3.3, “Introduction of SCLM to New Products” on page 28
- Translator macros. See Appendix B, “Translators” on page 107
- Process techniques for:
  - NLS development. See 8.1.2, “NLS Part Development” on page 74
  - Rebase - Retrofitting code coming from service which is developed at the same time in a release. See 8.1.1, “Rebase of Service to Current Development” on page 73
  - Meta SCLM projects. See 6.1, “Types of SCLM Projects” on page 55 and 6.2, “Meta SCLM Projects to Develop SCLM Projects” on page 59
- Migration tools to:
  - Generate CC architecture definition members from models. See 3.6, “Product Migration or Load Product into SCLM” on page 31 and D.2, “MAKEARCD - Generate Architecture Definitions from Models” on page 126.
  - Generate LEC architecture definition members from LINKJCL. See D.3, “LNK2ARCD - Generate Architecture Definitions from LINK JCL” on page 132.
- 8.3, “Delivery Processes” on page 88

- Service development. See 8.4, “Service Processes” on page 93

For several years we have also provided code for the Client/Server features of some of our products, so that we also have to develop on and for workstation systems such as OS/2, and AIX.

Before ISPF 4.2, the development for such areas was done on the workstation only and we used library systems such as PVCS and CMVC. For the driver build we used the available MAKE utilities, because both library systems do not have integrated support of any driver build function; they only provide source data management with a check-out, check-in technique. To package workstation and /390 code was again a separate process using tools such as Software Installer.

Now with the workstation build supported by SCLM with 4.2 we can use SCLM as the first library of choice for those /390 products that also have workstation support. This gives us one single library for the complete product and automates the packaging as an integral part of the /390 packaging. This document, however, does not address the workstation build of SCLM 4.2 as, at the time of writing, our experience with it was limited.

3.2 Roles Needed to Manage Projects

3.2.1 The BPL SCLM Team

The BPL SCLM team as a whole is responsible for the implementation of an SCLM Development Environment for all projects in BPL using SCLM as the library system. The team relies on an MVS installed system and an MVS system support group to provide the basic operating system and product installations.

The BPL SCLM team is divided into three teams with different scopes. This section describes the responsibilities and skill needed by each of the teams. How these then interact is described in detail in 3.3, “Introduction of SCLM to New Products” on page 28 and context related in other chapters.

3.2.2 SCLM Support Team

SCLM Support Team Responsibilities

BPL-product Related Activities

Consulting

- share experiences across products.
- advise the product teams how their development environment can be realized with SCLM.
- advise the product teams how their processes can be integrated and automated in an SCLM development environment as a single point of control.

Environment Setup

- provide an SCLM Meta project to create the SCLM development environment. See 6.2, “Meta SCLM Projects to Develop SCLM Projects” on page 59.
- support the integration people doing the setup or do the initial implementation of the SCLM project setup to get them started.

- provide documentation aids. See 7.2, “Documentation Development Environment” on page 70 and D.5, “REXX2LLD - Generate BookMaster Format from Sources” on page 137.
- provide setup tools and SCLM language translators. For example D.6, “SCLMINFO - Generate Data Set Allocation Information” on page 143, D.7, “SALLOC - Allocate Data Sets” on page 147, D.9, “SCLMSITE - Basic SCLM Project Environment Setup” on page 151 and Appendix B, “Translators” on page 107.
- provide migration tools such as those in D.2, “MAKEARCD - Generate Architecture Definitions from Models” on page 126 and D.3, “LNK2ARCD - Generate Architecture Definitions from LINK JCL” on page 132.
- implement and integrate product related processes in an SCLM development environment if they are used or have a potential to be used in at least two projects or are critical for them. For example see 8.1, “Development Processes” on page 73. For build process see Appendix A, “SCLM Definitions and Functions for Supported Languages” on page 95 and for packaging processes see 8.2, “Packaging Processes” on page 76.

Education

- educate the integration team to maintain the project setups themselves.
- educate the developers to use the SCLM environment as a vehicle to increase their productivity.

Process Automation

Automation of activities needed in all products, such as:

- common build services (for example, OCO-Scanner)
- Rebase process (see 8.1.1, “Rebase of Service to Current Development” on page 73)
- NLS support (see 8.1.2, “NLS Part Development” on page 74)

SCLM Environment Activities

- maintain and provide a setup on which the different SCLM project setups will rely (see 6.1, “Types of SCLM Projects” on page 55)
- provide exemplary SCLM languages (see Appendix A, “SCLM Definitions and Functions for Supported Languages” on page 95)
- provide commonly used SCLM translator macros (see Appendix B, “Translators” on page 107)
- provide product source migration aids (see 3.5, “SCLM Implementation” on page 30 and 3.6, “Product Migration or Load Product into SCLM” on page 31)
- prepare and test SCLM version upgrades
- prepare and test SCLM workstation environment
- provide environment documentation technique and guidelines

MVS Product Packaging Automation Support

Implement MVS packaging processes (see 8.2, “Packaging Processes” on page 76) to create:

- SMPE installable tapes
- VPL-listings
- Optional material from SCLM development environment

Service related Automation Support

Provide tools to transfer code from an SCLM library to a service library and on the other site to provide changed code from a service library into the SCLM library (see 8.4, “Service Processes” on page 93)

End of SCLM Support Team Responsibilities

The following skills are required to handle the previously described tasks:

SCLM Related Skills

- Library concepts, especially SCLM concepts and implementing
- Programming skill
  - REXX language, other language skill not really needed but helpful
  - almost no assembler skill; enough to write some small assembler macros to establish SCLM definitions
- Environment process skill
  - understand and interpret integration, development and service processes
- Some knowledge of the MVS environment

3.2.3 Project Integration Team

The following tasks have to be handled by the project integration team:

Project Integration Team Responsibilities

SCLM Project Environment Activities

- Lead product analysis if products are migrated to SCLM (for example, define languages, define group hierarchy, define names of types) as described in 3.3, “Introduction of SCLM to New Products” on page 28
- Implement and adapt SCLM project environment in cooperation with the development team as described in 3.5, “SCLM Implementation” on page 30
- Implement and automate project specific processes. Processes which are needed by more than one project are implemented by the SCLM support team
- Educate the development team
- Support the development team in case of problems

Project Integration Activities

- Verify build and packaging information stored in architecture definition members such as output types, compile options, link options, packaging parameters
- Execute Build and Packaging activities
- Control Build and Packaging processes

Administration Activities

- Administrate RACF authorization on all the groups
- Administrate the project data (compress or reorganize data sets, if necessary, issue additional backup procedures, if needed)

End of Project Integration Team Responsibilities

The following skills are required to handle these tasks:

SCLM Related Skills

- basic SCLM skill
- understand the basic SCLM macros (FLMABEG, FLMCNTRL, FLMATLC, FLMAGRP, FLMGROUP, FLMTYPE, FLMATVER, FLMLANGL, FLMINCLS, FLMSYSLB, FLMAEND)
- understand the architecture member definition concept of SCLM
- understand development and packaging processes
- basic programming skill to understand compile options, link options and so on
- some knowledge of the MVS environment

3.2.4 Project Development Team

Following is a detailed break-down of the individual responsibilities within the development team.

Project Development Team Responsibilities

SCLM Activities

- Develop source code under SCLM control
- Create or adapt relevant architecture definition members, containing information about output libraries, compile options, link options, packaging parameters and so on
- Create architecture definition members (WORKDEF members) defining work-lists and integration lists used in the next project driver. For more detail see 4.3, “Types of Metadata” on page 36
- Responsible for an error-free build (compile, formatting and so on) of all members used in the next product driver. This is actually enforced by SCLM because it promotes only output with correct builds.

End of Project Development Team Responsibilities

Besides product related know-how the following skills are required:

SCLM Related Skills

- be familiar with SCLM user interface, especially option 3.1.
- basic understanding of the architecture definition concept of SCLM
- some knowledge of the MVS environment

3.2.5 Methods to Implement Team Responsibilities

There are several methods to implement team responsibilities as they are described in 3.2.1, “The BPL SCLM Team” on page 17. In our SCLM environment, the different responsibilities are reflected in the methods described in the following chapters. The examples refer to the following exemplary SCLM project.

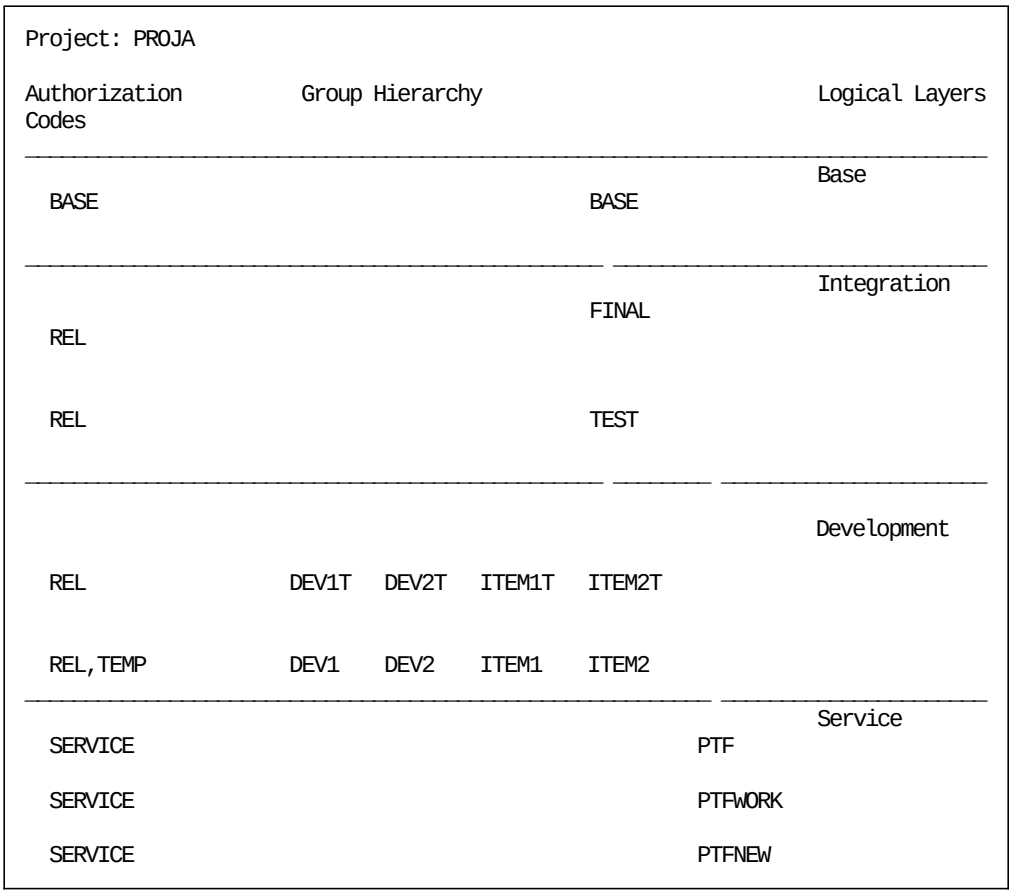


Figure 2. Sample of a BPL Project Hierarchy

Below follows a short description of the groups defined in Figure 2.

DEVx

groups where a single developer develops his code, such as modules and panels. On this group, the developer also defines his metadata members (for example, WORKDEF members, see 4.3, “Types of Metadata” on page 36) which describe the input for the next driver. If the source is ready for the next driver, the developer builds the WORKDEF member and promotes the related parts to the next group. The names of the WORKDEF members that the developers have to use for integration are defined for each driver by the integration team. The member also contains, as comments, information about the parts to be integrated and any restrictions for the testers.

ITEMx

groups where one or more users commonly develop a line item of a project. As described above for groups DEVx, the developers maintain a WORKDEF member, which is built and promoted to the next group if the source is ready for the next driver. The basic difference to DEVx groups is that not just one person has RACF update authority on this group but as many as are working on a line item.

DEVxT, ITEMxT

promote groups, which contain all parts used for the next driver. The project integration team promotes the WORKDEF members from these groups at cutoff time. The names of the members to be promoted are defined in advance by the integration team. Therefore, they rely on the

contents provided by the developer. If architecture members are changed or new, the integration team will check if they comply to the standard and the needed packaging format.

TEST

First integration group. Using this group, the project integration team does a rebuild on the whole product defined by a PRODDEF member (see 4.3, “Types of Metadata” on page 36) to resolve the dependencies across parts coming from the different development groups. The PRODDEF member is an HL architecture member describing a tree of all source parts making up the complete product. The result of this build is a consolidated product driver which now can be used to test the whole product in SCLM without a packaging step.

Sometimes this step and group are owned and managed by one developer to create development drivers.

FINAL

If the product test on group TEST was successful, the HL architecture member in type PRODDEF describing the whole project is promoted. So all parts referred to by this HL member are promoted to group FINAL. For this group packaging is done to create the product delivery format. This is described in 8.2, “Packaging Processes” on page 76. At the end of a development cycle, the group FINAL contains all parts which are changed or developed for the corresponding project release.

BASE

The group BASE contains the BASE release. It is loaded once and then frozen during the whole development cycle using RACF and SCLM authorization codes.

PTFx

These groups are used to integrate parts changed by the service people in parallel into the ongoing release development. For further information on these groups, see 8.1.1, “Rebase of Service to Current Development” on page 73.

3.2.5.1 RACF Protection

The ownership of the groups related to the roles is reflected in the following RACF access groups, declared for every project. Each RACF access group has different access rights to the data set profiles (see Table 1 on page 23).

<proj>DEV

For example, the access group PROJADEV contains the user IDs of the project development team members of project PROJA.

<proj>INT

For example, the access group PROJAINT contains the user IDs of the project integration team members of project PROJA.

<proj>ADM

For example, the access group PROJAADM contains the user IDs of the SCLM support team members and the user IDs of those people in the project integration team, responsible for the data administration. The user IDs of this access group are able to allocate, delete, reorganize whole project data sets. Therefore, it is recommended that this group only contains a few user IDs.

Below is an example of how the data sets of the different SCLM groups for project PROJA are protected from unauthorized access. For your information, the data sets 'PROJA.PROJDEFS.*' contain all data (for example, REXX execs) needed to execute the different processes during the project development. The data sets 'PROJA.PROJDEF$.*' reflect the SCLM control data sets (for example, accounting data sets).

Table 1. RACF Protection of SCLM Project Data Sets

Data set profile	**RACF access groups**			
	List of one or more users	**PROJADEV**	**PROJAINT**	**PROJAADM**
PROJA.BASE.*	READ	READ	READ	ALTER
PROJA.FINAL.*	READ	READ	UPDATE	ALTER
PROJA.TEST.*	READ	READ	UPDATE	ALTER
PROJA.<dev_group>T.*	UPDATE	READ	UPDATE	ALTER
PROJA.<dev_group>.*	UPDATE	READ	READ	ALTER
PROJA.PROJDEFS.*	READ	READ	UPDATE	ALTER
PROJA.PROJDEF$.*	UPDATE	UPDATE	UPDATE	ALTER

To establish the described RACF layers for the projects, the project administrator maintains a RACF job containing all relevant RACF definition statements.

```
//BVOLRACF JOB (0007,,),BVOL,USER=BVOL,
//          MSGLEVEL=(1,1),MSGCLASS=H,NOTIFY=BVOL
//*-------------------------------------------------------------
//*
//*        RACF JCL to protect SCLM data sets
//*        HLQ    : PROJA
//*
//*
//*  To use the CONNECT command you must have the SPECIAL GROUP
//*  attribute in the GROUP(xyz)
//*
//*  Use ADDSD statement instead of ALTDSD to add new data set
//*  profiles
//*
//*-------------------------------------------------------------
//RACF     EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD   SYSOUT=*
//SYSTSIN  DD   *
```

Figure 3 (Part 1 of 3). Sample RACF Job

```
/* User types ------------------------------------------------------*/

CONNECT (DEV1 DEV2 DEV3                                  ) GROUP(PROJADEV)
CONNECT (INT1 INT2                                       ) GROUP(PROJAINT)
CONNECT (ADM1                                            ) GROUP(PROJAADM)

LISTGRP  PROJADEV
LISTGRP  PROJAINT
LISTGRP  PROJAADM

/* Global authorization--------------------------------------------*/
ALTDSD ¢PROJA.**¢                        UACC(NONE)     OWNER(DE#00007)
PERMIT ¢PROJA.**¢                        ACCESS(READ)   ID(PROJADEV)
PERMIT ¢PROJA.**¢                        ACCESS(UPDATE) ID(PROJAINT)
PERMIT ¢PROJA.**¢                        ACCESS(ALTER)  ID(PROJAADM)

/* Project control data sets ---------------------------------------*/
ALTDSD ¢PROJA.PROJDEF$.*¢                UACC(NONE)     OWNER(DE#00007)
PERMIT ¢PROJA.PROJDEF$.*¢                ACCESS(UPDATE) ID(PROJADEV)
PERMIT ¢PROJA.PROJDEF$.*¢                ACCESS(UPDATE) ID(PROJAINT)
PERMIT ¢PROJA.PROJDEF$.*¢                ACCESS(ALTER)  ID(PROJAADM)

/* Data Sets containing data need for the build processes ----------*/
ALTDSD ¢PROJA.PROJDEFS.*¢                UACC(NONE)     OWNER(DE#00007)
PERMIT ¢PROJA.PROJDEFS.*¢                ACCESS(READ)   ID(PROJADEV)
PERMIT ¢PROJA.PROJDEFS.*¢                ACCESS(UPDATE) ID(PROJAINT)
PERMIT ¢PROJA.PROJDEFS.*¢                ACCESS(ALTER)  ID(PROJAADM)

/* Base data authorization -----------------------------------------*/
ALTDSD ¢PROJA.BASE.*¢                    UACC(NONE)     OWNER(DE#00007)
PERMIT ¢PROJA.BASE.*¢                    ACCESS(READ)   ID(PROJADEV)
PERMIT ¢PROJA.BASE.*¢                    ACCESS(READ)   ID(PROJAINT)
PERMIT ¢PROJA.BASE.*¢                    ACCESS(ALTER)  ID(PROJAADM)

/* Integration data authorization----------------------------------*/
ALTDSD ¢PROJA.FINAL.*¢                   UACC(NONE)     OWNER(DE#00007)
PERMIT ¢PROJA.FINAL.*¢                   ACCESS(READ)   ID(PROJADEV)
PERMIT ¢PROJA.FINAL.*¢                   ACCESS(UPDATE) ID(PROJAINT)
PERMIT ¢PROJA.FINAL.*¢                   ACCESS(ALTER)  ID(PROJAADM)

ALTDSD ¢PROJA.TEST.*¢                    UACC(NONE)     OWNER(DE#00007)
PERMIT ¢PROJA.TEST.*¢                    ACCESS(READ)   ID(PROJADEV)
PERMIT ¢PROJA.TEST.*¢                    ACCESS(UPDATE) ID(PROJAINT)
PERMIT ¢PROJA.TEST.*¢                    ACCESS(ALTER)  ID(PROJAADM)

/* Developer data authorization ------------------------------------*/
ALTDSD ¢PROJA.DEV1.*¢                    UACC(NONE)     OWNER(DE#00007)
PERMIT ¢PROJA.DEV1.*¢                    ACCESS(UPDATE) ID(DEV1)
PERMIT ¢PROJA.DEV1.*¢                    ACCESS(READ)   ID(PROJADEV)
PERMIT ¢PROJA.DEV1.*¢                    ACCESS(READ)   ID(PROJAINT)
PERMIT ¢PROJA.DEV1.*¢                    ACCESS(ALTER)  ID(PROJAADM)
```

Figure 3 (Part 2 of 3). Sample RACF Job

```
 ALTDSD ¢PROJA.DEV1T.*¢                    UACC(NONE)     OWNER(DE#00007)
 PERMIT ¢PROJA.DEV1T.*¢                    ACCESS(UPDATE) ID(DEV1)
 PERMIT ¢PROJA.DEV1T.*¢                    ACCESS(READ)   ID(PROJADEV)
 PERMIT ¢PROJA.DEV1T.*¢                    ACCESS(UPDATE) ID(PROJAINT)
 PERMIT ¢PROJA.DEV1T.*¢                    ACCESS(ALTER)  ID(PROJAADM)

...

 ALTDSD ¢PROJA.ITEM*.*¢                    UACC(NONE)     OWNER(DE#00007)
 PERMIT ¢PROJA.ITEM*.*¢                    ACCESS(UPDATE) ID(PROJADEV)
 PERMIT ¢PROJA.ITEM*.*¢                    ACCESS(READ)   ID(PROJAINT)
 PERMIT ¢PROJA.ITEM*.*¢                    ACCESS(ALTER)  ID(PROJAADM)

 ALTDSD ¢PROJA.ITEM*T.*¢                   UACC(NONE)     OWNER(DE#00007)
 PERMIT ¢PROJA.ITEM*T.*¢                   ACCESS(UPDATE) ID(PROJADEV)
 PERMIT ¢PROJA.ITEM*T.*¢                   ACCESS(UPDATE) ID(PROJAINT)
 PERMIT ¢PROJA.ITEM*T.*¢                   ACCESS(ALTER)  ID(PROJAADM)

 /* Service data authorization ------------------------------------*/
 ALTDSD ¢PROJA.PTF*.*¢                     UACC(NONE)     OWNER(DE#00007)
 PERMIT ¢PROJA.PTF*.*¢                     ACCESS(UPDATE) ID(PROJADEV)
 PERMIT ¢PROJA.PTF*.*¢                     ACCESS(UPDATE) ID(PROJAINT)
 PERMIT ¢PROJA.PTF*.*¢                     ACCESS(ALTER)  ID(PROJAADM)
/*
//
```

Figure 3 (Part 3 of 3). Sample RACF Job

3.2.5.2 Usage of Authorization Codes

Authorization codes control the movement of data within the hierarchy. The usage of authorization codes restricts the draw down and promotion of members to certain groups within the hierarchy (see *ISPF/Software Configuration and Library Manager, Project Manager's Guide*).

With this method we allow the promotion of parts from the groups ITEMx and DEVx to the group FINAL, because the authorization code REL is defined for all these groups (see Figure 2 on page 21). If the developer uses authorization code TEMP on the groups ITEMx and DEVx, he can change the code but he will not be able to promote it to higher hierarchy groups, if the authorization code is not changed back to REL.

Additionally, the BASE group is protected because an additional authorization code is defined. Therefore, the BASE group can never be overwritten by a promote, even if the RACF access would allow it. The same thing is valid for the service groups PTFNEW, PTFWORK and PTF. For these groups, only a promote from PTFNEW to PTFWORK and PTF is allowed.

The same method also can be used to separate NLS groups (National Language Support Groups) from the BASE group (see 8.1.2, “NLS Part Development” on page 74).

The implementation of these different authorization codes is shown in Figure 4 on page 26.

```
* Base group
BASE     FLMGROUP ALTC=#BAS,AC=(BASE)
*
* Integration groups
FINAL    FLMGROUP ALTC=#INT,AC=(REL),PROMOTE=BASE
TEST     FLMGROUP ALTC=#INT,AC=(REL),PROMOTE=FINAL
*
* Development groups
DEV1T    FLMGROUP ALTC=#DEV,AC=(REL),PROMOTE=TEST
DEV2T    FLMGROUP ALTC=#DEV,AC=(REL),PROMOTE=TEST
ITEM1T   FLMGROUP ALTC=#DEV,AC=(REL),PROMOTE=TEST
ITEM2T   FLMGROUP ALTC=#DEV,AC=(REL),PROMOTE=TEST
*
DEV1     FLMGROUP ALTC=#DEV,AC=(REL,TEMP),PROMOTE=DEV1T
DEV2     FLMGROUP ALTC=#DEV,AC=(REL,TEMP),PROMOTE=DEV2T
ITEM1    FLMGROUP ALTC=#DEV,AC=(REL,TEMP),PROMOTE=ITEM1T
ITEM2    FLMGROUP ALTC=#DEV,AC=(REL,TEMP),PROMOTE=ITEM2T
*
* Service groups
PTF      FLMGROUP ALTC=#PTF,AC=(SERVICE),PROMOTE=BASE
PTFWORK  FLMGROUP ALTC=#PTF,AC=(SERVICE),PROMOTE=PTF
PTFNEW   FLMGROUP ALTC=#PTF,AC=(SERVICE),PROMOTE=PTFWORK
```

Figure 4. Exemplary Group Definitions in Project Setup. The methods described in 3.2.5.2, "Usage of Authorization Codes" and 3.2.5.3, "Use of Several Accounting Data Sets" on page 27 result in the given group definitions in the project setup.

Principles

- use **different authorization codes** to protect the project base from development groups or NLS groups
- to **load project base code**, for the corresponding alternate project use the **authorization code REL for group BASE**. This method allows that the default authorization code is set to REL and the developer does not need to set it explicitly if he edits a part.

To load the BASE group with members using the authorization REL, even if the development project BASE has the authorization code BASE, we define an alternate project BASEU:

```
    (REL)       BASE


    (REL)       BASEU
```

Figure 5. Alternate Progect to Load Product into SCLM

This alternate project then is used to load members in BASEU. Migrate, build and promote them to BASE with authorization code REL. This technique enables us to have the authorization code REL for all members in the BASE group of the development project. This allows us to draw down members from the BASE

group to the edit groups without changing the authorization codes, because both are the same. This enhances the ease of use for our developers.

We also follow the principles:

- **avoid alternate projects whenever possible**
- **never make a non-editable group editable** in another alternate project. It is preferable to use an additional editable work group in the alternate project.

3.2.5.3 Use of Several Accounting Data Sets

In Figure 2 on page 21 you see that we divided the project hierarchy into different layers (Base, Integration, Development, Service). For each layer we allocate separate SCLM Control data sets.

Exemplary implementation of different Accounting data sets

```
         FLMCNTRL OPTOVER=Y,                                  *
               MAXVIO=2000000,                                *
               VIOUNIT=SYSDA,                                 *
               MAXLINE=55

#BAS     FLMALTC  ACCT=PROJA.PROJDEF$.ACCOUNT1.BAS,           *
               ACCT2=PROJA.PROJDEF$.ACCOUNT2.BAS

#INT     FLMALTC  ACCT=PROJA.PROJDEF$.ACCOUNT1.INT,           *
               ACCT2=PROJA.PROJDEF$.ACCOUNT2.INT

#DEV     FLMALTC  ACCT=PROJA.PROJDEF$.ACCOUNT1.DEV,           *
               ACCT2=PROJA.PROJDEF$.ACCOUNT2.DEV

#PTF     FLMALTC  ACCT=PROJA.PROJDEF$.ACCOUNT1.PTF,           *
               ACCT2=PROJA.PROJDEF$.ACCOUNT2.PTF
```

The relation of the different accounting data sets to the corresponding groups is shown in Figure 4 on page 26.

With this approach we expect the following advantages:

- higher performance due to different layers updating different VSAM accounting data sets. This reduces the size of the database when going up the tree accessing the database.
- reliability and maintainability. If one of the VSAM accounting data sets becomes corrupted, it can be recovered without influencing the higher layers.

Principles

- **For each logical set of groups, use separate control data sets**
- **Choose the following naming conventions for SCLM Control data sets**

```
<hlq>.PROJDEF$.ACCOUNTx.<logical_layer>   Accnt. data sets
<hlq>.PROJDEF$.VERSIONx.<logical_layer>   Audit Ctrl. data sets
<hlq>.PROJDEF$.XREF.<logical_layer>       Cross Refer. data set
```

3.3 Introduction of SCLM to New Products

This section explains how we proceed when we establish new SCLM projects for product development. It should serve as a guide for a reader to follow as the general principle because, even if the implementation described here is SCLM specific, it is the same for all project setups. The process steps described should be library independent. It is assumed that the roles are already defined as we did in 3.2, “Roles Needed to Manage Projects” on page 17 and the teams are established.

The major steps to do are:

1. 3.4, “Product Analysis”
2. 3.5, “SCLM Implementation” on page 30
3. 3.6, “Product Migration or Load Product into SCLM” on page 31
4. 3.7, “Release Library to and Education of new SCLM Users” on page 32
5. 3.8, “Ongoing Support” on page 33

3.4 Product Analysis

During this phase a tight working relationship must be established between the SCLM support team, the integration and the development team. Each team will contribute its knowledge and expertise to define an SCLM setup that will end up in a technically and team related optimized setup.

Even though the steps are listed in sequential order, there may be several iterations necessary or the process may have to be run several times with different sets of input data.

Step 1: Understand the existing product or the new product to be developed under SCLM control. This means defining:

- the complete set of input parts (source code) to be managed.
- the processes needed, such as which compilers are used to process those input parts.
- what process parameters are needed for the processes.
- what output of the processes is subject to be kept as output under SCLM control.
- what include libraries are needed for each process and which should or must be external or under SCLM control. This follows the principles described under 4.2, “Include Types” on page 35.

Step 2: Define all the names of the types needed based on step 1 and the principles stated in 4.1, “Input and Output Types” on page 35.

Step 3: Define the SCLM language names and types (CC , LEC, Generic) based on the process needed in step 1 and any possible additional ones for packaging, integration or development needs. The languages are, if possible, created later on from the set of translator macros (see Appendix B, “Translators” on page 107) already provided.

Step 4: Define the parameter sets needed for each language and the members to be processed based on the parameters found in step 1.

At this point we designed the smallest SCLM project which could be implemented in **one** SCLM group (see also Figure 5 on page 26). All the technical subjects related to the products are already covered (that is, group BASE in Figure 1 on page 15). Therefore, the SCLM support team and the

integration team could start implementing this one group project as described in 3.5, “SCLM Implementation” on page 30 while doing the next steps in parallel. Our experience shows that if the include sets and processes are well known it is just a matter of work to implement all this with SCLM. The hardest part is to define all the names so that they make sense up to the end of the development cycle and to find all the parts, processes and parameters.

The next topics cover the organization and team structures, addressing working modes and process related topics. It is very important to get all parties involved working with the library so that responsibilities and interfaces can be mapped to the library structure to be defined. Our experience shows that this is the hardest part of the complete definitions, after which implementation is quite easy. The reason why this is so difficult is caused by the final decision making process which task(s) are handled by a group or by an individual during the development of a product and its obvious synchronization problems.

Step 5: Define the integration groups and alternate projects needed, for example group FINAL, TEST in Figure 1 on page 15. Decide also if they should have hot fix groups to create temporary fix-level drivers.

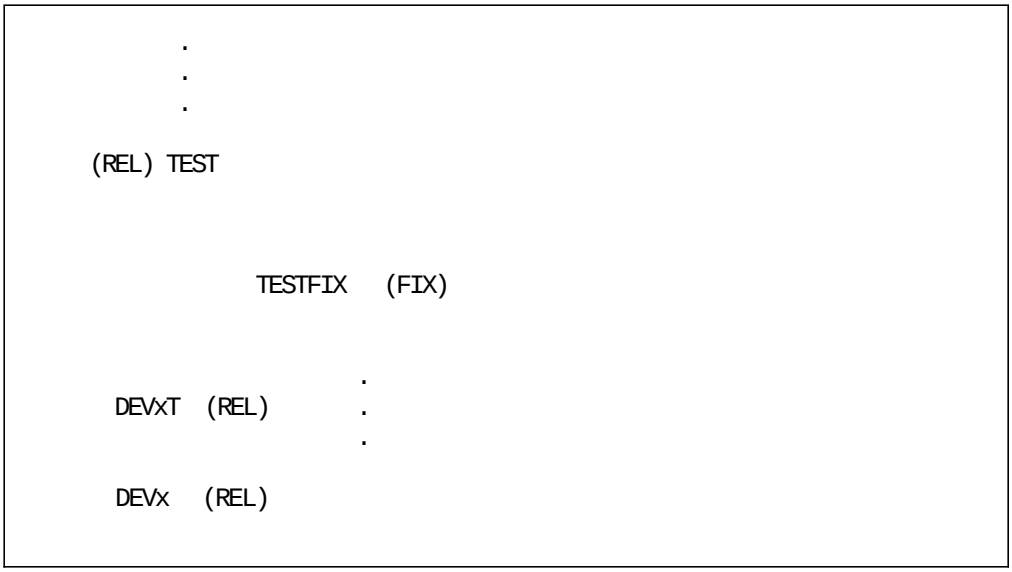


Figure 6. Hot Fix Group Structure

Step 6: Define the development groups based on the integration groups. The structure is determined by development teams and members considering the needs of parallel work. For example if only one developer works on this group, use the groups DEVx and DEVxT where DEVx is usually the user ID of the developer. If one or more developers are working on one line item simultaneously in one group then use the groups ITEMx and ITEMxT, where ITEMx stands for the line-item to be developed. Even though in this document we show just two-level development structures, the reality may end up in a more or less complex tree of development groups. These then address intermediate test steps and parallel work efforts. However they follow the same principles.

Step 7: Verify at this point if additional types are needed for any reasons.

Step 8: Define the authorization codes needed for protection. For detailed information refer to 3.2.5.2, “Usage of Authorization Codes” on page 25.

Step 9: Define the RACF authorization layer upon all the groups. For detailed information refer to 3.2.5.1, “RACF Protection” on page 22.

3.5 SCLM Implementation

3.5.1 Step 1 - Create the SCLM Project Definition

1. The project provides a high-level qualifier for the SCLM environment to store the product data and for the development.
2. The SCLM support team establishes a META project with the name of the high-level qualifier under the high-level qualifier SCLM2 as shown in Figure 13 on page 57 and described in 6.2, “Meta SCLM Projects to Develop SCLM Projects” on page 59.
3. The SCLM1.PROJDEFS.SHIPLIST is updated to enable the automatic delivery (see 8.3.2, “SHIPIT - Externalize Data during Promote Time” on page 88) and activation of the SCLM project to be defined. If the hlq is HCD then it looks typically like:

```
* Shipping list for Target HLQ HCD projects on MVS2 System
*                    Source HLQ SCLM2, Project HCD
* ____________________________________________________________________
* HCD project of HLQ HCD
*    to create a SCLM Development Environment for Product HCD
*
* The release development project
 SCLM2.HCD->HCD$.SCLMLOAD(HCD)   => HCD.PROJDEFS.LOAD
 SCLM2.HCD->HCD$.SCLMLIST(HCD)   => HCD.PROJDEFS.LOADLIST
*
* The initial update project for the BASE group
 SCLM2.HCD->HCD$.SCLMLOAD(BASEU) => HCD.PROJDEFS.LOAD
 SCLM2.HCD->HCD$.SCLMLIST(BASEU) => HCD.PROJDEFS.LOADLIST
*
 SCLM2.HCD->HCD$.SCLMJCL (HCD)   => HCD.PROJDEFS.ALLOCDEF
*
 SCLM2.HCD->HCD$.BM3820  ($HCD)  => HCD.PROJDEFS.LIST3820
 SCLM2.HCD->HCD$.BM3820  (#HCD)  => HCD.PROJDEFS.LIST3820
 SCLM2.HCD->HCD$.SCLMBOOK($HCD)  => HCD.PROJDEFS.BOOK
 SCLM2.HCD->HCD$.SCLMBOOK(#HCD)  => HCD.PROJDEFS.BOOK
* ____________________________________________________________
* Project specific execs developed
*      and to be delivered to HCD.PROJDEFS
*
```

   Note the BASEU project. It is used to fill and update the BASE group. This is described in 3.2.5.2, “Usage of Authorization Codes” on page 25.
4. Based upon the analysis done (see 3.4, “Product Analysis” on page 28) an SCLM project is created as already described (see 6.2, “Meta SCLM Projects to Develop SCLM Projects” on page 59).

   An example for such a project documentation is shown in Appendix G, “Sample Meta Project Documentation (Formatted Output)” on page 173.

It also considers the aspects described in 3.2.5.3, “Use of Several Accounting Data Sets” on page 27.

3.5.2 Step 2 - Build the SCLM Project Definition

Based upon the created SCLM definitions a build is issued under the meta project of hlq SCLM2 on the high-level architecture member of the complete project definition structure. This creates:

- Project load module
- Project documentation generation

  How this is done is described in 7.2, “Documentation Development Environment” on page 70.
- Allocation information generation

  How this is done is described in D.6, “SCLMINFO - Generate Data Set Allocation Information” on page 143.

3.5.3 Step 3 - Prepare SCLM Project

1. Establish all the hlq.PROJDEFS.* data sets as defined in 6.2.1, “hlq.PROJDEFS.* Data Sets” on page 62.
2. Promote the HL architecture member when the build was successful in step 2.
3. Use an allocation tool to allocate all data sets needed for the SCLM project. Refer to D.7, “SALLOC - Allocate Data Sets” on page 147
4. Activate the defined RACF layer. Establish the RACF JCL and submit it, as described in 3.2.5.1, “RACF Protection” on page 22.

3.6 Product Migration or Load Product into SCLM

At this point all the project definitions and basic setups are done. Now is the time to load the SCLM project with the data to be managed. The following section addresses this topic, as well as which tools and processes are used. In general the migration is described in D.2, “MAKEARCD - Generate Architecture Definitions from Models” on page 126.

Source part migration

In general, to migrate product source parts in an SCLM project we use the standard SCLM Migrate Dialog (Option 3.3). In addition see D.2, “MAKEARCD - Generate Architecture Definitions from Models” on page 126.

Generate the Architecture definitions

However besides the source parts to be migrated we also have to generate and establish the related architecture definitions as described in 4.3, “Types of Metadata” on page 36.

Create the COMPDEF members

To create the architecture definition for each source we have developed some architecture generation tools.

Tool MAKEARCD

Based on a model member and a list of members it is possible with this tool to generate the architecture definitions for those members.

In addition it is possible to generate LECs for VSE based on VSE linkbooks.

For more details see D.2, “MAKEARCD - Generate Architecture Definitions from Models” on page 126.

Create the LINKDEF members

Tool LNK2ARCD

With this tool it is possible to generate the related CC, LEC and HLs based on MVS LINKJCL.

For more details see D.3, “LNK2ARCD - Generate Architecture Definitions from LINK JCL” on page 132.

Create the PARMDEF members

The PARMDEF members are used to maintain PARMx definitions across members from a central place. In the generated CC and LEC architecture members there are COPY statements to those PARMDEF members.

Create the PRODDEF members

The PRODDEF members are partially already generated by the architecture member generation tools. However, they may have to be modified to define the correct set of members to build. Additional PRODDEF members may have to be created to define logical subgroups to be built for several purposes and with different options.

This work is normally done by the integration team. If it is the first time, the SCLM support team should provide guidance to help them, following the principle *training on the job*.

If the architecture definitions are created the integration team can start to create the BASE driver, that is, issue builds and verify that everything works properly.

If problems are found in the build functions then the SCLM support team has to provide a solution. If problems are found in the source or parameters of the build, then the developers must provide the solution. If environmental problems occur then the integrator has to do the follow up, either changing the SCLM definitions or data allocations or getting help from the Infrastructure Support Group which provides the MVS2 system.

3.7 Release Library to and Education of new SCLM Users

Education

Normally newcomers to the integration team are trained by the SCLM support group on the job. They create the project definition together with the team member step by step for the first time, so that they can do the following modifications and iterations by themselves.

For the complete project setup the SCLM support team provides a one day session for the development teams on request once or, if needed, several times. It is demonstrated how to do the edit, build, promote and other procedures they need with SCLM on their real project by providing, besides the development groups, also some dummy groups as shown below. These allow us to play around with real product members, processes and SCLM functions without disrupting the real development process and data.

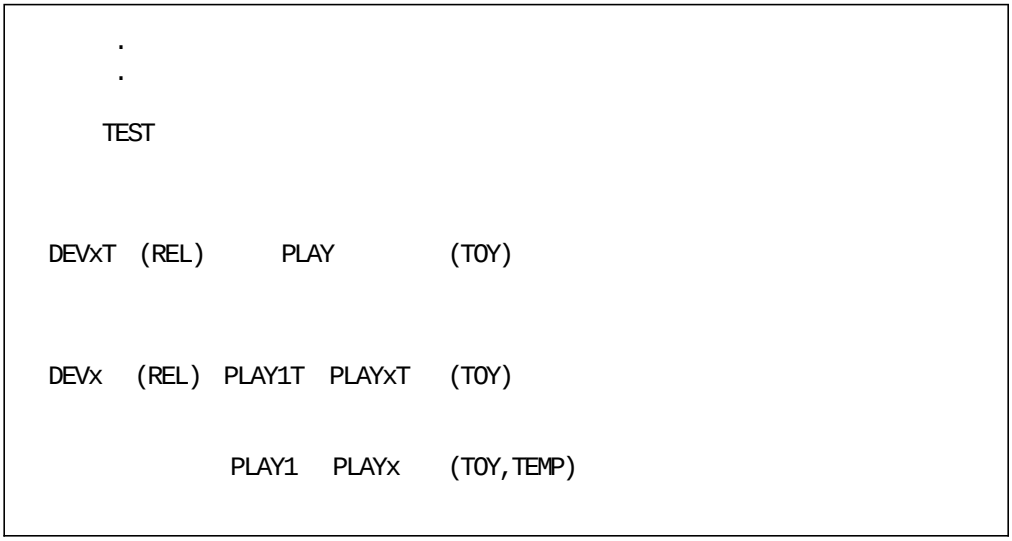


Figure 7. Setup for SCLM Training

Practice has shown that such a training is sufficient and only in the beginning are some additional hands-on training and telephone call assistance needed.

User environment setup

Before the user can work in an SCLM controlled environment he must issue the exec below from TSO READY or put this command in his default CLIST that runs at logon time. For an HCD developer or integrator it must look like:

```
EXEC ¢SCLM.RUNTIME.ALLOC(SCLMSITE)¢ ¢PROJECTS(HCD)¢ EXEC
```

More details about the exec are shown in D.9, “SCLMSITE - Basic SCLM Project Environment Setup” on page 151.

3.8 Ongoing Support

Under ongoing support we understand that the SCLM support team is still the focal point if problems arise using SCLM that cannot be solved by the end users or integrators. That means be the first contact for the integration team to answer SCLM related questions and provide help, but the second for the developers.

The SCLM support team tests the processes and works on extensions and new processes and exchanges experience across projects. The goal is to put all projects on a common set of processes and definitions as far as it can be done without detriment to the efficiency and quality of any single project.

If upgrades from one ISPF release to another are necessary, we prepare those upgrades and help to upgrade the single projects also at the appropriate times.

Chapter 4. SCLM Types

4.1 Input and Output Types

The development team is normally asked to define the names of the input types. The decision is based upon the types of program sources and the potential final output of data produced by used products and tools. So, for example, it is common to store all source modules independent of the programming languages (such as C, C++, PLI) in one type (for example, MOD) because all builds, even using different compilers, produce an object deck of the same output type. Panels for example (such as ISPF format or DTL) are stored in another type, (for example, PANEL) because they have other output types.

We differentiate between development defined output types and distribution defined output types.

Distribution defined output types contain all the members that are subject to packaging and delivery. Development defined output types are only needed during the development cycle, for example, intermediate output, certain listings or members for debuggers.

This is summarized in the principles listed below.

Principles to follow

- The **development team defines** the names of the **input types** and the **development output types**.
- The **integration team defines** the names of the **distribution output types** containing the parts to be packaged and delivered to customers.
- There are **no input distribution types**.
- If **source members have different languages** but the **same output types** are used, put the source members in the **same input types**.
- If **unique source member names** are to be ensured put all members in **one type**, even though they have different languages associated with them.
- If **duplicate member names** may be expected use **several types**.

4.2 Include Types

In SCLM there are several include techniques possible:

SCLM controlled members

Changes of the members cause rebuilds of related members. Types can be defined via:

- FLMINCLS macro

  This technique introduced with 4.1 is more flexible and reflects the purpose of include library support better. So it is the technique of choice for the SCLM controlled include libraries.

In practice we tend to define all include libraries using this technique.

- ... FLMTYPE EXTEND=...

  Up to 4.1 this was the only possible way. With 4.1 we decided to drop this technique because it works differently to the FLMINCLS technique. Therefore, we avoid confusion due to the different chaining techniques of FLMINCLS and FLMTYPE across groups. (See the SCLM documentation for the differences between FLMTYPE EXTEND= and FLMINCLS.)

  If we use it, then only in cases where we really mean to have an extended source type and not an include library.

Not SCLM controlled members

This is defined via the FLMSYSLB macro and the option ALCSYSLB=Y (with 4.1) of the FLMLANG macro.

This technique is only used if it is assured that no or very limited changes occur in the libraries during one development cycle. Rebuilds are not desired or, if necessary, they can be managed manually without SCLM doing the detection for rebuild.

As a summary:

Principles to follow

- For **SCLM controlled include libraries** use the **FLMINCLS macro**
- **Avoid if possible the FLMTYPE EXTEND=... technique**
- For external, **not SCLM controlled include libraries** use the **FLMSYSLB macro** and the option **ALCSYSLB=Y of the FLMLANG macro.**

4.3 Types of Metadata

With metadata we actually talk about the different architecture definitions, such as the CC, LEC, HL and generic architecture definitions.

Instead of just having one SCLM type, ARCHDEF, as used throughout the official SCLM documentation and various redbooks or using default types in languages we follow the principles:

Principles to follow

1. **Never use default types in languages**

   We found out that in most cases one needs architecture members in addition to default types. This will then complicate the maintenance and control of metadata. It is easier for the developers to associate:

   - a CC member with each source
   - an LEC with each module
   - HL members with logical groups

2. **Provide for each source member one architecture member with the same name**

   A sample list of ″type names″ we used is in Table 2 on page 38. This enables a straight forward mapping between source members and CC or LEC members. No naming convention is necessary in the member name for different kinds of architecture members. Also the list of members is shorter with this technique because the members are now distributed across types and not in just one type.

   For CC members the same member name can be used as defined by the member name of the SINC statement.

   For LEC members use the same name as the output member to be produced with the keyword LOAD.

3. **Have different types for CC, LEC and HL members**

   This allows us to have the same names and not have to invent some complicated or cryptic naming conventions.

4. **Never (or only in exceptional cases) use the architecture PROM keyword**

   This relates directly to principle 2 above.

5. **Use PARMDEF type members to keep PARMx keywords separate to allow central maintenance.**

   If parameters have to be specified in an architecture member then put them in a PARMDEF member, especially if the parameter is not only used for just one member.

Table 2 on page 38 shows an example of different types of architecture members and their usage.

Table 2. Metadata Types Usage

SCLM Type	Architecture Type	Description
COMPDEF	CC	Contains the CC architecture members for each source program (for example stored in a type MOD). MOD may contain members of different languages such as C, C++ or PLI. The names of the members in COMPDEF and MOD are the same. The members are typically owned by the integration team.
PANLDEF	Generic	Basically the same as COMPDEF, however for panels such as DTL, ISPF, or CICS BMS MAPS format, which may be stored in the source type PANL. The members are typically owned by the integration team.
JCLDEF	Generic	Basically the same as COMPDEF, however for job control languages which may be stored in the type JCL. The members are typically owned by the integration team.
LINKDEF	LEC	Contains the link information to create the load modules from COMPDEF definitions. The member names are the same as the produced load module or output. The members are typically owned by the integration team.
PRODDEF	HL	Contains the logical groups of the product and one or more packaging members to define the scope of total product build. The members are typically owned by the integration team.
WORKDEF	HL	Contains the work related versions of COMPDEFs and LINKDEFs and members to define work lists and integration lists (driver contents). The members are typically owned by the development team members. For their usage see also 3.2.5, “Methods to Implement Team Responsibilities” on page 20.
PARMDEF	COPY	Contains parameter sets used across members or even product. The members are typically owned by the integration team members.

Chapter 5. VSE Related Development Environment Differences

In general the development and integration of VSE code in SCLM isn't much different from the development and integration of other products in SCLM (as is described in 3.1, "BPL Development Environment to Manage Products" on page 13). Using SCLM, a developer issues a Build on a part via an architecture member, if necessary the member is translated and the different output is produced for example, lists, pre-linked parts and load modules. You can then promote all this to an upper group.

For VSE this is similar. You have to compile source code, objects will be generated and after that, you have to link one or more objects together into a PHASE (VSE executable modules). However this link can only be performed on a VSE system. Therefore, the difference between the VSE integration and the normal MVS product integration is that some steps of the build process happen outside SCLM on a VSE system. In addition the packaging is different and actually done on the VSE system itself. This chapter will explain how external processing and data control are done with SCLM.

5.1 VSE Hierarchy and Responsibilities

The VSE hierarchy and responsibilities are very similar to those described in 3.1, "BPL Development Environment to Manage Products" on page 13 and 3.2, "Roles Needed to Manage Projects" on page 17. However because a complete operating system is developed and packaged made up of several single product components, the hierarchy reflects those product development structures. This is shown in Figure 8 on page 40.

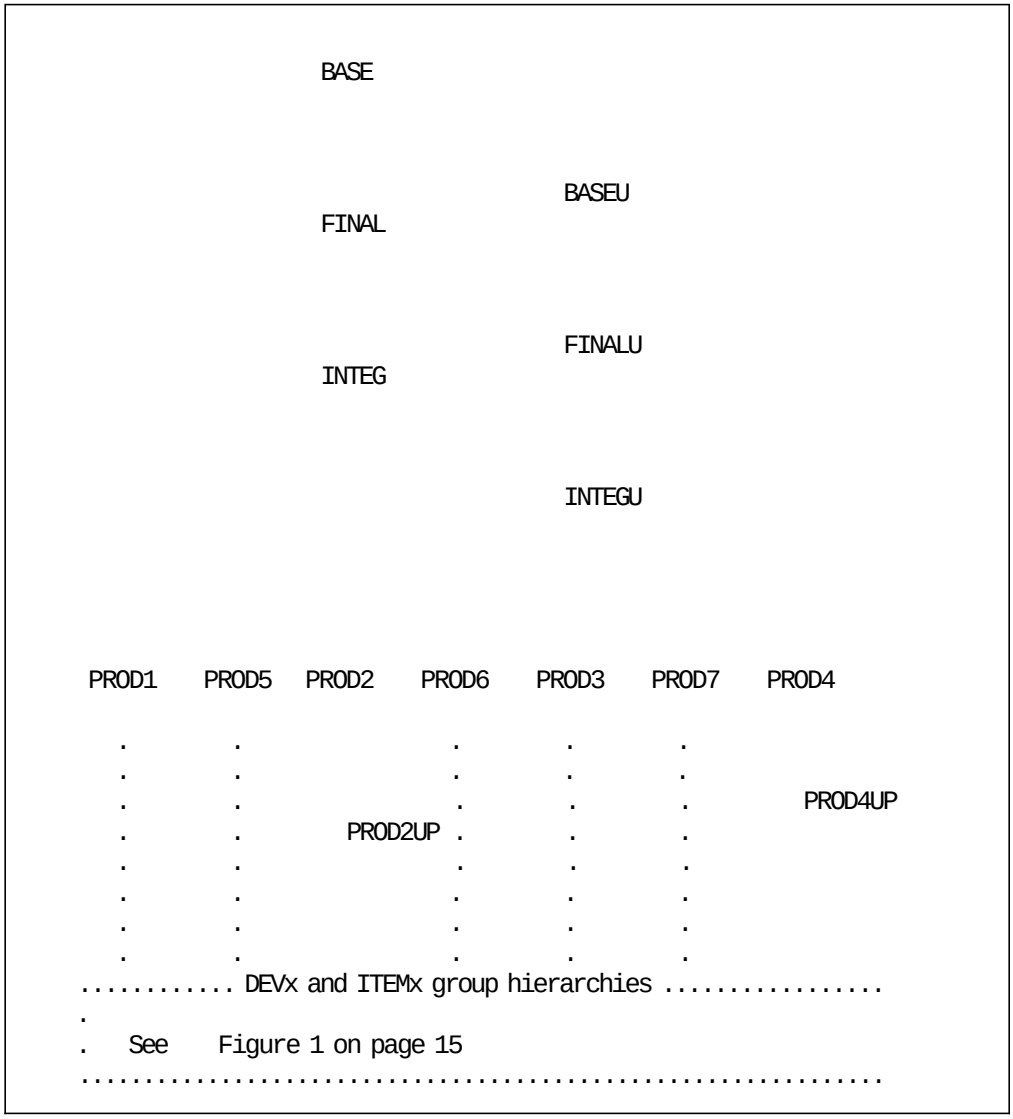


Figure 8. VSE Group Hierarchy

The product development teams are responsible for their PRODx groups and the integration team is responsible for the integration groups (that is INTEG, FINAL, BASE). The BASE group is the responsibility of the integration team as well.

BASE Base code to start new release development.

BASEU Update group for the BASE group.

FINAL Final release group. This group contains the code modified for the new release.

FINALU Update group for the FINAL group.

INTEG Integration level. This group comprises the code to be integrated into the VSE system.

INTEGU Update group for the INTEG group.

PRODx These groups reflect the different components of VSE/ESA such as POWER and MSHP.

PRODxUP Update groups for the PRODx groups.

As you can see, sometimes there appears to be only an update group for the PRODx groups. Because some components aren't very big, only one group is necessary.

5.1.1 Integration

The tasks of the integration team are in general described in 3.2.3, "Project Integration Team" on page 19. VSE has naturally some project specific processes, because the VSE objects have to be linked together to a PHASE on a VSE system. Therefore, the VSE integration team has to establish a VSE system environment, where the contents are also controlled through SCLM.

This means, that there must be a mapping between the SCLM hierarchy (the different groups (levels of code)) and the VSE libraries. This mapping is described in 5.1.3, "MVS to VSE Library Mapping." The different processes to synchronize SCLM and the VSE libraries are described in 5.2.7, "Process Synchronization MVS to VSE" on page 50.

5.1.2 Development

The tasks of the development team are in general described in 3.2.4, "Project Development Team" on page 20. In VSE the same concept for architecture members is used as described in 4.3, "Types of Metadata" on page 36. However the use may differ in that the architecture member types PRODDEF, COMPDEF, LINKDEF, are official SCLM architecture definition files, which are under integration control. This means the developer cannot do updates to those types but can use them only for builds. The developers now have the WORKDEF type which is an SCLM Architecture member for general development work referring to the other types if necessary and containing members with the shipment list for drivers (see 3.2.5, "Methods to Implement Team Responsibilities" on page 20.).

The WORKDEF members used for integration only comprise SCLM INCL architecture statements, which refer to existing PRODDEF, COMPDEF, LINKDEF, and so on.

The WORKDEF members are also used to hold architecture definition members for sending object code to VSE development test machines. This approach is described in 5.2.1, "VSESEND Process for Development Test" on page 47.

5.1.3 MVS to VSE Library Mapping

SCLM is the control library for all build processes, including the processes which happen on a VSE system. Therefore, an accurate mapping between the SCLM hierarchy and the VSE libraries is necessary to avoid network transmission overhead because some output generated on VSE is needed for packaging purposes.

Currently as a first step, we established VSE libraries only for the SCLM groups, which are controlled through the integration department (BASE, FINAL, INTEG and their update levels BASEU, FINALU, INTEGU).

The other groups don't have a corresponding VSE library yet, because we want to verify our concept. But the final goal is to have a VSE library or sub-library specified for each SCLM group and type.

There are three types of VSE libraries. The VSE group libraries (VSEGRP), production (VSEPRD) and the mini libraries (VSEMINI).

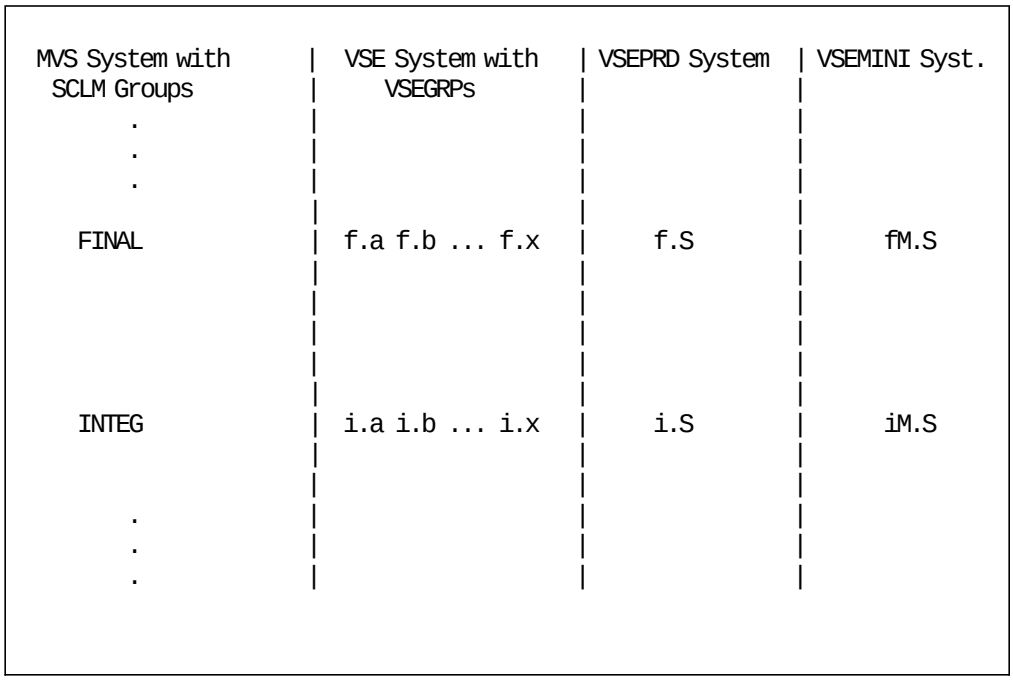


Figure 9. SCLM to VSE Library Mapping

SCLMGRP - SCLM group data sets:

<p>.<g>.<t>(<m>)

The data sets on MVS containing all source and output that are needed for developers and not for packaging or testing on a VSE system.

VSEGRP - VSE group libraries:

<g>.<compname>(<outmem>.<vsetype>)

These libraries comprise all objects, A-books, Z-books, phases and so on from a VSE component (for example, POWER or MSHP). They have a one to one mapping to the SCLM groups.

VSEPRD - VSE production system libraries:

<g>.<prodlib>(<outmem>.<vsetype>)

These libraries comprise all **shipped** objects, A-books, Z-books, phases and so on from VSE. They reflect the standard VSE libraries (for example, IJSYSRS.SYSLIB, PRD1.BASE, PRD2.GEN1). This is a system that represents a full blown packaged VSE system of a group level.

VSEMINI - VSE mini system libraries:

<g>M.<minilib>(<outmem>.<vsetype>)

These libraries only comprise the phases which are necessary to drive a VSE mini system (Supervisors, Librarian, VSE component AF and so on). These libraries are for such things as development purposes or for testing supervisors. This is a system that represents a packaged minimal VSE system of a group level.

The meanings are:

< p > SCLM project

< g > SCLM group

< t > SCLM type

< m > SCLM member

<compname> Determined from an architecture parameter

PARMx -COMP <compname>

<prodlib> Determined from an architecture parameter

PARMx -DISTsss <prodlib_synonym>

where sss is LNK for linked, OBJ for object and SRC for source members and <prodlib_synonym> is taken from an environment file parameter

<prodlib_synoym>: *.<prodlib>

where * is replaced by <g> and <prodlib> the real name to be used on VSE.

<minilib> This is determined from an environment file parameter

VSE_system_mini_slib: <minilib>

<outmem> This is determined from an architecture parameter

OUTx <outmem> <outtype>

where <outmem> is the created *reference member* containing at least which data is stored on which VSE machine and is under SCLM control. During PROMOTE this member is used by a promote exit to synchronize promote actions on VSE. The <outtype> is the SCLM type of the *reference member*.

<vsetype> This is determined from an architecture parameter

PARMx -VTYPsss <vsetype>

where sss is the same as for <prodlib>.

All generated objects, source code (such as A-books) and phases will be cataloged into the VSE group libraries. After that, depending on parameters in the COMPDEF member, the part will be cataloged into a production library and into a mini library. This is described later with an example in 5.2.7, “Process Synchronization MVS to VSE” on page 50.

The mapping information for the SCLM group.type and the VSE libraries are stored in the environment data set (see 5.1.4, “VSE Environment Data Set”).

5.1.4 VSE Environment Data Set

The VSE environment data set is used by build and promote translators and comprises all information about library mapping, exec variables, and so on. It’s a PDS in the <p>.PROJDEFS.VSEENV data sets and consists of different members. Each member contains information for special purposes, but all have the same structure.

Member SHIPCODE

This member is commonly used by all different environment members via *ship_code: SHIPCODE*. A sample is shown below. It contains at least the definition of which SCLM groups are subjects for mapping to VSE libraries and systems. In other words only for those groups a copy or remote execution during build and promote are done. For all others it is ignored.

```
*   Environment file for different execs to get level of
*   shipping code to the VSE System
*
* ---------------------------------------------------------
* REMEMBER!  All between : and -- will be the variable value
* ---------------------------------------------------------
* keyword:            value          -- description
* ---------------------------------------------------------
VSE_ship_code:        YES            -- Ship to VSE code
*                                       NO  =ship it
*                                       YES = don¢t ship it
Ship_level:     BASE,BASEU,FINAL,FINALU,INTEG,INTEGU
*                                       -- groups,
*                                       from which the code should be
*                                       shipped
* ------------------------------------------------------------
* The next lines show from which SCLM groups the
* code will be shipped to which VSE library
* SCLM group          VSE library
* ------------------------------------------------------------
BASE:                 BASE           -- ship code from group BASE
*                                       to VSE library       BASE
BASEU:                BASE           -- ship code from group BASEU
*                                       to VSE library       BASE
FINAL:                FINAL          -- ship code from group FINAL
*                                       to VSE library       FINAL
FINALU:               FINAL          -- ship code from group FINALU
*                                       to VSE library       FINAL
INTEG:                INTEG          -- ship code from group INTEG
*                                       to VSE library       INTEG
INTEGU:               INTEG          -- ship code from group INTEGU
*                                       to VSE library       INTEG
```

Member VSEMOD

This member is used for compiles done via COMPDEF members. The SCLM language defines its use. A sample is shown below:

```
*   Environment file for members with translator
*
*        VSEMOD
*
* ---------------------------------------------------------
* The format of the file must be like the following line
* ---------------------------------------------------------
* REMEMBER!  All between : and -- will be the variable value
* ---------------------------------------------------------
* keyword:            value          -- description
* ---------------------------------------------------------
```

```
Ship_code:            SHIPCODE       -- Ship code to VSE system
*                                       NO = no code shipment performed
*                                       xxxxx= name of VSEENV member
*                                       which includes information
* Source specs defaults
Ship_Source:          NO             -- Ship SOURCE code
outsrc_vsetype:       A              -- default membertype for SOURCE
outsrc_distlib:       INTERNAL       -- default distribution library
*
* Object specs defaults
Ship_Object:          NO             -- Ship OBJECT code
outobj_vsetype:       OBJ            -- default membertype for OBJECTS
outobj_distlib:       INTERNAL       -- default distribution library
*
* VSE System control infos
max_valid_rc:         4              -- maximum valid rc
VSE_system_node:      BOEVMCT1       -- NodeID of the Integration VSE System
VSE_system_uid:       VSE230         -- UserID of the Integration VSE System
VSE_spool_class:      B              -- Spool Class for the VSE System
*
* VSE Job submission skeletons
vsejcl_skel:          @VSEMODT       -- name of VSE JCL skeleton
*
* VSE Job submission data insertion keywords
incl_SRC_id:          #$INCLSRC      -- id for SRC code include
incl_OBJ_id:          #$INCLOBJ      -- id for OBJ code include
*
* VSE Job output information
VSE_system_INFO_lib:  INTEG          -- Library name
VSE_system_INFO_slib: SCLMINFO       -- Sublib name
*
* Target mini system
VSE_system_MINI_lib:  INTEG          -- Integr. VSE System MINI lib
VSE_system_MINI_slib: MINI           -- Integr. VSE System MINI sublib
*
* Timing for Foreground
wait_sleeptime:       10             -- sleeptime for wait in sec.
wait_timeout:         4              -- timeout for wait in min.
wait_msg_count:       5              -- issue wait_sleeptimes msg
*
* Timing for Batch
b_wait_sleeptime:     10             -- sleeptime for wait in sec.
b_wait_timeout:       30             -- timeout for wait in min.
b_wait_msg_count:     12             -- issue wait_sleeptimes msg
*
* ═══════════════════════════════════════════════════════════════
* LIBRARY SYNONYMS to map to real VSE library and sublibs.
*
SYSRES:             @@FLMGRP.SYSRES   -- System library IJSYSRS.SYSLIB
SYSLIB:             @@FLMGRP.SYSRES   -- System library IJSYSRS.SYSLIB
GENLIB:             @@FLMGRP.PRD2GEN1 -- Generation library PRD2.GEN1
MACLIB:             @@FLMGRP.PRD1MAC  -- Macro library PRD1.MACLIB
BASLIB:             @@FLMGRP.PRD1BASE -- Base library PRD1.BASE
OPTIONAL:             IOPT230.OPT     -- OPTIONAL Material
INTERNAL:           @@FLMGRP.INT      -- INTERNAL Library
*
* ═══════════════════════════════════════════════════════════════
```

Member VSELNK

This member is used for links done via LINKDEF members. The SCLM language defines its use. A sample is shown below:

```
*   Environment file for member with translator
*
*        VSELNK
*
* ---------------------------------------------------------
* REMEMBER!  All between : and -- will be the variable value
* ---------------------------------------------------------
* keyword:            value          -- description
* ---------------------------------------------------------
Ship_code:            SHIPCODE       -- Ship code to VSE system
*                                       NO = no code shipment performed
*                                       xxxxx= name of VSEENV member
*                                       which includes information
*
* Source member specs default
outsrc_vsetype:       A              -- default membertype for SOURCE
*
* Object member specs default
outlnk_vsetype:       PHASE          -- default membertype for OBJECTS
*
* Phase member specs default
outphs_distlib:       SYSLIB         -- default distribution library
*
* VSE system processing parameters
max_valid_rc:         4              -- maximum valid rc
VSE_system_node:      BOEVMCT1       -- NodeID of the Integration VSE System
VSE_system_uid:       VSE230         -- UserID of the Integration VSE System
VSE_spool_class:      B              -- Spool Class for the VSE System
*
* VSE job skeletons
vsejcl_skel:          @VSELNKT       -- name of VSE JCL skeleton
*
* VSE Job submission data insertion keywords
incl_SRC_id:          #$INCLSRC      -- id for SRC code include in VSE Job
incl_LNK_id:          #$INCLLNK      -- id for LNK code include in VSE Job
incl_LBD_id:          #$INCLLBD      -- id for LNK code include in VSE Job
*
* VSE Job output information
VSE_system_INFO_lib:  INTEG          -- Integr. VSE System Information lib
VSE_system_INFO_slib: SCLMINFO       -- Integr. VSE System Information slib
*
* Target mini system
VSE_system_MINI_lib:  INTEG          -- Integr. VSE System MINI lib
VSE_system_MINI_slib: SYSLIB         -- Integr. VSE System MINI sublib
*
* Timing for Foreground
wait_sleeptime:       5              -- sleeptime for wait in sec.
wait_timeout:         4              -- timeout for wait in min.
wait_msg_count:       10             -- after x wait_sleeptimes msg
*
```

```
* Timing for Batch
b_wait_sleeptime:     10            -- sleeptime for wait in sec.
b_wait_timeout:       30            -- timeout for wait in min.
b_wait_msg_count:     12            -- after x wait_sleeptimes msg
*
* ═══════════════════════════════════════════════════════════════
* LIBRARY SYNONYMS
*
SYSRES:             @@FLMGRP.SYSRES   -- System library IJSYSRS.SYSLIB
SYSLIB:             @@FLMGRP.SYSRES   -- System library IJSYSRS.SYSLIB
GENLIB:             @@FLMGRP.PRD2GEN1 -- Generation library PRD2.GEN1
MACLIB:             @@FLMGRP.PRD1MAC  -- Macro library PRD1.MACLIB
BASLIB:             @@FLMGRP.PRD1BASE -- Base library PRD1.BASE
OPTIONAL:            IOPT230.OPT      -- OPTIONAL Material
INTERNAL:           @@FLMGRP.INT      -- INTERNAL Library
*
* ═══════════════════════════════════════════════════════════════
```

5.2 Overview on Used SCLM Processes

5.2.1 VSESEND Process for Development Test

In our VSE development environment the code is edited and compiled on a MVS system. However it must be linked, executed and tested on a VSE system. For this purpose the developers wrote a tool using CLEAR as the development library. This did the compiles on checked-out code and sent the generated objects to their desired target test system and optionally did the link of objects to the desired phase on the target environment.

The compiles are now covered by SCLM. For the delivery to the target VSE system we adopted the same process in SCLM by adding a new language which will send a part to a desired VSE system with additional LINKJCL, if desired. An architecture member is used to pass the needed parameters, such as target system id, names, option if catalog, catalog and link or only link.

This language is then defined for the generated objects during the compile (build time) using the LANG keyword of the FLMALLOC macro in the related FLMTRNSL that does the compile.

Therefore, doing a build on such an architecture member replaces the tool created formerly and integrates the function in the same SCLM user-interface and technique. This architecture members can then be added in other members so that the developer is now able to do all compiles of objects, catalog them and then link the phases in one step. This function goes beyond the scope of our old tool.

This technique is used as long as there is no total mapping of all SCLM groups to VSE group libraries.

5.2.2 Panel Development with SDF II

In VSE one user interface is CICS. For these programs SDF II is used to create the BMS maps. We adapted the described interface to SCLM to the extent that we generalize the exits to have parameters to be set for customization either placed at one point to be specified or taken directly from variables set by SCLM.

The REXX execs and macros we changed are *DGIIEDT, DGIIX, DGIIXBLD, DGIIXINV, DGIIXISP, DGILXIO, DGILXLI, DGILXOD, DGILXOL, DGILXOR*.

In addition we set the IMACRO to call the SDF II edit exit for the SDF II panel group and panel member data sets. This brings up the SDF II editor automatically if a developer edits a member.

With these modifications we achieved an easy integration and usage of SDF II in the standard SCLM interface. To edit, build, promote SDF II parts now works as for any other part with the only difference that instead of getting the ISPF editor, we automatically get the SDF II editor for the SDF II types.

5.2.3 Compiles on MVS

As described above, this works the same as for MVS products with the only difference that if outputs generated have to be stored on VSE they are transferred to VSE as described in 5.2.7, “Process Synchronization MVS to VSE” on page 50.

A sample CC architecture member is:

```
*
* MODULE  PROCESSING MEMBER
*
SINC $$A$SUPI     MODDOS
LIST $$A$SUPI     LISTING
OBJ  $$A$SUPI     OBJ
OUT0 $$A$SUPI     VSEMOD             --> VSELIB LIBRARY INFORMATION
*                                        (Reference Member)
*
COPY DOSMOD  PARMDEF    --> PRODUCT COMPONENT PARAMETER
*
* ═══════════════════════════════════════════════════════
*   -SHIPSRC    YES       - Ship SRC to customer
*               NO        - Don¢t ship SRC to customer
*   -DISTSRC    SYSLIB    - VSE library IJSYSRS.SYSLIB
*               MACLIB    - VSE library PRD1.MACLIB
*               GENLIB    - VSE library PRD2.GEN1
*               BASLIB    - VSE library PRD1.BASE
*               OPTIONAL  - VSE library optional Material
*               INTERNAL  - VSE library for internal Macros
*   -VTYPSRC    A         - VSE type
*               PROC      - VSE type
*               Z         - VSE type
*
PARM0 -SHIPSRC  YES
PARM0 -DISTSRC  OPTIONAL
PARM0 -VTYPSRC  A
* ═══════════════════════════════════════════════════════
```

```
*   -SHIPOBJ    YES      - Ship SRC to customer
*               NO       - Don¢t ship SRC to customer
*   -DISTOBJ    SYSLIB   - VSE library IJSYSRS.SYSLIB
*               BASLIB   - VSE library PRD1.BASE
*   -VTYPOBJ    OBJ      - VSE type
*
PARM0 -SHIPOBJ  NO
PARM0 -DISTOBJ  SYSLIB
PARM0 -VTYPOBJ  OBJ
* ═══════════════════════════════════════════════════════════════
*   -OCO     YES      - SRC code is OCO
*            NO       - SRC code isn¢t OCO
*
PARM0 -OCO   NO
* ═══════════════════════════════════════════════════════════════
*   -SEQ#NUM YES      - SRC code with sequence numbers
*            NO       - SRC code without sequence numbers
*                       (after column 72)
*
PARM9 -SEQ#NUM  YES
* ═══════════════════════════════════════════════════════════════
```

The DOSMOD PARMDEF member is:

```
* --> PRODUCT COMPONENT PARAMETER Definition
*
*     DOSMOD     -  MODULE for DOS (AF)    COMP 06606
PARM0 -COMP    DOS
PARM0 -COMPID  06606
PARM0 -CLC     15C
PARM0 -PRODID  06615C
*
* ═══════════════════════════════════════════════════════════════
```

5.2.4 Catalog Parts to VSE System

If parts stored in SCLM must also be stored on VSE a build translator step will issue the transfer of the part to VSE as described in 5.2.7, “Process Synchronization MVS to VSE” on page 50.

5.2.5 Link Objects on VSE

If objects have to be linked to a phase this must be done on a VSE system. To do this we use the LEC architecture members with a user-defined linkage editor language which gets the parts to be linked and the associated parameters. A link job is generated that is sent to the VSE system to do the link. The objects have already been sent to VSE during the compile step.

Sample of a LINKDEF member:

```
*
* Link Definitions for DOS   modules to create a PHASE
*
 INCL $$A$SUPI COMPDEF
*
 LOAD   $$A$SUPI VSELINK  -- The VSE Link Book to gen the phase
 OUT0   $$A$SUPI VSEPHASE -- The reference member to the Power PHASE
*LMAP   $$A$SUPI LINKMAP  -- The link map, Output from the
* ---------------------------------------------------------------
* Parm defintions like LKED, PARMx are hidden in the following
* members:
*
 LKED   LNKVSE                  -- call link EXIT
*
* --> Product, Component and CLC Information
COPY    DOSMOD    PARMDEF
*
* --> Product Information Parameters
PARM0   -COMP    DOS
*
* --> Product Integration-Processing Parameter
PARM0   -DISTPHS SYSLIB
*
* ===============================================================
* Created from VSE.BASEU.OBJ($$A$SUPI)
* originally at 28 Feb 1996 16:56:39 by exec MAKEARCD D960118
```

5.2.6 Packaging of a VSE System

Packaging of a VSE system is done by a copy of members to a target system. From this system the distribution is done by issuing MSHP to create the tapes or distribution media.

5.2.7 Process Synchronization MVS to VSE

This section explains the process of build that initiates external VSE processes. Today there are basically two types of processes needed. The catalog process (storing and deleting members from VSE libraries) and the link of objects, stored on a VSE system, to phases.

The members used to build these two basic processes are listed in 5.2.3, “Compiles on MVS” on page 48 and 5.2.5, “Link Objects on VSE.” The overview of these processes is shown in Figure 10 on page 51.

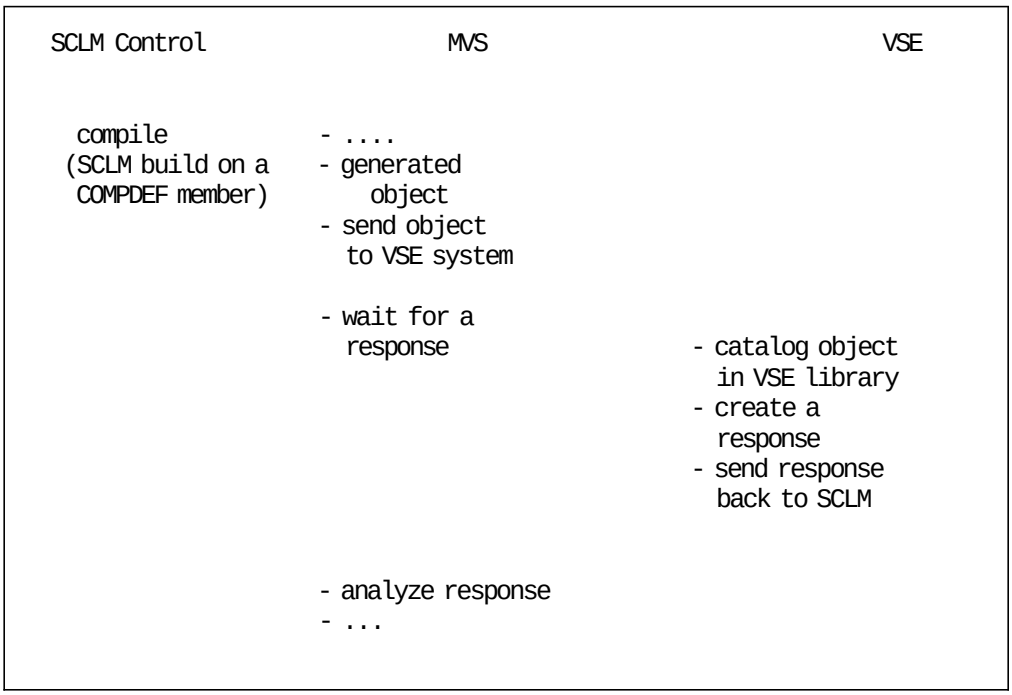


Figure 10. Overview of SCLM Controlled MVS Compile and/or VSE Catalog Processes

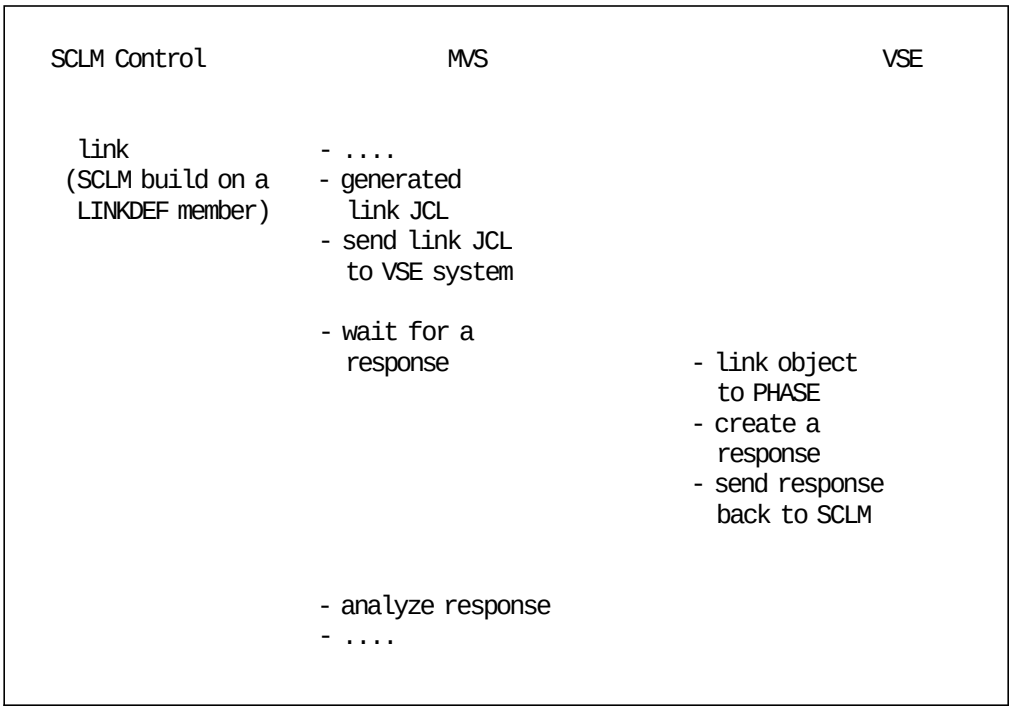


Figure 11. Overview of SCLM Controlled VSE Link Processes

5.2.7.1 Process on the MVS System

1. Issue a build on a COMPDEF or LINKDEF member
2. One translator step creates a job stream dependent on the passed parameters (see 5.2.3, “Compiles on MVS” on page 48, 5.2.5, “Link Objects on VSE” on page 50) and the environment definition file (see 5.1.4, “VSE Environment Data Set” on page 43) containing the actions to be done on the VSE system.

   The job contains also data to be sent that is inserted in the used job skeletons at the given *insert_keywords incl_*sss*_id:* defined also in the environment file.
3. Send job to VSE
   - Create an empty response file that will be updated by the return of the VSE job.
   - The job is transmitted by ALLOC to JES. This starts the 5.2.7.2, “Process on the VSE System.”
4. Wait for a response (Feedback)

   Now the translator goes into a loop doing a sleep and wake-up sequence while waiting for the response from the VSE system. This is established by checking the contents of the response file.

   To optimize the loop there are several timer options specified in the environment file. They can also be set for each member via an architecture member parameter. Time options are, for example, the duration of sleep (first time and interval), time out if no response is received due to network problems.
5. Analyze response (Feedback)

   When a response comes back to the defined response file it may look like:

```
Catalog $$ABERAC.OBJ                    tolib BASE.DOS    rc=000
Catalog $$ABERAC.A                      tolib BASE.DOS    rc=000
Copy    $$ABERAC.A    fromlib BASE.DOS  tolib BASE.OPTDOS rc=000
```

   The return codes are analyzed if the ongoing translator build step will be flagged as successful or unsuccessful.
6. Generate the reference member

   This member contains at least the same information shown step 5, but may contain also additional packaging information found in architecture parameters and/or the environment file.

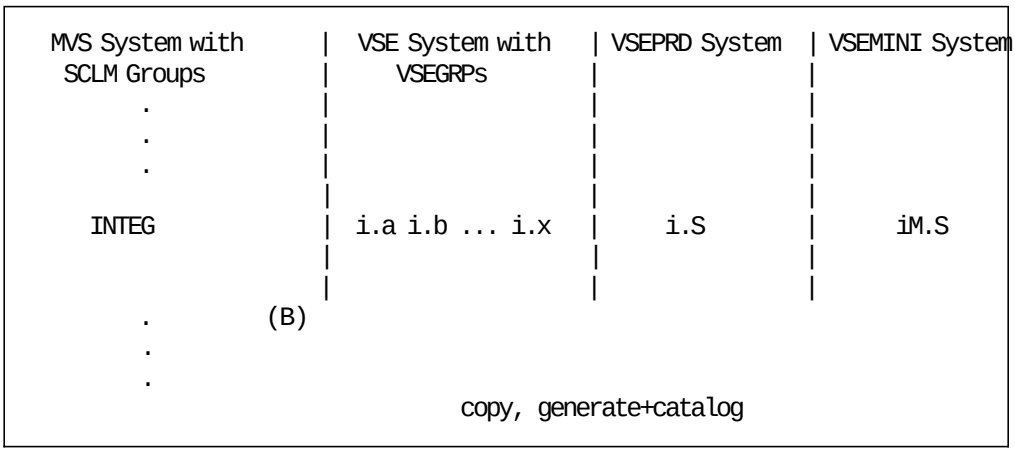
   The reference file is used to document what members on the VSE side are stored and are under SCLM control. The member is used during promote time. Therefore, the member has a language associated with it which contains the promote translator steps to investigate which members must also be promoted on VSE.

5.2.7.2 Process on the VSE System

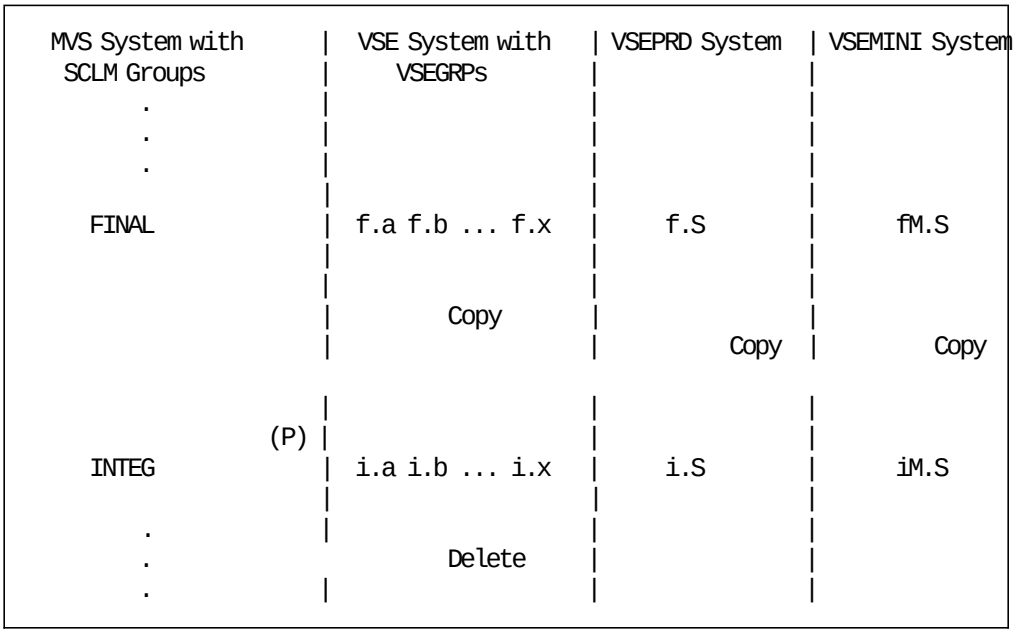
1. Receive and execute job

   The job arrives at VSE in a hot reader, which means it executes without manual interaction.

During an SCLM build at least the VSEGRP libraries on the same group level are updated. Dependent on the parameters set also the VSEPRD and VSEMINI libraries are updated.



During an SCLM promote at least the VSEGRP libraries on the next group level are updated. Note that during the promote the VSEPRD and VSEMINI libraries are not updated. They are updated using a cumulative technique and therefore no delete must take place on those systems.



2. Create feedback for waiting build on MVS

   Collect RCs via SETPARM and REXX execs. This step now collects the output and analyzes it to create the response information to be sent back to the waiting SCLM build step in MVS.

3. Punch feedback job to MVS so that the waiting build may proceed. See step 4 on page 52.

Chapter 6. SCLM Project Characteristics

6.1 Types of SCLM Projects

We not only use SCLM to manage our product development environments as described in 3.1, “BPL Development Environment to Manage Products” on page 13.

We use SCLM also to:

- create product SCLM projects. This is described in 6.2, “Meta SCLM Projects to Develop SCLM Projects” on page 59.
- create and maintain necessary programs and tools. This is described for REXX in 7.1, “REXX Program Development” on page 67.
- create documentation as described in 7.2, “Documentation Development Environment” on page 70.

Overall we established a four level SCLM library sequence starting with one very small seed SCLM library and then created all the other SCLM controlled development environments under SCLM control. This is described in Figure 12 and Figure 13 on page 57 which shows the interaction between the different SCLM project setups.

It starts with a first level SCLM project and goes up to the third level SCLM project. These are all meta projects. The sequence ends in a fourth level SCLM project which are the product development projects described in 3.1, “BPL Development Environment to Manage Products” on page 13.

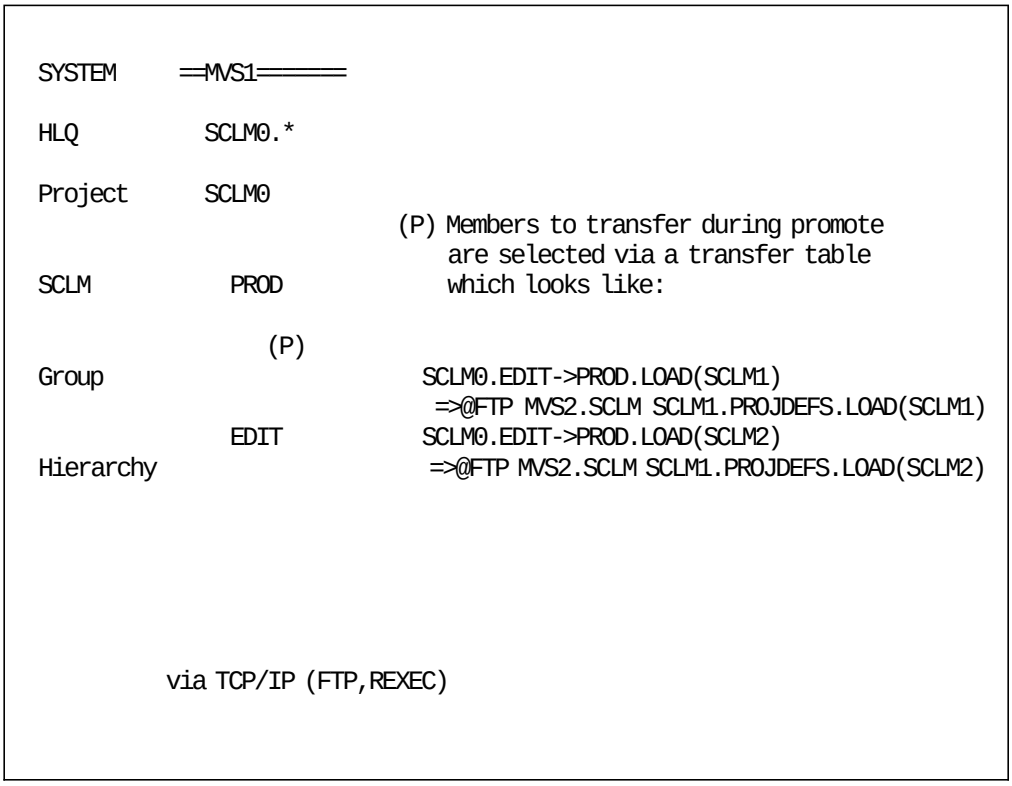


Figure 12. Overview of Used SCLM Structures in BPL (First Level SCLM Project)

First level SCLM project: Seed project

This level is shown in Figure 12. Our SCLM seed project is located on the MVS system MVS1 under the high-level qualifier SCLM0, the project SCLM0 and just has two groups EDIT and PROD. It has the languages SCLMMOD, SCLMMAC and ARCHDEF and is very limited in its functional scope to keep it simple, because the definition members are not under any SCLM control (it is at this point the hen-egg-problem). The names of the two generated projects are SCLM1 and SCLM2.

Because these generated projects run under another MVS system (MVS2) with high-level qualifier SCLM1, they must be copied to this system. This is done by a promote from group EDIT to PROD using our copy translator technique described in 8.3.2, “SHIPIT - Externalize Data during Promote Time” on page 88. For this case we invoke, through the copy translator, the TCP/IP services FTP and REXEC to do the copy of the load module members across two MVS systems:

- *SCLM0.EDIT.LOAD(SCLM1) on MVS1 to SCLM1.PROJDEFS.LOAD(SCLM1) on MVS2*
- *SCLM0.EDIT.LOAD(SCLM2) on MVS1 to SCLM1.PROJDEFS.LOAD(SCLM2) on MVS2*

Other parts are also copied as described in 6.2.1, “hlq.PROJDEFS.* Data Sets” on page 62.

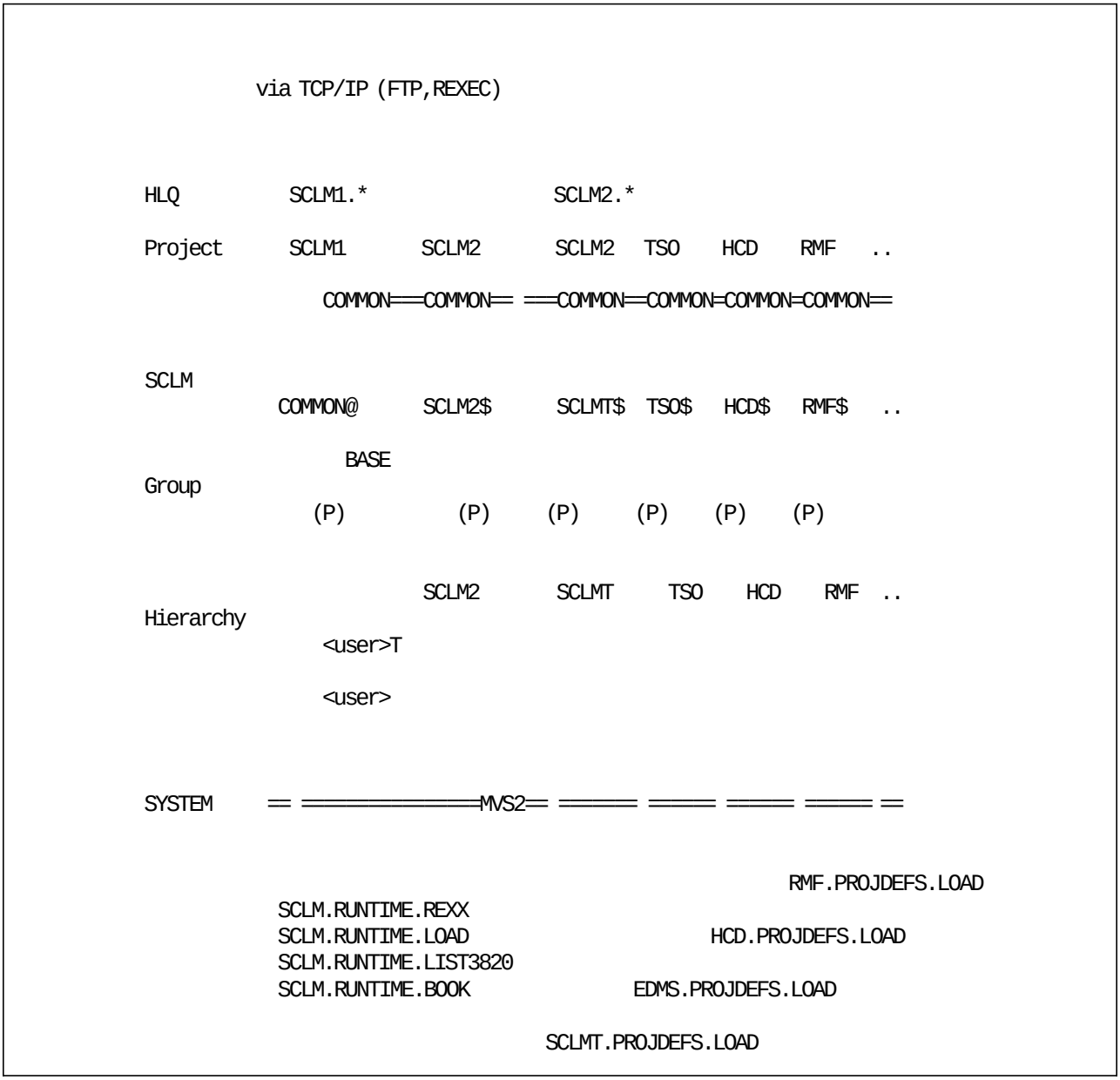


Figure 13. Meta SCLM Projects (Second and Third Level SCLM Project)

Second level SCLM project: SCLM setup environments

This level is shown in the left part of Figure 13. The projects described now are SCLM1 and SCLM2.

They are created as described from the first level SCLM project and reside on the MVS system MVS2 under the high-level qualifier SCLM1. These are the SCLM development projects for the SCLM support team. For its responsibilities see 3.2.2, “SCLM Support Team” on page 17.

SCLM1 This project is used to develop all the translator functions, tools, additional documentation, commonly used SCLM translator macros, and so on, to maintain and provide the base for the SCLM product environments.

We have two types of delivery mechanism for parts developed in this project.

- In general parts are only promoted to the group BASE. During the promote to this group the distribution to the SCLM run-time environment, on which all the other project environments rely, is done using the copy translator technique described in 8.3.2.1, “Default Action - Local Copy to Data Sets” on page 89. As shown in Figure 13 on page 57 they are copied normally to the data set starting with *SCLM.RUNTIME.**.
- However such parts as the SCLM translator macros (see Appendix B, “Translators” on page 107) are either created and maintained through group COMMON@ or BASE and promoted to group COMMON. The COMMON group data sets are shared between **all the projects** found in the high-level qualifiers SCLM1 and SCLM2, but are read-only except for project SCLM1 with high-level qualifier SCLM1. The protection is done using RACF and the SCLM authorization codes.

SCLM2 This project under hlq SCLM1 is used similarly to the first level project to create other SCLM projects (the third level). This project is maintained also by the SCLM support team, while the third level projects are the SCLM development environment for the integration teams.

In this project the project definitions to be created are identical in most cases, except for the project names of projects to be created.

The working groups are SCLM2 and SCLM2$. When a project definition is promoted from group SCLM2 to SCLM2$ the copy translator technique described in 8.3.2.1, “Default Action - Local Copy to Data Sets” on page 89 is used to copy the project definitions to the target data sets starting with *SCLM2.PROJDEFS.**.

Things that are copied for project HCD are:

```
* ____________________________________________________________
* HCD  project of HLQ SCLM2
*  to create and SCLM Development Environment for Product HCD
*
 SCLM1.SCLM2->SCLM2$.SCLMLOAD(HCD) =>SCLM2.PROJDEFS.LOAD
 SCLM1.SCLM2->SCLM2$.SCLMLIST(HCD) =>SCLM2.PROJDEFS.LOADLIST
*
*  Data set allocation information
 SCLM1.SCLM2->SCLM2$.SCLMJCL (HCD) =>SCLM2.PROJDEFS.ALLOCDEF
*
*  User and administration guide
 SCLM1.SCLM2->SCLM2$.SCLM3820($HCD)=>SCLM2.PROJDEFS.LIST3820
 SCLM1.SCLM2->SCLM2$.SCLM3820(#HCD)=>SCLM2.PROJDEFS.LIST3820
 SCLM1.SCLM2->SCLM2$.SCLMBOOK($HCD)=>SCLM2.PROJDEFS.BOOK
 SCLM1.SCLM2->SCLM2$.SCLMBOOK(#HCD)=>SCLM2.PROJDEFS.BOOK
```

For the meaning of the data sets see 6.2.1, “hlq.PROJDEFS.* Data Sets” on page 62.

Third level SCLM project: SCLM project definition environments

These projects are used to create the product development environments for the project development teams. There product integrator will do the definitions for the fourth level. For the responsibilities see 3.2.3, “Project Integration Team” on page 19.

It is shown in the right part of Figure 13 on page 57. The projects are TSO, HCD and RMF.

Besides the generated members as already described in the second level project SCLM2 some more are provided under *SCLM2.PROJDEFS.**. These are:

- SCLM2.PROJDEFS.INFOS
- SCLM2.PROJDEFS.ISPSLIB
- SCLM2.PROJDEFS.RACFDEF
- SCLM2.PROJDEFS.REXX
- SCLM2.PROJDEFS.SHIPLIST
- SCLM2.PROJDEFS.SHIPLOG
- SCLM2.PROJDEFS.VSAMCNTL

For the meaning of the data sets see 6.2.1, “hlq.PROJDEFS.* Data Sets” on page 62.

Fourth level SCLM project: SCLM product development environments

These projects are used to create the products to be shipped to the customer and by the product integrators and the project development teams. For the appropriate responsibilities see 3.2.3, “Project Integration Team” on page 19 and 3.2.4, “Project Development Team” on page 20.

However there is one high level qualifier SCLMT that is used by the SCLM support team to have a test environment to test all the developed languages, translator macros, translator functions in the same way as the products would be used in their SCLM controlled environment.

6.2 Meta SCLM Projects to Develop SCLM Projects

This chapter explains all common components of the meta SCLM projects sketched out in 6.1, “Types of SCLM Projects” on page 55. The purpose of meta SCLM projects is to create the SCLM environment to establish another SCLM project. One example of a meta SCLM project is listed as a user′s and administration guide in Appendix F, “VSE Project View under HLQ SCLM2 in Source Edit Format” on page 169 and Appendix G, “Sample Meta Project Documentation (Formatted Output)” on page 173, the implementation is in Appendix H, “Sample Meta Project Implementation” on page 187.

Setup requirements

For all SCLM projects there is a set of environment data sets provided. The overall set is located on qualifier SCLM.RUNTIME.* and contains the basic parts to establish the common SCLM environment for all projects. The data sets are allocated at logon time for each SCLM user.

SCLM.RUNTIME.LOAD
: Contains load modules needed for the environment

SCLM.RUNTIME.REXX
: Contains REXX programs needed for the environment

SCLM.RUNTIME.ISREDIT
: Contains ISPF edit macros needed for the environment

SCLM.RUNTIME.ISPSLIB
: Contains ISPF skeletons needed for the environment

SCLM.RUNTIME.Vxxx.ISPSLIB
: Contains ISPF skeletons needed for the special ISPF versions. This means if a user runs with ISPF 4.1 he is allocated

SCLM.RUNTIMEV410.ISPSLIB. A user running with ISPF 4.2 is allocated SCLM.RUNTIME.V420.ISPSLIB.

The second set is described in 6.2.1, “hlq.PROJDEFS.* Data Sets” on page 62 and is allocated dependent on the project to be used.

Starting with these data sets, the user has to invoke a REXX exec described in D.9, “SCLMSITE - Basic SCLM Project Environment Setup” on page 151 during logon that will do the allocations for his project and the ISPF release dependent allocation.

If we have to run two or more ISPF releases in parallel for one project we use the technique that one project definition, let’s say VSE, is established, but then use two members, VSEI41 and VSEI42, which just have a COPY statement on member VSE.

Member VSE VSE, Language=SCLMMAC

```
* Member contains project definition
*
*     for project VSE
*     on  HLQ     VSE
*     of  System  MVS2
*_____________________________________
      FLMABEG
...
      FLMAEND
```

Member VSEI41 VSE, Language=SCLMI41

```
* Member to create project definition
*
*     for project VSE
*     on  HLQ     VSE
*     of  System  MVS2
*
* for a ISPF 4.1 environment
*_____________________________________
      COPY VSE
```

Member VSEI42 VSE, Language=SCLMI42

```
* Member to create project definition
*
*     for project VSE
*     on  HLQ     VSE
*     of  System  MVS2
*
* for a ISPF 4.2 environment
*_____________________________________
      COPY VSE
```

Member VSEI41 has the language SCLMI41 and member VSEI42 has the language SCLMI42. The languages themselves differ only in the SCLM macro libraries used. With the normal build technique two load modules are then created, SCLMI41 LOAD and SCLMI42 LOAD. With our promote exit described in 8.2.2, “Product Packaging” on page 85 we then have the following entries in the shiplist:

SHIPLIST format

```
SCLM2.VSE->VSE$.SCLMLOAD(VSEI41)  => VSE.PROJDEFS.LOAD
SCLM2.VSE->VSE$.SCLMLOAD(VSEI42)  => VSE.PROJDEFS.LOAD
SCLM2.VSE->VSE$.SCLMLOAD(VSEI42)  => VSE.PROJDEFS.LOAD(VSE)
```

where VSEI42 is copied twice with a different name to provide the default environment so that a user does not have to specify the alternate project name.

Create an SCLM meta project

How to do this is described in Appendix G, “Sample Meta Project Documentation (Formatted Output)” on page 173. However for better understanding some additional explanation follows.

As mentioned, for the project definition we use not just one type but four types. The reason for this is to show clearly the level of responsibilities and have an easier use of the member list of the SCLM option 3.1 in the SCLM dialog.

The first type <hlq>

is typically the same as the high-level qualifier for the project to be created. For example for a project to be established under the high level qualifier VSE, the SCLM data set is in our environment SCLM1.SCLM2.VSE under the alternate project SCLM2. In this type only project definition master files are found. This gives then a list of project definitions for a target high-level qualifier.

The associated CC architecture type is SCLMDEFC and the associated LEC architecture type is SCLMDEFL.

The second type is SCLMVIEW

Here all the members are stored that are referenced by project definition members in type <hlq>. They define the organizational aspects and not the processing aspects. These members are normally only subject for reuse within one project for alternate project views.

Naming conventions (see G.5.3, “Develop SCLM Projects” on page 180) are used for the different definitions. Up to this type all members are updated and provided by the maintainer of the target project. This means the integration team is normally responsible for third level project as described in 6.1, “Types of SCLM Projects” on page 55, whereas the SCLM support team will be responsible for the other level.

The associated architecture type is SCLMDEFV.

The third type is SCLMLANG

and contains only members defining SCLM languages. The members have the same name as the language defined. Starting with this type, some of the members are subject for reuse.

This means the SCLM support team provides some language definitions which are common across projects where the integration teams may modify those or provide their own, which can be subject for reuse again. This means new languages are investigated by the SCLM support team if they are moved to the COMMON group and removed from the project scope.

The associated architecture type is SCLMDEFS.

The fourth and final type is SCLMMACS

and contains only members that are translator definitions in copy or macro format. For naming conventions refer to G.5.3, “Develop SCLM Projects” on page 180. These members are subject to high reuse only so that normally the integration team never creates their own and uses only the parts provided by the SCLM support team in group COMMON. However because there are always exceptions, it is possible for them to create their own definition if they have to.

The associated architecture type is SCLMDEFS.

In the described SCLM project types the members do not only contain the SCLM definitions (SCLM macros FLMxxxxx) but also the necessary user and administration information of the project and additional data set allocation information for each type.

The formats are shown in:

- Appendix G, “Sample Meta Project Documentation (Formatted Output)” on page 173
- Appendix F, “VSE Project View under HLQ SCLM2 in Source Edit Format” on page 169
- Appendix H, “Sample Meta Project Implementation” on page 187

Because this information is provided together with the pure SCLM definition, it allows us not only to create the project load module for SCLM with the build but also in one step create the user′s guide and the administration guide and also a data set allocation file that then can be used to generate all the needed data sets for one project if needed.

The documentation technique is described in 7.2, “Documentation Development Environment” on page 70. The allocation technique is described in D.6, “SCLMINFO - Generate Data Set Allocation Information” on page 143.

6.2.1 hlq.PROJDEFS.* Data Sets

The following list shows all the hlq.PROJDEFS.* data sets that each of our SCLM projects will have and describes their purpose.

Members created in an SCLM project

The members of the following data sets are created in an SCLM project and are transferred using the promote exit described in 8.3, “Delivery Processes” on page 88. Exceptions are the members in the seed project (first level).

hlq.PROJDEFS.LOAD

The project definition load modules for the hlq. The members are created in a lower level SCLM project.

hlq.PROJDEFS.LOADLIST

The assembler listing of the project definition module(s).

hlq.PROJDEFS.ALLOCDEF

The allocation information member(s) produced at the same time as the project load module. The generation is described in D.6, “SCLMINFO - Generate Data Set Allocation Information” on

page 143. This information is used to do the allocation of data sets needed for the associated project under the high-level qualifier hlq. It is input to the tool described in D.7, “SALLOC - Allocate Data Sets” on page 147 to do all the needed allocations.

hlq.PROJDEFS.LIST3820
The project definition documentation in printable format. Two versions are provided. A Users Guide and an Administration Guide. The Administration Guide is actually the Users Guide plus some implementation information and the SCLM definition statements. The technique to create this documentation is shown in 7.2, “Documentation Development Environment” on page 70.

hlq.PROJDEFS.BOOK
The project definition documentation in on-line format to be displayed using BookManager. The technique to create this documentation is shown in 7.2, “Documentation Development Environment” on page 70.

Members created outside of a SCLM project
The members of the following data sets are created outside an SCLM project and edited there and are therefore not under SCLM control. However, most of them can be SCLM controlled as well. It is a matter of trade off between control, history, backup and usage.

hlq.PROJDEFS.INFOS
Information to be displayed at logon time to the user of the project or some other project related information. This is done by D.9, “SCLMSITE - Basic SCLM Project Environment Setup” on page 151.

hlq.PROJDEFS.ISPSLIB
ISPF skeletons used just for projects of this hlq. The data set will be allocated to ISPSLIB during logon time. It contains ISPF release independent members. See also 6.2.2, “Modified Skeletons for SCLM in Batch” on page 64.

hlq.PROJDEFS.Vnnn.ISPSLIB
ISPF skeletons used just for projects of this hlq. The data set will be allocated to ISPSLIB during logon time. It contains ISPF release dependent members where nnn indicates the release. For example for ISPF 4.1, the name would be hlq.PROJDEFS.V410.ISPSLIB. Typically it contains the modified ISPF skeletons to run SCLM functions in batch. Members we modified are for example, FLMB$, FLMDSU$. See also 6.2.2, “Modified Skeletons for SCLM in Batch” on page 64.

hlq.PROJDEFS.RACFDEF
Contains members with the RACF definition statement to establish the RACF layer(s) for the projects under the hlq. A sample is provided in Figure 3 on page 23.

hlq.PROJDEFS.REXX
REXX programs used just for projects of this hlq. The data set will be allocated to SYSEXEC during logon time.

hlq.PROJDEFS.SHIPLIST

Members containing ship information during promote time. It is used in the process described in 8.3.2, “SHIPIT - Externalize Data during Promote Time” on page 88.

hlq.PROJDEFS.SHIPLOG

One member exists for each member in hlq.PROJDEFS.SHIPLIST holding the log information if a shipment is done during promote time.

hlq.PROJDEFS.VSAMCNTL

A set of JCL members to do some VSAM accounting data set analysis and recovery and reorganization. Some of those members are listed in Appendix E, “VSAM Control File Utility Jobs” on page 161.

6.2.2 Modified Skeletons for SCLM in Batch

As described in the SCLM documentation, at least the skeleton FLMLIBS must be modified to do the standard allocation for the ISPF to be used in batch. It also should contain any allocations which are user and project independent. In our case it just contains the allocation for an ISPF environment.

```
//* ================================================================
//* == Start of  SCLM.RUNTIME.V410.ISPSLIB(ISPLIBS)
//* ================================================================
//*****************************************************************
//* ISPF PRODUCT LIBRARIES
//*****************************************************************
//*
//STEPLIB    DD DSN=SYS4.ISPF.V410.SISPLOAD,DISP=SHR
//ISPLLIB    DD DSN=SYS4.ISPF.V410.SISPLOAD,DISP=SHR
//ISPMLIB    DD DSN=SYS4.ISPF.V410.SISPMENU,DISP=SHR ISPF/PDF MSGS
//ISPSLIB    DD DSN=SYS4.ISPF.V410.SISPSENU,DISP=SHR PDF SKELS
//ISPPLIB    DD DSN=SYS4.ISPF.V410.SISPPENU,DISP=SHR PDF PANELS
//ISPTLIB    DD DSN=SYS4.ISPF.V410.SISPTENU,DISP=SHR PDF TABLES
//SYSEXEC    DD DSN=SYS4.ISPF.V410.SISPEXEC,DISP=SHR
//ISPTABL  DD  UNIT=&VIOUNIT.,DISP=(NEW,PASS),SPACE=(CYL,(1,1,5)),
//             DCB=(LRECL=80,BLKSIZE=19040,DSORG=PO,RECFM=FB),
//             DSN=&TABLESP                  TEMPORARY TABLE LIBRARY
//*
//ISPPROF  DD  UNIT=&VIOUNIT.,DISP=(NEW,PASS),SPACE=(CYL,(1,1,5)),
//             DCB=(LRECL=80,BLKSIZE=19040,DSORG=PO,RECFM=FB),
//             DSN=&TABLESP                  TEMPORARY TABLE LIBRARY
//*
//ISPLOG   DD  SYSOUT=*,
//             DCB=(LRECL=120,BLKSIZE=2400,DSORG=PS,RECFM=FB)
//*
//ISPCTL1  DD DISP=NEW,UNIT=VIO,SPACE=(CYL,(1,1)),
//            DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)   TEMPORARY FILE
//*                                                 TAILORING DATA SET
//*****************************************************************
//*
//*---------------------------------------------------------------
//* TEMPORARY CLIST CONTAINING COMMAND TO BE EXECUTED
//*---------------------------------------------------------------
//SYSPROC  DD  DSN=&&&&CLIST&STEP,DISP=(OLD,DELETE)
```

```
//         DD  DSN=SYS4.ISPF.V410.SISPCLIB,DISP=SHR CLIST LIBRARY
//*
//* ================================================================
//* == END   of  SCLM.RUNTIME.V410.ISPSLIB(ISPLIBS)
//* ================================================================
```

For the build and promote, (members FLMB$, FLMP$) we usually only change such things as the region size used for the TSO environment and the size of the data sets for the LISTnn.

We also have to change the generation of the job statement if we run on a JES3 system. The reason for this is that we have to ensure that all SCLM activities run on one physical machine to avoid VSAM synchronization problems. To trigger this automatically, we added the MAIN SYSTEM=JGLOBAL card in the member FLMDSU$.

```
)CM * GLOBAL JOBCARD SKELETON                                          *
)CM * FUNCTION: THIS SKELETON IS USED TO CONSTRUCT THE JOB STATEMENTS. *
)CM ************************************************************************
&XJOBC1
)SEL &XJOBC2  = &Z
&XJOBC2
)ENDSEL
)SEL &XJOBC3  = &Z
&XJOBC3
)ENDSEL
)SEL &XJOBC4  = &Z
&XJOBC4
)ENDSEL
//*MAIN    SYSTEM=JGLOBAL       MAIN CARD INSERTED FROM MEMBER FLMDSU$
```

For the build and promote, (members FLMB$, FLMP$) we usually only change such things as the region size used for the TSO environment and the size of the data sets for the LISTnn. However we add a)IM SCLMSITE statement before the ISPSTART statement as below:

```
...
/*
//BUILD&STEP..SYSTSIN  DD *
)CM INCLUDE NOW THE BPL ALLOCATION INVOCATION FOR THE PROJECT
)IM SCLMSITE
  ISPSTART CMD(%TEMPNAME)
/*
```

The contents of the member appear so:

```
)CM Contains the logic to do the allocates for BPL projects in batch
)CM independent of ISPF releases
)CM
)CM The variable &SPRJ1 must already be set to the project from
)CM where the batch job is invoked. The variable is normally set by
)CM the SCLM dialog.
)CM
)CM Change activity:
)CM
)CM 940412 JP created
)CM ____________________________________________________________________
  EXEC ¢SCLM.$ASIS$.ALLOC(SCLMSITE)¢ ¢PROJECTS(&SPRJ1)¢ EXEC
```

This enables us to establish, in the same way as in foreground, the necessary environment because it calls the same exec that does all the allocations (refer to D.9, “SCLMSITE - Basic SCLM Project Environment Setup” on page 151).

This technique must be done for all SCLM batch functions that rely on some allocations being done before invocation of an SCLM function. These are typically only the build and the promote because of the translators to be invoked.

Chapter 7. Development

7.1 REXX Program Development

Because most of our translator functions are written in REXX we also develop them and maintain the sources in an SCLM controlled environment. Not only this, we also established a reuse technique for the REXX program development.

We maintain a set of REXX macros which are essentially members containing commonly used code across several REXX execution members. They are stored in the type REXXMAC.

Example of a REXX macro

```
/*REXX: $LIST REXXMAC */
!LIST_new: procedure expose !.
...
return lid

!LIST_append: procedure expose !.
parse arg lid, data
...
return count

!LIST_read: procedure expose !.
parse arg lid, ix
...
return data

...
```

The body of the different execution members are stored in the type REXXMOD.

Sample of a REXX module using a REXX macro

```
/*REXX: SAMPLE REXXMOD */
lid = !LIST_new()
do i = 1 to 5
  xx = !LIST_append(lid,†This is line† i)
end
do i = 5 to 1 by -1
  say !LIST_read(lid,i)
end
exit

%im $LIST
```

Now we have two languages, REXXMOD and REXXMAC, which in our case have the same names as the previously mentioned types. The REXXMOD language will create output members of type REXX from the REXXMOD members (for example, SAMPLE REXXMOD). A REXX translator function is used that will basically include members from type REXXMAC (for example, $LIST REXXMAC) if functions of such members are used in the REXXMOD member SAMPLE.

In the REXXMOD members there are imbed statements of REXXMAC members. Because we allow REXXMAC members to imbed statements also, we have written a parser in the languages REXXMAC and REXXMOD to search for the dependencies.

With this approach we do a lot of reuse by imbed and no longer by copy as was done in the past. This gives us the benefit of building up our own function library and standards and fixing problems in one place thereby improving quality and performance.

Additionally we use the same documentation technique also for our REXX execs as described in 7.2, “Documentation Development Environment” on page 70. This gives us, besides the already described benefits, hard copy and on-line documentation of our REXX based translator functions.

Currently we have established the following sets of reusable functions:

$list Contains a set of functions to maintain an abstract datatype of a list and to perform actions on it.

```
!list_new          - Initialize a new list to store data
!LIST_drop         - Drop a list so that storage can be reused later on
!LIST_debug        - Debug the LIST routines defined by flags
!LIST_reset        - Reset the LIST to a zero count
!LIST_lines        - Return the number of elements currently in a LIST
!LIST_concat       - Concat two lists to one
!LIST_append       - Append data to a LIST
!LIST_read         - Read element from a LIST
!LIST_stem         - Return the LIST stem for a given LIST
!LIST_member_list - Get the member list of a DA in LIST
!LIST_from_member - Fill a new LIST with PDS member lines as elements
!LIST_from_ddname - Fill a LIST from a ddname allocated member
!LIST_to_member    - Write the LIST contents to a PDS member
!LIST_to_ddname    - Write LIST to a ddname file
!list_dd_allocs    - Creates a LIST for da_names to a given dd_name
```

$TSO Contains a set of functions to ease the use of TSO commands and related debugging.

```
!tso                      - function to cover TSO service calls with
                            rc handling
!tso_copy_load            - Do a copy of load modules from one PDS to another
!tso_linkmvs              - LINKMVS service calls with rc handling
!tso_address              - Issue an ADDRESS request for TSO environment
!tso_file_is_there        - Check if a file is there and return properties
!tso_file_infos           - Provide the file info
!tso_copy                 - Do a copy of members from one PDS to another
!tso_alias                - Create ALIAS members
!tso_iebcopy              - Invoke the IEBCOPY service
!tso_iebgener             - Invoke the IEBGENER service
!tso_ddlist               - Provide an alternate ddlist to be passed
                            to a program
!tso_listalc              - Return allocated data set to ddnames back in lists
!tso_sysprint             - Handle the SYSPRINT allocation and free
!tso_iewl                 - Invoke the linkage editor
!tso_file_new             - Allocate new file(s)
!tso_stem_to_file         - Read the contents of a stem into a file
!tso_file_to_stem         - Read the contents of a file into a stem
!tso_append_stem_to_file  - Append the contents of stem to a file
```

$ISPF Contains a set of functions to ease the use of ISPF functions and related debugging.

```
!ispf_copy_member - Copy a member from one PDS to another using LMCOPY
!ispf_skel_get    - Read a skeleton into a temporary data set
!ispf             - Function to cover ISPF service calls with rc handling
```

$MSGS

Contains a set of functions to issue messages to terminal and/or files.

```
!msg - Write out messages to terminal
```

$PARSE

Contains a set of functions that are used for different parsing of sources.

```
!parse_SCLM_parms - Parse the SCLM related parameters
!parse_dsname     - Parse the next token as dsname
!parse_parm       - Parse a parameter out of a string
```

$STRING

Contains a set of string functions that are used in addition to the provided REXX string functions.

```
!string_replace - Replace a text in a string with another text
```

$REXX

Contains a set of functions to enhance the REXX programming in general, such as debugging and error analysis.

```
!rexx_init               - Set the REXX diagnosis and access to variables
!rexx_trap_on_code_error - Stop if code error found by REXX interpreter
!rexx_trap_on_call_error - Stop if runtime error found by REXX interpreter
```

$TREE

Contains a set of functions to manage and manipulate tree structures.

```
!tree_new          -  Create a new tree item
!tree_drop         -  Drop a tree
!tree_debug        -  Do debugging on tree functions
!tree_debug_set    -  Set debugging on tree functions
!tree_dump         -  Dump the tree contents for diagnosis
!tree_display_next -  Display next element of a tree element
!tree_element_add  -  Add an element to a tree
!tree_elements     -  Return number of elements in the tree
!tree_element      -  Return an element of a tree
!tree_init         -  Initialize a tree
!tree_make         -  Create the tree structure explicitly
!tree_roots        -  Return the routs found of a tree element
!tree_down         -  Go down the tree
!tree_type         -  Return the type of an element in a tree
!tree_up           -  Go up the tree
!tree_level        -  Return the level of a tree element
!tree_display      -  Display a tree
!tree_sample       -  Invoke a sample of a tree generation
```

7.2 Documentation Development Environment

The following documentation related to product development is created:

- Program end user documentation
- Program specification
- Program design
- Workbooks
- Program documentation
- Presentations
- Any type of plans

Most of them are created using BookMaster. BookMaster is a tag language which therefore needs a “compile” after editing as any programming language to create the printable output or on-line documentation (using BookManager).

Therefore, this type of documentation is a candidate to be managed within SCLM using the Build technique. We use a special technique to provide and maintain documentation and program source in the same file. We do this today for the SCLM definitions in assembler format and our REXX programs.

How such BookMaster sources are managed and documentation is created out of an assembler source is described in the following chapters. Sample documentation is show in:

- Appendix F, “VSE Project View under HLQ SCLM2 in Source Edit Format” on page 169
- Appendix G, “Sample Meta Project Documentation (Formatted Output)” on page 173
- Appendix H, “Sample Meta Project Implementation” on page 187

We added a real time example to the document to show what can be achieved with such an approach.

The source Appendix F, “VSE Project View under HLQ SCLM2 in Source Edit Format” on page 169 was input to the REXX2LLD program. This then created one file which is shown in this document under H.1, “Member VSE SCLM2” on page 187 and G.2, “VSE Project View under HLQ SCLM2” on page 176. If the reader compares those sections he will see how they relate to each other.

7.2.1 Benefits of our Documentation Approach

- The document sources are treated as program source
- Documents follow the same principles as any program source. Instead of a compiler other translators are used:

  BookMaster to create printable documents
  BookManager to create on-line documents that can be read on different operating systems, such as OS/2, MVS and VM. We distribute using FTP of TCP/IP in a REXX translator function to store the on-line information in the appropriate places.

- Documentation is part of the program source
  - Documentation is closer to the program logic in both senses, physically and context related
  - Documentation is automatically extracted during the build step
  - The generated BookMaster source then gets used in the documentation

7.2.2 Sample Code and Document Parts

Table 4 on page 72 shows the full or partial contents of all the architecture members, the editable and generated documentation sources. The REXX exec to create the generated sources is described in D.5, “REXX2LLD - Generate BookMaster Format from Sources” on page 137.

Table 3 lists all the members needed to create the documentation. This includes all the editable and all those that are generated and are then input to the document formatting process. With the provided translators we are able to create printable documents in the SCLM type LIST3820 and on-line documents in BookManager format of SCLM type BOOK.

Table 3. Sample Scenario for In-line Program Documentation. Assume we have the following ″source″ parts

Name	**Type**	**Language**	***″*Mode*″***	**Description**
PGM1	MOD	ASM	editable	Assembler program containing BookMaster symbols
MAC1	MAC	MACRO	editable	Assembler macro containing BookMaster symbols
MAC2	MAC	MACRO	editable	Assembler macro containing BookMaster symbols
MAC3	MAC	MACRO	editable	Assembler macro containing BookMaster symbols
ABC	SCRIPT	BOOKIE	editable	My master document
PGM1	SCRIPT	BOOKIE	non-editable	Output from PGM1 MOD process
MAC1	SCRIPT	BOOKIE	non-editable	Output from MAC1 MAC process
MAC2	SCRIPT	BOOKIE	non-editable	Output from MAC2 MAC process
MAC3	SCRIPT	BOOKIE	non-editable	Output from MAC3 MAC process
XYZ	SCRIPT	BOOKIE	editable	An imbed to MAC3 MAC member

Table 4. The Documentation Development

HL Architecture	LEC Architecture	CC Architecture	Source code	Generated Source
The documentation development				
IMB COMPDEF /---------- | INCL PGM1 LINKDEF | INCL MAC1 COMPDEF | INCL MAC2 COMPDEF | INCL MAC3 COMPDEF		ABC COMPDEF /---------- | INCL IMB COMPDEF | CMD .setdvcf UG on | CMD .setdvcf DES off | CMD .setdvcf code off | SINC ABC SCRIPT | OUT1 ABC LIST3820 | OUT2 ABC BOOK	ABC SCRIPT /---------- |... |:p.... |.im pgm1 |...	
			XYZ SCRIPT /---------- |.... |:p.... |:p..... |...	
The program development				
	PGM1 LINKDEF /---------- | INCL PGM1 COMPDEF | LOAD PGM1 LOAD | OBJ PGM1 OBJ	PGM1 COMPDEF /---------- | SINC PGM1 MOD | OUT1 PGM1 SCRIPT | OBJ PGM1 OBJ	PGM1 MOD /---------- |... | COPY MAC1 |*>> |*.im mac1 |*<< |...	PGM1 SCRIPT /---------- |... |:qualify.PGM1 MOD |:.xmp keep=1. | COPY MAC1 |:exmp. |:e.qualify. |.im mac1 |...
		MAC1 COMPDEF /---------- | SINC MAC1 MAC | OUT1 MAC1 SCRIPT |*	MAC1 MAC /---------- |... | COPY MAC2 |*>> |*.im mac2 |*<< |...	MAC1 SCRIPT /---------- |... |:qualify.MAC1 MOD |:.xmp keep=1. | COPY MAC2 |:exmp. |:e.qualify. |.im mac2 |...
		MAC2 COMPDEF /---------- | SINC MAC2 MAC | OUT1 MAC2 SCRIPT |*	MAC2 MAC /---------- |... | COPY MAC3 |*>> |*.im mac3 |*<< |...	MAC2 SCRIPT /---------- |... |:qualify.MAC2 MOD |:.xmp keep=1. | COPY MAC3 |:exmp. |:e.qualify. |.im mac3 |...
		MAC3 COMPDEF /---------- | SINC MAC3 MAC | OUT1 MAC3 SCRIPT |*	MAC3 MAC /---------- |... |*>> |*:p.... |*<< |... | FLMALLOC ... |*>> |*.im xyz |*<< |...	MAC3 SCRIPT /---------- |... |:p.... |:qualify.MAC3 MOD |:xmp keep=1. | FLMALLOC ... |:exmp. |:equalify. |.im xyz |...

Chapter 8. Processes

8.1 Development Processes

8.1.1 Rebase of Service to Current Development

This is a process to integrate parts changed in parallel to the ongoing release at certain points. For this purpose we established a three group hierarchy as seen in Figure 1 on page 15 or Figure 2 on page 21 for each release environment. The group hierarchy is PTFNEW -> PTFWORK -> PTF or similar names.

If there is the potential of multiple parallel work, then several such three group hierarchies must be established. This is necessary because it is assumed that only the same update process can overwrite members in group PTFNEW so as not to lose any changed code. For example two versions of the product are in service in the field. Two fixes of the same source are developed at the same time for the different versions. Both must then be retrofitted into the current release. To handle this there must be such a three level hierarchy maintained for each version in service.

How the groups are used is described in the following.

PTFNEW The members of this groups are only updated by some synchronization exits as described in 8.4.1, “Interface to Service Library” on page 93. It represents the state of changes made in other projects which are still to be integrated into the current release.

It contains not only the source but also changed architecture members if process parameters have changed.

To begin work on a member, the user creates an HL architecture member referring to the parts he has to integrate. After this he does a build on the member to verify that it still compiles on the base. Because only projects with common base code should do updates on this group and the provided members are already tested in the parallel project. If errors occur then the provider of such members must be informed that there is a potential problem between the setup or the interface of the two projects.

If the build is successful, then he does a promote to put the parts in PTFWORK. This ensures that as long as he is working on the parts his version cannot be overwritten by the same code being loaded again into PTFNEW.

PTFWORK In this group exist parts that developers are working on to integrate the changes into their currently developed release.

The developer does this with ISPF 4.2 by using the compare function of the editor or SUPERC or other techniques to find out the differences between the part in PTFWORK and the one in the release hierarchy to integrate the changes.

At the end he has the changed members in his development group.

There normally he creates an HL member with the same name as in PTFWORK that also refers to the same parts as in PTFWORK.

From then on he follows the normal procedures defined for his project to get the member via the HL member into group FINAL as seen in Figure 1 on page 15.

As soon as the HL member referring to the parts is in FINAL a promote will be issued for the HL member in PTFWORK to move the parts worked upon into the final state.

PTF This group indicates the parts for which the rework is finished.

This process today is not fully automated, because it is sufficient as it is. Depending on future problems in this process we might enhance it with promote translators from PTFNEW to PTFWORK, PTFWORK to PTF and to group FINAL to ensure that promotes can only be done without corrupting any states.

For example, a promote from PTFNEW to PTFWORK can only be done if none of the parts to be promoted already exist in PTFWORK. Also if the HL member gets promoted to FINAL then automatically the HL member in PTFWORK will be promoted too.

8.1.2 NLS Part Development

Due to that fact that most of our products have a user interface they are also translated into different languages. This means that if the base parts which are subjects for translation are ready for distribution (the development is done and the part is in group FINAL as seen in Figure 14 on page 75), the translation of the parts may start.

To have the process controlled by SCLM it is recommended to establish the following structure.

Base Release Development (Project PROD)
This is the normal SCLM project setup with the addition of the three level structure XLATEFOR -> XLATETO -> XLATEFIN. How those groups are used is shown below.

NLS Release Development (Project PRODNLS)
This is an alternate project that includes groups BASE and FINAL from the Base Release Development. However the groups FINAL, NLSBASE and BASE are protected for update across this alternate project. How the groups are used is shown below.

Similar mechanisms to those described in 8.4.1, “Interface to Service Library” on page 93 are used here.

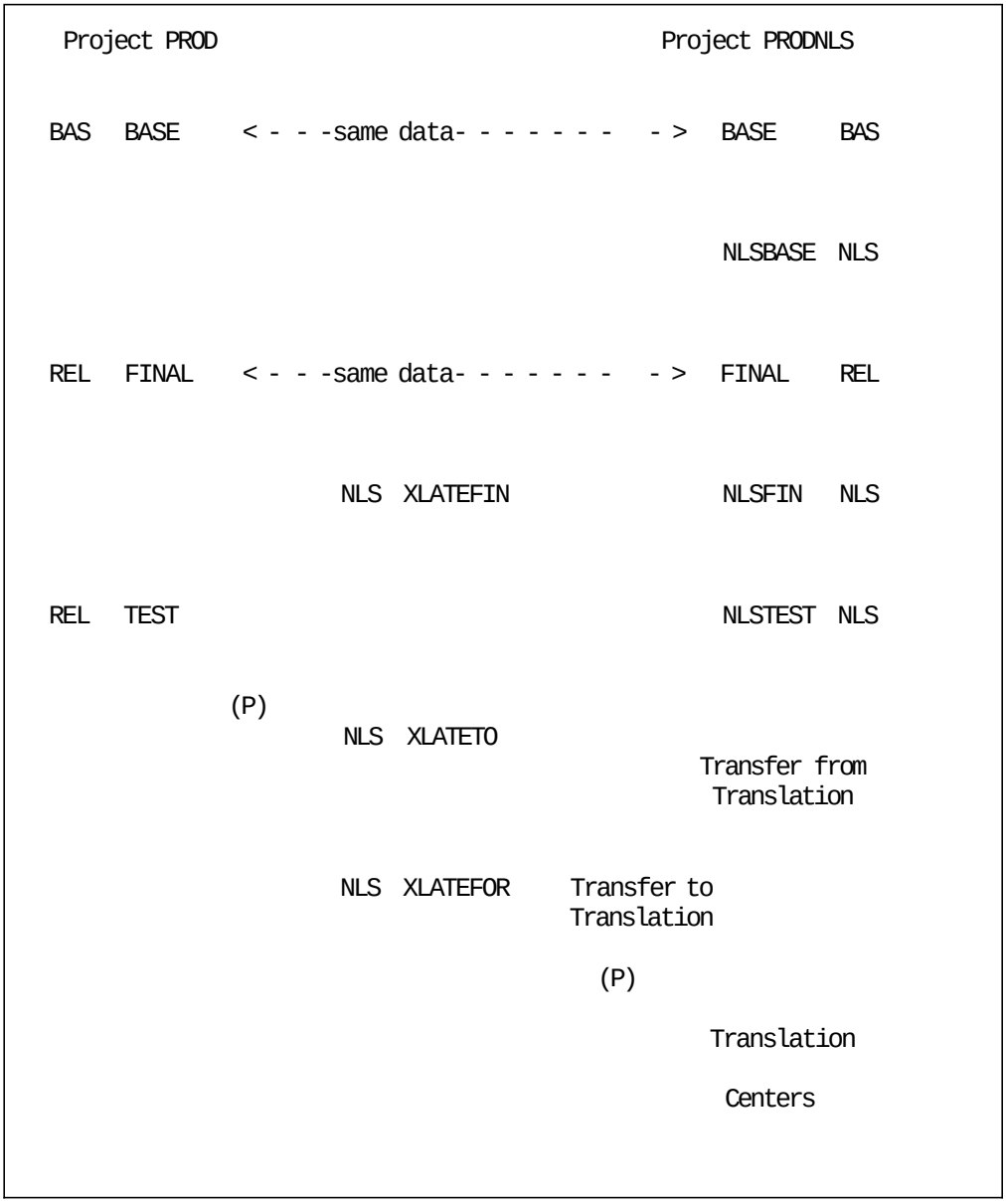


Figure 14. NLS Part Development Environment Structure

In detail, the translation process works as follows:

1. If parts are promoted from TEST to FINAL the parts which are subjects for translation are copied using a promote exit to group XLATEFOR. This is an ongoing process. There might be some verification functions built in to decide if the part must be translated again. If the contents do not differ, there is no reason to translate it again. In this case parts are not copied to the translation alternate project group.
2. The translation integrator and translation people actually work with the alternate project groups shown in Figure 14. This includes only two group chains:

   XLATEFOR -> XLATETO -> XLATEFIN -> FINAL -> BASE
   NLSTEST -> NLSFIN -> FINAL -> NLSBASE -> BASE

   For parts in XLATEFOR a work list will be created to ship parts to the translation centers. After a build is done on this work list, a promote is

issued. This promote then transfers the members to the translation centers for translation. Again a promote exit will do the transfer.

3. The group XLATETO shows all the parts in translation state.
4. As soon as the translation center has sent back all the translated members transmitted to it, which means that they arrive in the group NLSTEST, an initial test of those members is done. The test includes a build of the translated members and some verification testing.

   If the test shows problems, a new transmit will be issued from the XLATEFOR group.

   If the test shows no problems, then a promote is done to the NLSFIN group for the translation packages. At this point the same package or at least the members in group XLATETO are also promoted to XLATEFIN.
5. From NLSFIN the packaging process for the NLS product shipment is started as described in 8.2, “Packaging Processes.”

This process is also not fully automated as the process described in 8.4.1, “Interface to Service Library” on page 93. However it shows that without full automation the concepts of groups in SCLM does help to get processes organized in process steps to keep track of the different states of work to be done. The automation then is just a matter of work and investment based on a benefits/costs equation. Today just having this structure and the minimal promote technique available is a big step forward compared to the past.

8.2 Packaging Processes

SCLM does not support any implicit or explicit packaging function, however as we show in this section, it is possible to implement it using SCLM capabilities. We developed our own process to do the several packaging processes we need. Packaging in this sense means to extract data for any delivery out of the SCLM controlled library and put it in a defined format for delivery. It does not include the delivery itself, but all the data and metadata to do it. The extraction itself using the database search based on a high-level architecture member is not seen as a packaging process, but an essential part of it and is provided by SCLM.

We have implemented three important packaging processes.

SMPE Generate SMPE-Drivers (**S**ystem **M**odification **P**rogram **E**xtended) out of the SCLM hierarchy.

The process describes how we write a product release on a standard label tape, containing both the distribution libraries as unloaded partitioned data sets in RELFILE format, as well as the SMP/E system modification control statements and the JCLIN. The JCLIN is used to link the project on the customer system. See also 8.2.2, “Product Packaging” on page 85.

VPL Generate VPL formats (**V**iew **P**rogram **L**istings). VPL is an on-line process which allows users to view product program listings for MVS, VM, VSE and other IBM program products.

Our VPL process takes the generated compile listings and formats the listings in a specific VPL-format. These are

then sent to IBM Software Manufacturing who administrates the VPL listings for all projects.

This is the simplest packaging we have today because it only collects the listed members belonging to the delivered product and puts them in a data set in a specific format.

Transfer to Service Transfer developed code to the service library.

Service packaging describes the process of how we transfer developed code and all information needed for the service library. For further information see 8.2.3, “Product Transfer to Service Library” on page 87

8.2.1 Packaging Process Steps

This section describes the packaging process in general as a generic process that only differs in using different functions driven by parameters to create different formats.

Because our major and most complex packaging process is that for MVS SMP/E, we refer to the example provided in 8.2.2, “Product Packaging” on page 85.

The complete process to do the packaging is logically divided into two steps.

The first is the project build during release development based on a high-level architecture member describing the product. Apart from the normal release development data to create development drivers it contains some packaging data for different packaging formats. This is described in 8.2.1.1, “Step 1: Project Build.”

The second is the packaging build step which is implemented as an SCLM language, so that it is also started as any other build. The build is started on a packaging description file as described in 8.2.1.2, “Step 2: Package Build” on page 79.

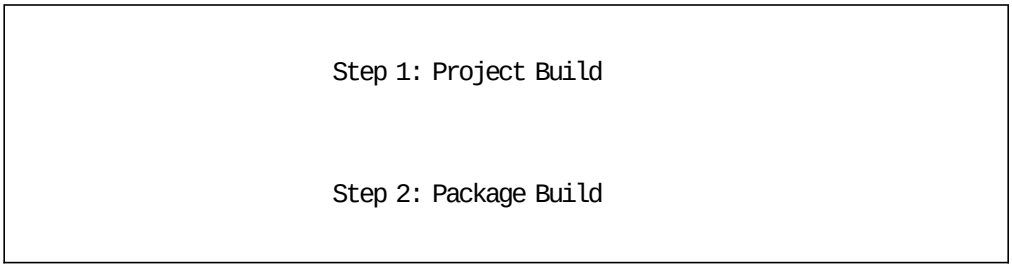


Figure 15. Packaging Process in an SCLM Controlled Project

8.2.1.1 Step 1: Project Build

This step is part of the normal release development builds. During this cycle it is used to collect and/or verify the member specific information to have an early awareness of the later packaging needs. In the past this was done afterwards and had therefore some drawbacks with regard to design and coding.

This step is optional, but it has the benefit that information which is required for the development builds and the packaging can be automatically extracted from the development specifications (such as parameters for certain processes,

linkmap information) and formatted to the packaging needs, creating a packaging information member for each member to be distributed.

This step is explained using the SMPE packaging process because it shows the principle used for all packaging processes which need to provide member related packaging information and additional members. The differences to other packaging is only the amount of additional data needed and the format. See also the sample member set described in 8.2.2, “Product Packaging” on page 85.

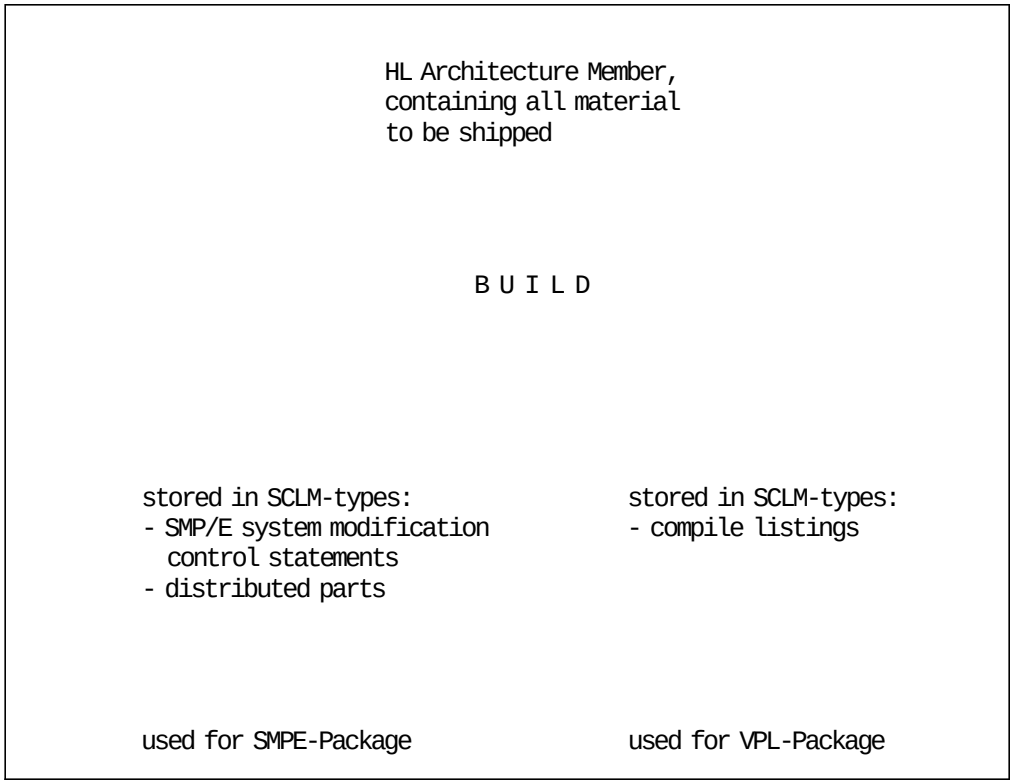


Figure 16. Step 1 - Project Build

Two essential steps are executed during SCLM project development build (see Figure 16):

1. For each part, which will be shipped on tape, one output member is created, containing the related SMP/E system modification control statement (MCS) with information about DISTLIB, SYSLIB, ALIAS names and so on.
2. All parts, packaged later for SMPE or VPL are distributed to *SCLM-distribution-types* or stored in other output types (listings).

   It is recommended that the chosen *SCLM-distribution types* match the *actual distribution libraries* (see 4.1, “Input and Output Types” on page 35).

In our example in Figure 22 on page 86 the outputs listed in Table 5 on page 79 are generated during the project build.

Table 5. Outputs Created during SCLM Project Builds	
member type	**Description**
MODAx CXXXMOD MODBx CXXXMOD	pre-linked modules used for distribution
MACA SXXXMAC	macro source used for distribution
PANA SXXXPAN	panel source used for distribution
MODAx LISTING MODBx LISTING	compile listings of modules, also needed for VPL-process
MODAx MCSMOD MODBx MCSMOD MACA MCSMAC PANA MCSPAN	members containing SMP/E system modification control statements. Example: ++MOD (MODA1) DISTLIB(CXXXMOD) CSECT(MODA1) ++MAC (MACA) DISTLIB(SXXXMAC) SYSLIB(MACLIB) ++PANLENU (PANA) DISTLIB(SXXXPAN) SYSLIB(SXXXPENU) Note: For modules the CSECT information is generated automatically by scanning the LINKMAP of the prelink-step during the build.

8.2.1.2 Step 2: Package Build

This step is divided into package generation and package delivery.

8.2.1.3 Step 2a - Package Generation

For our example, shown in Figure 22 on page 86, the *packaging description file* has the contents as seen in Figure 18 on page 80. Note that the file as listed contains only information for SMPE packaging. For VPL and Service packaging other information is needed.

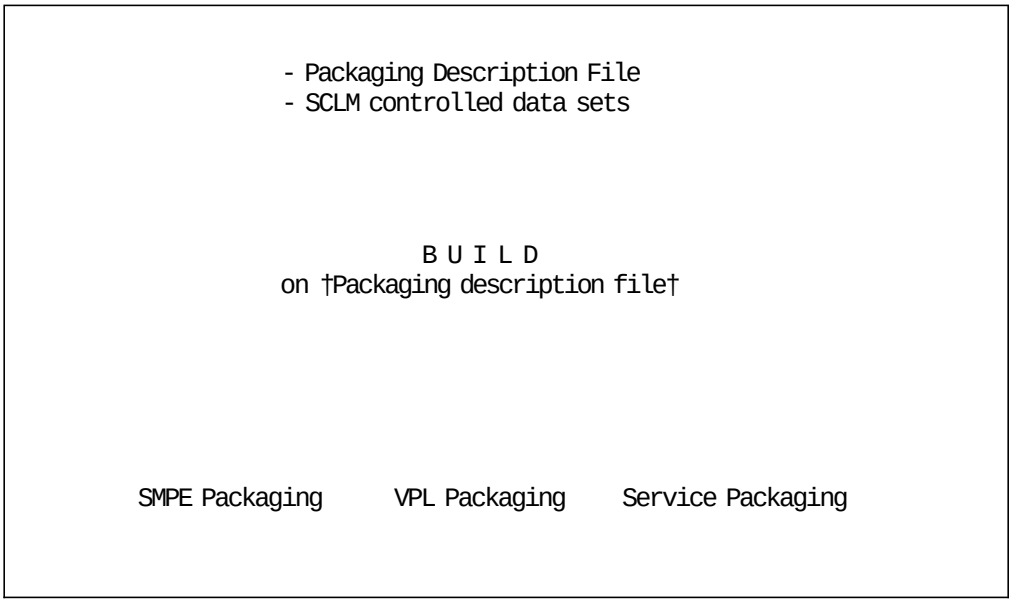


Figure 17. Step 2a - Package Generation. For the details of the different packaging steps and outputs see Figure 19 on page 84, Figure 20 on page 84 and Figure 21 on page 85.

To do the package generation, the integrator starts the packaging request issuing a build on *PRODA PKGDEF* which refers to the *packaging description file PRODA PKGDATA* (see Figure 18 on page 80). This file contains:

- Environment specifications
- SCLM project specifications, such as project name, HL architecture member of the project and so on.
- Product specifications such as FMID.
- Distribution library specifications such as RECFM, LRECL, BLKSIZE.
- Include statements of members which contain common sections used in several *packaging description file*s.

```
*-------------------------------------------------------------------------*
* Name   = TEST                                                           *
* Title  = Packaging description file                                     *
*-------------------------------------------------------------------------*
* Notes: The following keywords are used to generate an ISPF-table       *
* <X>      specification variable                                         *
* <NAMES>  column names of ISPF-table to be generated                     *
* <COLS>   position of generated columns data                             *
* <R>      one row of data of ISPF-table defined by <NAMES> and <COLS>    *
*                                                                         *
* Environment Specifications                                              *
* --------------------------                                              *
* IDRIVER - High Level Qualifier where distribution libraries are         *
*           stored                                                        *
* STORCLAS- contains storclas where Driver data sets have to be stored    *
*                                                                         *
<X> IDRIVER  SMPETEST
<X> STORCLAS SMS338
*                                                                         *
* SCLM Project Specifications                                             *
* ---------------------------                                             *
* PROJ    - SCLM Project Name                                             *
* APROJ   - SCLM Alternate Project Name                                   *
* GROUP   - SCLM Packaging level; collects packaging information from     *
*           this level up to project base                                 *
* VMETADAT- Member name of project architecture definition, building      *
*           the whole product                                             *
* VMETATYP- SCLM type of Product Architecture Definition, building        *
*           the whole product                                             *
*                                                                         *
<X> PROJ     PRODA
<X> APROJ
<X> GROUP    TEST
<X> VMETADAT PRODA
<X> VMETATYP PRODDEF
*                                                                         *
```

Figure 18 (Part 1 of 3). Packaging Description File PRODA PKGDATA

```
* Product Specifications for MCS-Header                                *
* ------------------------------------                                 *
* FMID1   - first FMID processed by the Product                         *
*           High-level Architecture Member, which is declared           *
*           in section ¢SCLM Project Specifications¢ of                 *
*           this member.                                                *
* FMID2   - second FMID processed by the Product                        *
*           High-level Architecture Member, which is declared           *
*           in section ¢SCLM Project Specifications¢ of                 *
*           this member (if it is not used, leave blank).               *
* FMID3   - third FMID processed by the Product                         *
*           High-level Architecture Member, which is declared           *
*           in section ¢SCLM Project Specifications¢ of                 *
*           this member (if it is not used, leave blank).               *
* SMPHEAD1- Data set name containing SMPMCS-Header for FMID             *
*           given in FMID1                                              *
* SMPHEAD2- Data set name containing SMPMCS-Header for FMID             *
*           given in FMID2 (if it is not used, leave blank)             *
* SMPHEAD3- Data set name containing SMPMCS-Header for FMID             *
*           given in FMID3 (if it is not used, leave blank)             *
*                                                                       *
<X> FMID1    ABC0001
<X> FMID2    XYZ0002
<X> FMID3
<X> SMPHEAD1 AAA.PACKAGE.TEST(ABC0001)
<X> SMPHEAD2 AAA.PACKAGE.TEST(XYZ0002)
<X> SMPHEAD3
*                                                                       *
* Distribution Library Specifications                                   *
*-----------------------------------------------------------------------*
* Columns= LIBRARY  SCLM type of pre-distributed data sets              *
*          FMID     related SMP/E FMID for the libraries              *
*          DISTLIB  Type of actual packaging/distribution libraries   *
*          MCSTYPE  SCLM-Type containing SMPE Modification control    *
*                   statements                                        *
*          FLAGS    Series of status flags  (see below)               *
*          RECFM    Library record format                             *
*          LRECL    Library logical record size                       *
*          BLKSIZE  Library blocksize                                 *
*          DBS      Number of directory blocks                        *
*          BLOCKS   Number of (primary) blocks                        *
*-----------------------------------------------------------------------*
```

Figure 18 (Part 2 of 3). Packaging Description File PRODA PKGDATA

```
* The FLAGS field contains a string of characters indicating the       *
* status of various options available for this library. They are:      *
*                                                                      *
* †A† - library allocated only when building drivers                   *
* †D† - library contains executable code (for drivers)                 *
* †P† - library is eligible for partlist                               *
*                                                                      *
<NAMES> LIBRARY FMID DISTLIB MCSTYPE
<NAMES> FLAGS RECFM LRECL BLKSIZE DBS BLOCKS
<COLS>  5 23 31 40 54 68 72 78 84 89
*---+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----
*   LIBRARY          FMID    DISTLIB  MCSTYPE      FLAGS       RECFM      BLKSIZE    BLOCKS
*                                                                LRECL        DBS
*
<R>
<R> DEFPKGS          ABC0001 DEFPKGS               A            FB  80    6160  0    10
<R> DEFPKGS          XYZ0002 DEFPKGS               A            FB  80    6160  0    10
*
<R> CXXXMOD          ABC0001 CXXXMOD  MCSMOD      DP           U   0     6144  100  4000
<R> SXXXMAC          ABC0001 SXXXMAC  MCSMAC      DP           FB  80    6160  10   450
<R> SXXXPAN          XYZ0002 SXXXPAN  MCSPAN      DP           FB  80    6160  10   300
```

Figure 18 (Part 3 of 3). Packaging Description File PRODA PKGDATA

The build on the *package description file* invokes an SCLM database report on a high level architecture definition member, describing the whole product. So we get a member list that describes the subset of members for distribution, a subset of members used for VPL-processing and/or a subset of members used for library transfer dependent on the specified information in the *package description file*.

The corresponding SCLM service to generate the SCLM database report based on the HL architecture member *PRODA PRODDEF* starting from group TEST is, for example:

```
FLMCMD DBUTIL,PRODA,,*,*,,,,,*,,,IN,TEST,PRODDEF,PRODA
              ,,NO,NO,,MSGDD,REPTDD,LISTDD
              ,@@FLMMBR @@FLMGRP @@FLMTYP
```

Based on the generated member list, stored in the data set associated with LISTDD, the process results in the following actions for the different packaging types:

SMPE packaging

1. The library extraction step:
   - All the SMP/E system modification control statements stored in types MCSxxx as listed in Table 5 on page 79 are joined in one data set. This then contains the SMP/E system modification statements for the whole product.
   - The pre-distributed build-outputs stored in SCLM types (see column LIBRARY) across different hierarchy levels in SCLM are copied to distribution libraries (see column DISTLIB) outside of SCLM control. These libraries must have the name of the *actual distribution library* names and must satisfy the required data set specifications required by SMPE, such as logical record length, record format. The names of the *actual distribution libraries* and

the data set specifications are taken from the table of the *package description file* (see Figure 18 on page 80).

- We are also developing a process to generate the JCLIN (LINKJCL to link the whole product on the customer system) from the LEC architecture definition member structure and other information available during development builds. At the moment the JCLIN is manually edited.

2. The formatting step to create the tape(s):

   To produce the SMP/E tapes we use the SUBUILD Package.

   Note: SUBUILD is an IBM internal tools package and part of the CLEAR installation, which can be used without CLEAR. REL File Tape Generator and STACKER are parts of this package.

   The input to this step is the output of the first build step shown in Figure 19 on page 84:

   - control file, containing the SMP/E system modification control statements
   - actual distribution libraries
   - data set containing the related COPYRIGHT statements
   - JCLIN: LINKJCL to link the whole product on the customer system

   The process now results in the following actions:

   a. The REL File Tape Generator creates unloaded partitioned data sets in RELFILE format and associates the generated SMP/E system modification control statements to the corresponding RELFILEs.

   b. The STACKER creates an MVS SMP/E installable product tape.

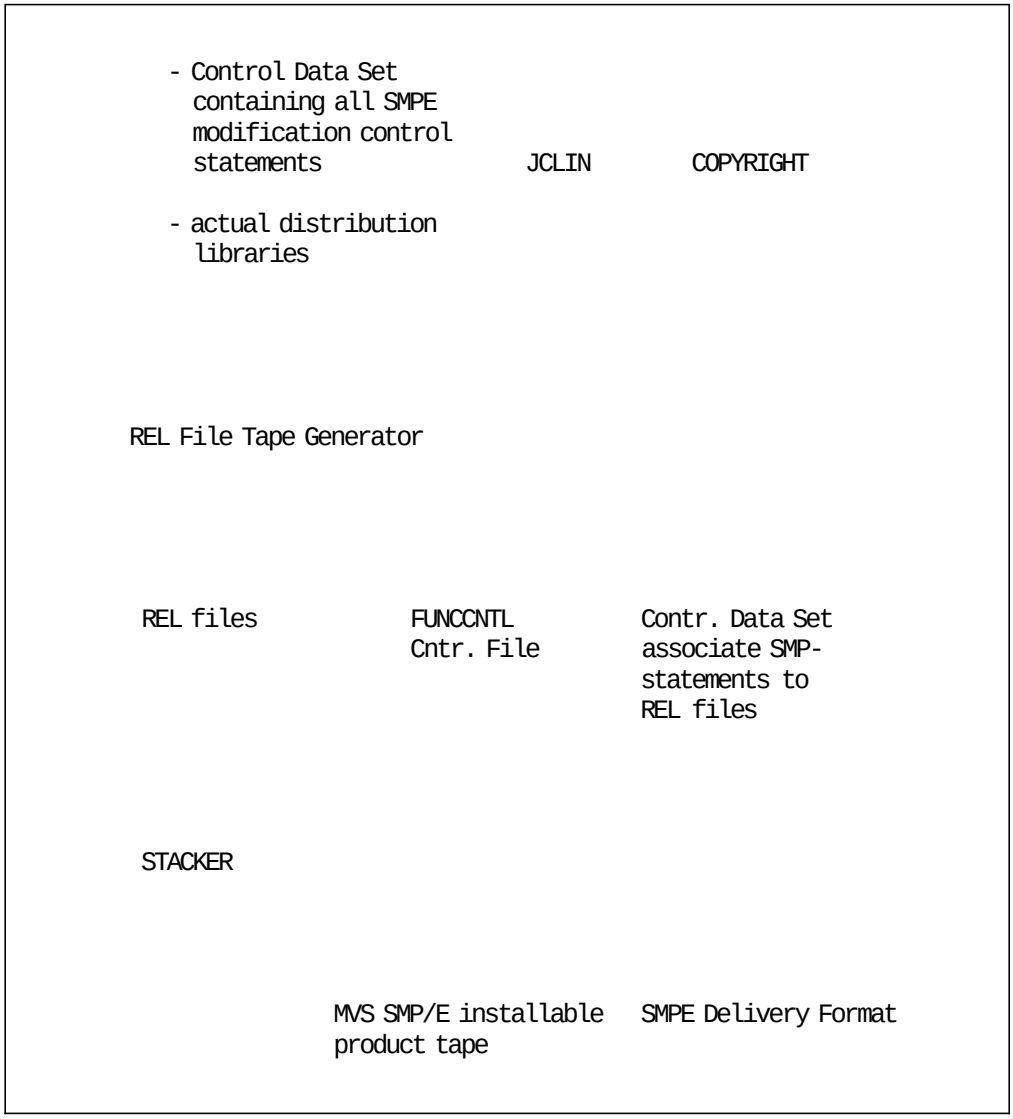


Figure 19. Step 2a - SMPE Packaging

VPL packaging

The extraction step already includes the VPL packaging formatting to create the VPL specific format for each listing. There are no further actions necessary.

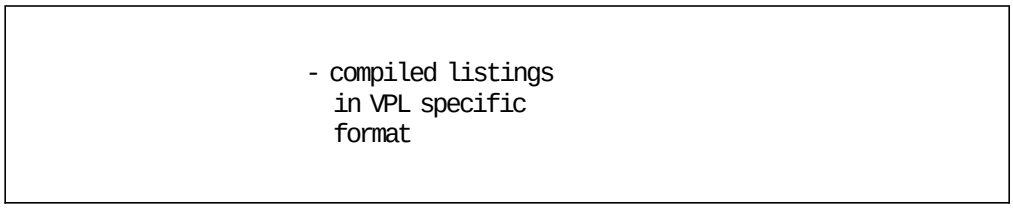


Figure 20. Step 2a - VPL Packaging

Service packaging

The extraction step already contains the following functions:

- A METADATA file is generated, containing all information about parts which are transferred to the service library.

- Source code stored in the SCLM library is copied to transfer data sets outside of SCLM. There is no additional packaging action necessary.

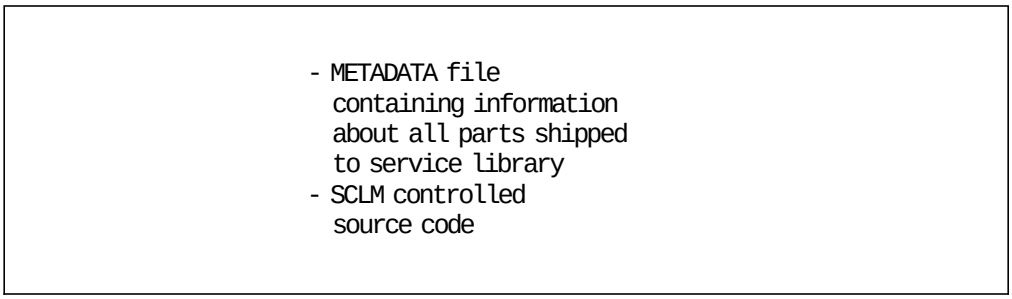


Figure 21. Step 2a - Service Packaging

8.2.1.4 Step 2b - Package Delivery

Package delivery is a separate process step coming after any packaging step. It does not depend on the packaging process but only on the output of such defined package formats.

Therefore for more information refer to 8.3.1, “Package Delivery” on page 88.

8.2.2 Product Packaging

Product packaging for MVS products is not described in this chapter in detail. The reason is that it follows the same packaging principles described in 8.2, “Packaging Processes” on page 76. However to get a better understanding of how it works we provide an example in this chapter which is also used in the general description. All other packaging processes are only a subset with possible different formatting functions needed.

```
    PRODA PRODDEF
INCL MODA     LINKDEF
INCL MODB     LINKDEF
INCL MACA     MACDEF
INCL PANELA   PANLDEF
```

```
   MODA LINKDEF
INCL MODA1 COMPDEF
INCL MODA2 COMPDEF
INCL MODA3 COMPDEF
LOAD MODA  LINKLIB
COPY LEC#1 PARMDEF
```

```
      MODA3 COMPDEF
   MODA2 COMPDEF
 MODA1 COMPDEF
SINC MODA1  MOD
LIST MODA1  LISTING
OUT0 MODA1  CXXXMOD
OUT7 MODA1  MCSMOD
COPY $MOD   SMPEDEF
COPY MOD#1  PARMDEF
```

```
   MODB LINKDEF
INCL MODB1 COMPDEF
INCL MODB2 COMPDEF
INCL MODB3 COMPDEF
LOAD MODB  LINKLIB
COPY LEC#2 PARMDEF
```

```
      MODB3 COMPDEF
   MODB2 COMPDEF
 MODB1 COMPDEF
SINC MODB1  MOD
LIST MODB1  LISTING
OUT0 MODB1  CXXXMOD
OUT7 MODB1  MCSMOD
COPY $MOD   SMPEDEF
COPY MOD#5  PARMDEF
```

```
 MACA MACDEF
SINC MACA   MAC
OUT1 MACA   SXXXMAC
OUT8 MACA   MCSMAC
COPY $MAC   SMPEDEF
COPY MAC#9  PARMDEF
```

```
 PANELA PANLDEF
SINC PANELA DTL
OUT2 PANELA SXXXPAN
OUT9 PANELA MCSPAN
COPY $PAN   SMPEDEF
COPY PAN#3  PARMDEF
```

```
 PRODA PKGDEF
SINC PRODA  PKGDATA
...
```

Figure 22. Sample SCLM Architecture Definition Hierarchy

Description of types used in Figure 22 on page 86

Editable types used in the project are:

MOD Module sources
MAC Macro sources
DTL DTL Panels
PKGDATA Package description file (see Figure 18 on page 80)
MACDEF CC architecture definition members to process macros
PANLDEF CC architecture definition members to process panels
COMPDEF CC architecture definition members to process modules
PKGDEF CC architecture definition members to start the packaging process
LINKDEF LEC architecture definition members to link modules
PRODDEF HL architecture definition members for product and logical component groups
SMPEDEF architecture definition members, containing parameterized models of SMP/E system modification control statements. Examples are listed in Table 6.

Table 6. Models for SMP/E System Modification Control Statements. @@FLMOUx is the SCLM variable containing the member name of the OUTx- keyword. If the type of OUTx key is the same as the distribution type DISTLIB it could be coded as DISTLIB(@@FLMDOx). The models are stored in this example in type SMPEDEF.

Member name	MCS-model
$MOD	PARM7 ++MOD (@@FLMOU0) DISTLIB(CXXXMOD)
$MAC	PARM8 ++MAC (@@FLMOU1) DISTLIB(SXXXMAC) SYSLIB(MACLIB)
$PAN	PARM9 ++PNLENU (@@FLMOU2) DISTLIB(SXXXPAN) SYSLIB(SXXXPENU)
...	...

PARMDEF common process options used in LEC and CC architecture definition members.

Output types used in the project are:

LISTING compile listings
CXXXMOD pre-linked module used for distribution
SXXXMAC macro used for distribution
SXXXPAN translated DTL panel used for distribution
MCSMOD contains SMP/E system modification statements for a module used for distribution
MCSMAC contains SMP/E system modification statements for a macro used for distribution
MCSPAN contains SMP/E system modification statements for a panel used for distribution
LINKLIB contains executable modules

8.2.3 Product Transfer to Service Library

At the same time as the product is transferred to the customer, it must be transferred to our service library.

To do so, the following has to be transferred:

- all the source parts from SCLM

- all product and process parameters (product FMID, release name, part names in association with used compilers, compile options, link options and so on). Those parameters are all defined in the SCLM library and have to be generated from the SCLM library into a METADATA file. The METADATA format is defined by the service library and is used to load the product and process information into its database.

8.3 Delivery Processes

8.3.1 Package Delivery

The package delivery process only relies on the provided formats to be delivered and does not care how such a package is created. It covers the mechanism to automate and control the distribution of electronical parts. It normally does not address the physical distribution where electronical parts are stored on physical devices such as tapes or diskettes and are then transported physically.

The delivery step is a manual step today using several actions. However it would be possible to put this under SCLM control if we decide to do so. Package delivery therefore today is not under SCLM control.

If the deliveries have to be sent to several places and several times, then the time would come to put it under SCLM control to document and automate the deliveries.

SMPE Delivery Either the tape is delivered physically or we use a utility to automatically unload the tape and transfer it electronically to a target system.

VPL Delivery VPL listings are transmitted to IBM Software Manufacturing.

Service Delivery The service delivery results in the following actions:

1. Transmit the project source code to the service library
2. Transmit the METADATA file to the service library and fill the service library database.

8.3.2 SHIPIT - Externalize Data during Promote Time

In our development process, we want to deliver SCLM project definitions, developed code, such as project load modules REXX execs and documentation, to the run-time environment on other local libraries or libraries on other systems. Therefore we start some specific actions during the promote stage. To be as flexible as possible, we have developed a tool named SHIPIT, which is called as a promote translator. With the help of this tool, we can start different actions during the promote driven by a provided mapping member called *shiplist*.

The invocation of the SHIPIT tool is:

```
SHIPIT project , from_group , to_group , type , member => /
```

The operands to call the exec SHIPIT are as follows:

project SCLM project name. Typically variable @@FLMPRJ.

from_group destination group when promote exit will be active. Typically variable @@FLMGRP.

to_group target group when promote exit will be active. Typically variable @@FLMTOG.
type data set type. Typically variable @@FLMTYP.
member member name. Typically variable @@FLMMBR.

The SHIPIT tool refers to a *shiplist*, which is scanned through for one or more matching entries of the member to be promoted. The *shiplist* information has to be stored in the data set member *<project>.PROJDEFS.SHIPLIST(<project>)*.

The syntax of one line of the *shiplist* member is:

```
* text
project . from_group -> to_group . type ( member ) => action
                                           *
@INCLUDE( include_member )
```

action:

```
local_copy
@XMIT
@SEND
@FTP
@TSO
@EXEC
```

There could be as many of such lines in one member.

The operands for the *shiplist* are as follows:

project project name
from_group destination group when promote exit will be active
to_group target group when promote exit will be active
type data set type
member member name
***** if '*' specified all members are processed
text any text to be considered a comment
include_member information contained in a member be inserted here

The following actions are supported:

local_copy copy parts to external libraries on the same system (see 8.3.2.1, "Default Action - Local Copy to Data Sets")
@XMIT distribute parts to other systems (see 8.3.2.2, "Distribute Data Sets to Other Systems" on page 90)
@SEND sequential transmits used for the sending of parts from MVS to VM.
@FTP copy parts to an external library on another system (see 8.3.2.2, "Distribute Data Sets to Other Systems" on page 90)
@TSO issue a TSO command (see 8.3.2.3, "Generic Actions during Promote" on page 91)
@EXEC issue a REXX exec (see 8.3.2.3, "Generic Actions during Promote" on page 91)

8.3.2.1 Default Action - Local Copy to Data Sets

local_copy:

```
target_library
```

target_library target library where the data will be externalized. It can be just a data set or a data set member.

One action is a local copy to external data sets. In the following example, we copy the member 'SCLMSITE' of type 'REXX' during the promote from the promote group TEST to the target group PROD and to the external data set 'PC.$ASIS$.ALLOC' with the same member name. The corresponding *shiplist* contains the following information:

```
SCLM1.TEST  -> PROD.REXX(SCLMSITE) => PC.$ASIS$.ALLOC
```

8.3.2.2 Distribute Data Sets to Other Systems

When we have to distribute parts to other systems, our 'SHIPIT' process provides the following possibilities.

Action @XMIT

@XMIT:

```
@XMIT  target_system  .  target_userid
```

target_system
Name of the target system to where the member will be sent.

target_userid
Name of the target user ID to where the member will be sent.

The following example transmits the part SCLMSITE during the promote from group TEST to group PROD and then to the user ID ABC on system XYZ using the TSO transmit command. The *shiplist* contains the following information:

```
SCLM1.TEST   -> PROD.REXX(SCLMSITE) => @XMIT XYZ.ABC
```

Action @SEND

@SEND:

```
@SEND  target_system  .  target_userid
```

target_system
Name of the target system to where the member will be sent.

target_userid
Name of the target user ID to where the member will be sent.

The following example transmits the part SCLMSITE during the promote from group TEST to group PROD and then as a sequential data set to the user ID ABC on the VM system VMXYZ using the TSO transmit command. The *shiplist* contains the following information:

```
SCLM1.TEST   -> PROD.REXX(SCLMSITE) => @SEND VMXYZ.ABC
```

Action @FTP

@FTP:

```
@FTP  \\  target_system  \  target_library
```

target_library
target library where the data will be externalized. It can be just a data set or a data set member.

target_system
target system to where member will be copied/transmitted.

The example copies member SCLMSITE to the target library PC.$ASIS$.ALLOC on system XYZ with the member name SCLMAAA using the TCP/IP command FTP. The IP-address of the target system, login user ID and password are stored in a separate file.

```
SCLM1.TEST->PROD.REXX(SCLMSITE)=>FTP XYZ¢PC.$ASIS$.PROD(SCLMAAA)¢
```

8.3.2.3 Generic Actions during Promote

With our SHIPIT tool, we actually can invoke any desired actions using the following commands. The following variables can be used @@FLMPRJ, @@FLMGRP, @@FLMTOG, @@FLMTYP and @@FLMMBR.

Action @TSO

@TSO:

```
@TSO  tso_command
```

tso_command
tso command which will be issued when promote exit is active.

The following example simulates an external promote. It does a copy of the member AAA to the 'PC.$ASIS$.PROD' and issues a TSO command to delete member AAA from 'PC.$ASIS$.TEST' during the promote from group TEST to group PROD.

```
SCLM1.TEST  -> PROD.REXX(SCLMSITE) =>             ¢PC.$ASIS$.PROD(AAA)¢
SCLM1.TEST  -> PROD.REXX(SCLMSITE) => @TSO DELETE ¢PC.$ASIS$.TEST(AAA)¢
```

Action @EXEC

@EXEC:

```
@EXEC  exec_command
```

exec_command
REXX exec which will be issued when promote exit is active.

The following entries in the *shiplist* starts the REXX Exec TESTEXEC with option ABC for all members of type ASM promoted from TEST to PROD.

```
SCLM1.TEST->PROD.ASM(*)=>@EXEC ¢PC.SCLMSITE.REXX(TESTEXEC)¢ ¢@@FLMMBR (ABC¢
SCLM1.TEST->PROD.ASM(*)=>@EXEC TESTEXEC  @@FLMMBR (ABC
```

8.3.3 CLEAR as Shadow Library, SCLM as Master

During our migration from the CLEAR library to the SCLM library, we had to use the packaging features such as SMPE Tape generation, and VPL, as they exist in CLEAR.

To use these functions with almost no extra effort we had to keep CLEAR as a shadow library for this project, until our SCLM environment also offered the packaging functions. We implemented a PROMOTE exit, which started a REXX exec (as is described in 8.3.2.3, "Generic Actions during Promote") and automatically synchronized the CLEAR library with the SCLM library. The process guaranteed that the view from group FINAL up (see Figure 23 on page 92) exactly represents the CLEAR contents.

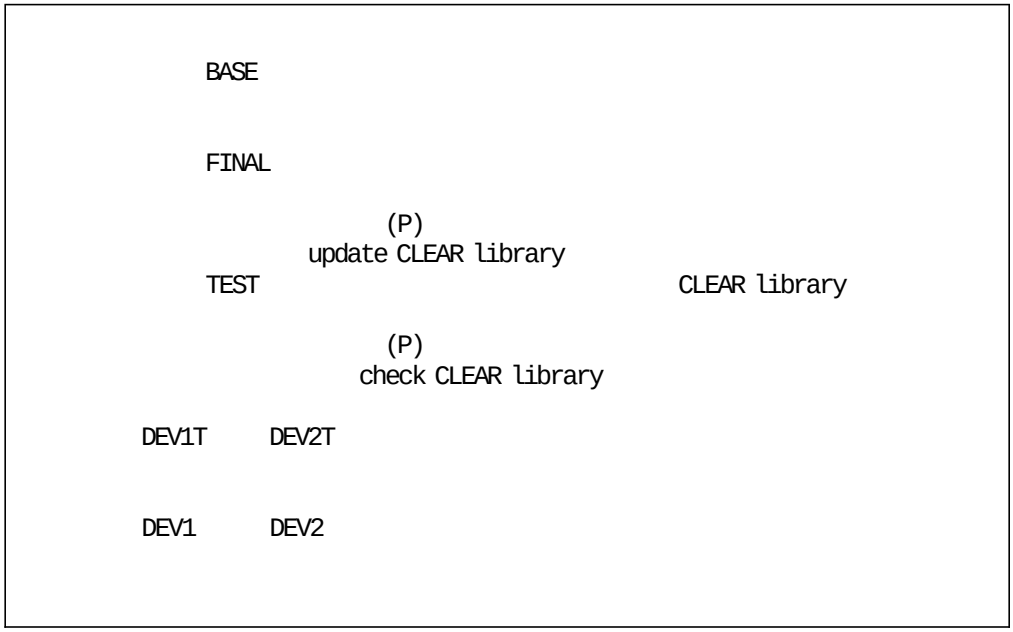


Figure 23. Process to Synchronize CLEAR Library with SCLM Library

Description of the processing to synchronize the two libraries (see Figure 23)

1. At the beginning of development the BASE group in SCLM and the CLEAR database have the same contents and all processes needed to process parts are implemented in SCLM. Development build processes such as compiles in CLEAR, are implemented in SCLM so that they are synchronized.
2. The developers developed their code in groups DEV1 and DEV2.
3. If their code was ready for the next driver, they promoted their code to DEV1T and DEV2T.
4. The integrator promoted the code to the common group TEST. During this step, the CLEAR database was checked if the part exists in the CLEAR database. If some inconsistencies existed (for example, the part was not defined in CLEAR) a corresponding message was written to a LOG file and the promote ended successfully with return code 4. The integrator had to solve the inconsistencies.
5. After testing the driver on level TEST, the integrator promoted the code to level FINAL. If some inconsistencies still existed in the CLEAR database, the promote ends unsuccessfully. In this case, the integrator had to solve the inconsistencies and issued the promote once more. If no inconsistency exists during this step, the CLEAR database is updated.

   After a successful promote, the group hierarchy FINAL-BASE exactly represents the contents of the CLEAR library.

   Because the process used in SCLM for compiles is the same as that used in CLEAR, the product tape generation was a clean process of doing recompiles. Problems were likely only for CLEAR specific packaging steps.

In effect this approach can be seen as reducing the number of CLEAR users from n developers to just one virtual developer.

8.4 Service Processes

In addition to the CLEAR library on MVS to do program development, we have a service library on VM to maintain the products in service with an interface to RETAIN and to produce the APAR and PTF formats for all the /390 operating systems VSE, MVS and VM.

Due to the fact that RETAIN interface and the creation of the APAR and PTF formats are anchored in the service library system, we still transfer our products at delivery to the service library to develop fixes. Because we used the service library already in parallel to our CLEAR library in the past we have some interfaces with the development library. This means that we have migrated the following interface to work also with SCLM:

- a tool to transfer the source code and process parameters at the end of the development cycle to the service library. For more details see 8.2.3, “Product Transfer to Service Library” on page 87.
- an interface to store parts in the active development library or libraries. The parts are modified parts in the service library and delivered as PTFs. They are then there available to integrate (rebase) the fixes in the new products. This interface is described in more detail in 8.4.1, “Interface to Service Library” and the process of rebase is described in more detail in 8.1.1, “Rebase of Service to Current Development” on page 73.

8.4.1 Interface to Service Library

Because we have a separate service library on VM we have to establish an interface to SCLM. Parts, that are modified in the Service Library in the sense that a field problem is corrected and delivered to the customer, must be integrated in the sources of the ongoing development done in SCLM. Therefore a mechanism must be in place to track those modified parts for integration of the applied fixes in the parallel ongoing release developments.

The interface works basically in the way that we use an exit of the service library which will, at the time the fix is done, transfer the parts to MVS via a batch job and store them, driven by a provided table, in the desired SCLM groups and types with names such as PTFNEW as shown in Figure 1 on page 15. From there they are handled using the process described in 8.1.1, “Rebase of Service to Current Development” on page 73.

Appendix A. SCLM Definitions and Functions for Supported Languages

This chapter lists languages we support in our environment which we think may also be interesting to the reader. Some are listed in detail and some just in the format of a user's guide style, because to list all would be beyond the scope of the document. If you need more information please contact the authors.

A.1 Process C++ Module Source

```
************************************************************************
*****                Language definition for C / C++                *****
************************************************************************
&CPPVER  SETC  ¢3.1.0¢
*
*        Define macros outside of hierarchy
*
CPP      FLMSYSLB &PREFCPP..SCLB3H.H
         FLMSYSLB &PREFCPP..SCLB3H.HPP
         FLMSYSLB &PREFCPP..SCLB3H.INL
         FLMSYSLB &PREFLE..SCEEH.H
         FLMSYSLB &PREFLE..SCEEH.SYS.H
*
*        Define language
*
         FLMLANGL LANG=CPP,VERSION=&CPPVER
*
*        Define include sets
*
         FLMINCLS TYPES=(HPP)
*
*        Language Parser (In fact, it¢s a dummy (TEXT) Parser)
*
          FLMTRNSL   CALLNAM=¢C++-Parser   ¢,                          *
               FUNCTN=PARSE,                                           *
               COMPILE=FLMLPGEN,                                       *
               PORDER=1,                                               *
               OPTIONS=(SOURCEDD=SOURCE,                               *
               STATINFO=@@FLMSTP,                                      *
               LISTINFO=@@FLMLIS,                                      *
               LISTSIZE=@@FLMSIZ,                                      *
               LANG=T),                                                *
               PARMKWD=PARM0
*              --  SOURCE  --
               FLMALLOC  IOTYPE=A,DDNAME=SOURCE
               FLMCPYLB @@FLMDSN(@@FLMMBR)
```

```
*
*         Language Build processor (TRANSLATOR) Step 1
*
          FLMTRNSL   CALLNAM=¢C++-Compile ¢,                          *
               FUNCTN=BUILD,                                          *
               COMPILE=SCLMISP,                                       *
               CALLMETH=LINK,                                         *
               VERSION=&CPPVER,                                       *
               PORDER=1,                                              *
               GOODRC=4,                                              *
               OPTIONS=¢SCLMCPP @@FLMMBR¢,                            *
               PARMKWD=PARM1
*  1        -- SYSIN  -- (Input)
            FLMALLOC  IOTYPE=S,KEYREF=SINC,DDNAME=CPPIN,               *
               RECFM=VB,LRECL=255,RECNUM=99000,CATLG=Y
*  2        -- SYSLIN  -- (Output)
            FLMALLOC  IOTYPE=O,KEYREF=OBJ,DDNAME=CPPLIN,LANG=CPPOBJ,   *
               RECFM=FB,LRECL=80,RECNUM=99000,BLKSIZE=3200,CATLG=Y
*  3        -- USERLIB  -- (User H / HPP Files)
            FLMALLOC  IOTYPE=I,KEYREF=SINC,DDNAME=CPPULIB
*  4        -- SYSCPRT  -- (LISTING)
            FLMALLOC  IOTYPE=W,PRINT=Y,DDNAME=CPPCPRT,                 *
               RECFM=VBA,LRECL=137,RECNUM=20000,CATLG=Y
*
*         C++-TRANSLATOR Step 2: Store and Pack listing
*
          FLMTRNSL   CALLNAM=¢C++-Store/Pack¢,                        *
               FUNCTN=BUILD,                                          *
               COMPILE=SCLMISP,                                       *
               CALLMETH=LINK,                                         *
               VERSION=&CPPVER,                                       *
               PORDER=1,                                              *
               GOODRC=4,                                              *
               OPTIONS=¢SCLMPOST SYSPRPAC/COPY IN(CPPCPRT) OUT(LISTCPP)*
               /¢,                                                    *
               PARMKWD=PARM2
* 01        -- SYSPRPAC  -- (Messages from EXEC)
            FLMALLOC  IOTYPE=W,DDNAME=SYSPRPAC,PRINT=Y,                *
               RECFM=FBA,LRECL=133,RECNUM=1000
* 02        -- LISTCPP  -- (Output listing file, may now be packed)
            FLMALLOC  IOTYPE=O,KEYREF=LIST,DDNAME=LISTCPP,             *
               RECFM=VBA,LRECL=137,RECNUM=100000
* 03        -- CPPCPRT  -- (Listing file of C++ Compiler)
            FLMALLOC  IOTYPE=U,DDNAME=CPPCPRT
```

```
*
*         C++-TRANSLATOR Step 3: Track dynamic includes
*
          FLMTRNSL   CALLNAM=¢C++-Includes¢,                          *
               FUNCTN=BUILD,                                          *
               COMPILE=SCLMISP,                                       *
               CALLMETH=LINK,                                         *
               VERSION=&CPPVER,                                       *
               PORDER=1,                                              *
               GOODRC=4,                                              *
               OPTIONS=¢SCLMPMON   SYSPRMON/@@FLMPRJ,@@FLMALT,@@FLMGRP,*
               @@FLMMBR,@@FLMINC/LANGCPP(CPPCPRT,HPP)¢,               *
               PARMKWD=PARM4
* 01       -- SYSPRMON --
           FLMALLOC  IOTYPE=W,RECFM=FBA,LRECL=133,BLKSIZE=6118,       *
               RECNUM=500,DDNAME=SYSPRMON,PRINT=Y
* 02       -- CPPCPRT --
           FLMALLOC  IOTYPE=U,DDNAME=CPPCPRT
```

A.2 Prelink C++ Objects

```
*************************************************************************
*****            C++ Prelinker & DFSMS Program Binder             *****
*************************************************************************
&CPLVER  SETC  ¢3.1.0¢
*
*
*
         FLMLANGL    LANG=CPPOBJ,CANEDIT=N,VERSION=&CPLVER
*
*        No language parser !
*
*
*        Language Build Processor Step 1 (Invoke C++-Prelinker)
*
          FLMTRNSL   CALLNAM=¢C++-Prelinker¢,                        *
               FUNCTN=BUILD,                                         *
               COMPILE=EDCPRLK,                                      *
               VERSION=&CPLVER,                                      *
               PORDER=1,                                             *
               GOODRC=8,                                             *
               OPTIONS=(MAP,ER,NONCAL),                              *
               PARMKWD=PARM1
*  1       -- SYSIN --
           FLMALLOC  IOTYPE=S,KEYREF=INCL,RECFM=FB,LRECL=80,          *
               RECNUM=20000,DDNAME=SYSIN
*           FLMCPYLB &PREFCPP..SCLB3SID(IOSTREAM)
*           FLMCPYLB &PREFCPP..SCLB3SID(COMPLEX)
*           FLMCPYLB &PREFCPP..SCLB3SID(APPSUPP)
*           FLMCPYLB &PREFCPP..SCLB3SID(COLLECT)
*  2       -- SYSMSGS --
           FLMALLOC  IOTYPE=A,DDNAME=SYSMSGS
            FLMCPYLB &PREFLE..SCEEMSGP(EDCPMSGE)
*  3       -- SYSLIB --
           FLMALLOC  IOTYPE=A,DDNAME=SYSLIB
            FLMCPYLB &PREFCPP..SCLB3CPP
            FLMCPYLB &PREFLE..SCEECPP
```

```
*  4       --  SYSDEFSD  --
           FLMALLOC  IOTYPE=A,DDNAME=SYSDEFSD
            FLMCPYLB NULLFILE
*  5       --  SYSMOD  --
           FLMALLOC  IOTYPE=O,DDNAME=SYSMOD,RECFM=FB,LRECL=80,          *
               BLKSIZE=3120,RECNUM=200000,KEYREF=OUT7
*  6       --  SYSPRINT  --
           FLMALLOC  IOTYPE=W,DDNAME=SYSPRINT,RECFM=FBA,LRECL=133,      *
               BLKSIZE=1330,RECNUM=1000
*  7       --  SYSOUT  --
           FLMALLOC  IOTYPE=W,DDNAME=SYSOUT,PRINT=Y,RECFM=FB,LRECL=80, *
               RECNUM=200000,BLKSIZE=23200
*  4       --  SCLB3SID  --
           FLMALLOC  IOTYPE=A,DDNAME=SCLB3SID
            FLMCPYLB &PREFCPP..SCLB3SID
*
*        Language Build Processor step 2 (Invoke Program Binder)
*
            FLMTRNSL   CALLNAM=¢C++-Prebind¢,                           *
                FUNCTN=BUILD,                                           *
                COMPILE=IEWL,                                           *
                VERSION=&CPLVER,                                        *
                PORDER=3,                                               *
                GOODRC=8,                                               *
                OPTIONS=(LIST,MAP,NOXREF,REUS=REFR,SIZE=1024K,LET),     *
                PARMKWD=PARM2
*  1       --  SYSLIN  --
           FLMALLOC  IOTYPE=U,DDNAME=SYSMOD
*  2       --  LOAD MODULE NAME  --
           FLMALLOC  IOTYPE=L,KEYREF=LOAD
*  3       --  SYSLMOD  --
           FLMALLOC  IOTYPE=P,KEYREF=LOAD,RECFM=U,LRECL=6144,           *
               RECNUM=30000,DIRBLKS=20
*  4       --  SYSLIB  --
           FLMALLOC  IOTYPE=A
            FLMCPYLB &PREFLE..SCEELKED
*  5       --  N/A  --
           FLMALLOC  IOTYPE=N
*  6       --  SYSPRINT  --
           FLMALLOC  IOTYPE=W,RECFM=FBA,LRECL=133,DDNAME=LKEDTEMP,      *
               RECNUM=100000,PRINT=Y
*  7       --  N/A  --
           FLMALLOC  IOTYPE=N
*  8       --  SYSUT1  --
           FLMALLOC  IOTYPE=W,RECFM=U,LRECL=6144,RECNUM=2000
*  9       --  N/A  --
           FLMALLOC  IOTYPE=N
* 10       --  N/A  --
           FLMALLOC  IOTYPE=N
* 11       --  N/A  --
           FLMALLOC  IOTYPE=N
* 12       --  SYSTERM  --
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
```

```
*
*          C++ Prelink-Translator Step 3: Store and Pack data sets
*
          FLMTRNSL   CALLNAM=¢C++-Store Maps¢,                          *
               FUNCTN=BUILD,                                            *
               COMPILE=SCLMISP,                                         *
               CALLMETH=LINK,                                           *
               VERSION=&CPLVER,                                         *
               PORDER=1,                                                *
               GOODRC=4,                                                *
               OPTIONS=¢SCLMPOST SYSPRPAC/COPYMULT IN(*SYSOUT,LKEDTEMP)*
                OUT(LISTCPL) WORK(LISTTEMP)/¢,                          *
               PARMKWD=PARM3
* 01        --  SYSPRPAC  -- (Output file for messages of EXEC)
           FLMALLOC  IOTYPE=W,DDNAME=SYSPRPAC,PRINT=Y,                  *
               RECFM=FBA,LRECL=133,RECNUM=1000
* 02        --  LISTCPL  -- (Combined Prelink/IEWL listing)
           FLMALLOC  IOTYPE=O,KEYREF=LMAP,DDNAME=LISTCPL,               *
               RECFM=VBA,LRECL=137,RECNUM=300000
* 03        --  LISTTEMP  -- (Temporary for sum of PLXPRTMP+LISTING)
           FLMALLOC  IOTYPE=W,DDNAME=LISTTEMP,                          *
               RECFM=FBA,LRECL=133,RECNUM=300000
* 04        --  SYSOUT  -- (Prelinker output)
           FLMALLOC  IOTYPE=U,DDNAME=SYSOUT
* 05        --  LKEDTEMP  -- (IEWL output)
           FLMALLOC  IOTYPE=U,DDNAME=LKEDTEMP
```

A.3 Process Architecture Definition

LANG=ARCHDEF_ ; Member=SCLMLANG(ARCHDEF)

Description

This is a Parser Translator only used to gather statistics for the architecture definitions. For details about architecture definition members refer to the SCLM documentation.

PARSER Translator Steps

For parser translator used see B.1.2, “P$ARCHDEF SCLMMACS - Parse for Architecture Definitions Members” on page 107.

A.4 Process Plain Text without Keywords

LANG=TEXT____ ; Member=SCLMLANG(TEXT   )

Description

This is a language with:

- a parser to gather statistics for plain text without any keywords or special sections
- no build steps

PARSER Translator Steps

For parser translator used see B.1.3, “P$TEXT SCLMMACS - Parse for Text Only Members” on page 107.

A.5 Process to Distribute Parts during Promote Out of Project

LANG=SHIPIT__ ; Member=SCLMLANG(SHIPIT)

Description

This is a language with:

- a parser to gather statistics for plain text without any keywords or special sections.
- no build steps
- a copy translator to distribute members during promote according to the SHIPIT execs information provided.

PARSER Translator Steps

For parser used see B.1.3, “P$TEXT SCLMMACS - Parse for Text Only Members” on page 107.

COPY Translator Steps

For copy translator used see B.3.1, “C$SHIPIT SCLMMACS - General Promote Exit” on page 118.

A.6 Process SCLM Project Definitions

LANG=SCLMMOD_ ; Member=SCLMLANG(SCLMMOD)

Description

This language will invoke the process for SCLM project definition parts which are not master files to be compiled to create a load module. The parts are however, used by master files via either a copy or a macro include.

For external libraries used see C.1, “E#SCLM SCLMVIEW - External SCLM Macro Libraries” on page 119.

For internal libraries used see C.2, “I$SCLM SCLMVIEW - Internal SCLM Macro Libraries” on page 120.

PARSER Translator Steps

For parser translator used see B.1.4, “P$ASM SCLMMACS - Parse of Assembler Format” on page 107.

BUILD Translator Steps

For build translator used see B.2.8, “B$SCLMAC SCLMMACS - Process SCLM Project Definition” on page 117

A.7 Process SCLM Project Parts

LANG=SCLMMAC_ ; Member=SCLMLANG(SCLMMAC)

Description

This language will invoke the process for SCLM project definition parts which are not master files to be compiled to create a load module. However, the parts are used by master files via either a copy or a macro include.

For external libraries used see C.1, “E#SCLM SCLMVIEW - External SCLM Macro Libraries” on page 119.

For internal libraries used see C.2, “I$SCLM SCLMVIEW - Internal SCLM Macro Libraries” on page 120.

PARSER Translator Steps

For parser translator used see B.1.4, “P$ASM SCLMMACS - Parse of Assembler Format” on page 107.

Build Translator Steps

For build translator see B.2.8, “B$SCLMAC SCLMMACS - Process SCLM Project Definition” on page 117.

A.8 Link Objects to an Executable Load Module

LANG=LE370___ ; Member=SCLMLANG(LE370)

Description

This language processes objects to create a linked load module.

BUILD Translator Steps

For build translator used see B.2.5, “B$LE370 SCLMMACS - Create a Load Module” on page 113.

A.9 Process for Load Modules

LANG=LOAD____ ; Member=SCLMLANG(LOAD)

Description

This is a language with:

- a parser to gather statistics for plain text without any keywords or special sections.
- a copy translator to distribute members during promote according to the information provided by the SHIPIT EXECs.

PARSER Translator Steps

For parser translator used see B.1.3, “P$TEXT SCLMMACS - Parse for Text Only Members” on page 107.

COPY Translator Steps

For copy translator used see B.3.1, “C$SHIPIT SCLMMACS - General Promote Exit” on page 118.

A.10 Process REXX Modules

LANG=REXXMOD_ ; Member=SCLMLANG(REXXMOD)

Description

This is a language which will parse REXX modules and process with the RXPREP program.

PARSER Translator Steps

For parser translator used see B.1.5, “P$RXPREP SCLMMACS - Parse for REXX to be Preprocessed” on page 107.

BUILD Translator Steps

For build translator used see B.2.6, “B$RXPREP SCLMMACS - Preprocess a REXX Part” on page 114.

A.11 Process REXX Macros

LANG=REXXMAC_ ; Member=SCLMLANG(REXXMAC)

Description

This is a language which will parse REXX macros used by the RXPREP program.

PARSER Translator Steps

For parser translator used see B.1.5, “P$RXPREP SCLMMACS - Parse for REXX to be Preprocessed” on page 107.

A.12 Process REXX Programs

LANG=REXX____ ; Member=SCLMLANG(REXX)

Description

This language will treat executable REXX programs. Currently no BUILD steps are defined. A copy translator is defined to enable the distribution of the REXX programs during promote.

PARSER Translator Steps

For parser translator used see B.1.3, “P$TEXT SCLMMACS - Parse for Text Only Members” on page 107.

COPY Translator Steps

For copy translator used see B.3.1, “C$SHIPIT SCLMMACS - General Promote Exit” on page 118.

A.13 Process DTL Syntax to Produce ISPF Panels, Messages and Tables

LANG=DTL_____ ; Member=SCLMLANG(DTL)

Description

This is the language to create from DTL source all the possible members in one step.

PARSER Translator Steps

A parser translator to get accounting information is described in B.1.1, “P$DTLC SCLMMACS - Parse DTL Source Members” on page 107.

BUILD Translator Steps

Now create from DTL the possible formats as described in B.2.4, “B$DTLC SCLMMACS - Process DTL Source” on page 113.

A.14 Get Dependencies for BookMaster Format without Formatting

LANG=IMBED___ ; Member=SCLMLANG(IMBED)

Description

This language allows to parse BookMaster sources to get the dependencies, but no formatting will be done. This is necessary for imbed files and other BookMaster dependent parts.

PARSER Translator Steps

For parser translator used see B.1.6, “P$BMASTR SCLMMACS - Parse of BookMaster Format” on page 107.

Code Fragment 1 (Member=IMBED SCLMLANG)

```
************************************************************************
* Change Activities:
* 950817 JP Created
************************************************************************
         FLMLANGL    LANG=IMBED,VERSION=D950817
*
         COPY P$BMASTR
```

End of Code Fragment 1 (Member=IMBED SCLMLANG)

A.15 Process BookMaster Format to Create Printable or Online Books

LANG=BOOKIE__ ; Member=SCLMLANG(BOOKIE)

Description

This language allows to process BookMaster sources to create LIST3820 and/or BOOK format outputs to be either printed or read online by BookManager.

Outputs may be subject to distribution by the shipit process.

For internal include libraries see C.4, "I$SCRIPT SCLMVIEW - Internal Documentation Libraries" on page 121.

Code Fragment 1 (Member=BOOKIE SCLMLANG)

```
************************************************************************
* Change Activities:
* 950619 JP created
************************************************************************
*
         FLMLANGL    LANG=BOOKIE,VERSION=D950619
*
         COPY I$SCRIPT
```

End of Code Fragment 1 (Member=BOOKIE SCLMLANG)

PARSER Translator Steps

For parser translator used see B.1.6, "P$BMASTR SCLMMACS - Parse of BookMaster Format" on page 107.

Code Fragment 2 (Member=BOOKIE SCLMLANG)

```
*
         COPY P$BMASTR
```

End of Code Fragment 2 (Member=BOOKIE SCLMLANG)

BUILD Translator Steps

1. For build translator used see B.2.2, "B$BOOKIE SCLMMACS - Create Printable Documentation" on page 111.

2. For build translator used see B.2.1, “B$BMANGR SCLMMACS - Create On-line Documentation” on page 108.

Code Fragment 3 (Member=BOOKIE SCLMLANG)

```
*
         COPY B$BOOKIE
*
         COPY B$BMANGR
```

End of Code Fragment 3 (Member=BOOKIE SCLMLANG)

COPY Translator Steps

For build translator used see B.3.1, “C$SHIPIT SCLMMACS - General Promote Exit” on page 118.

Code Fragment 4 (Member=BOOKIE SCLMLANG)

```
*
         COPY C$SHIPIT
```

End of Code Fragment 4 (Member=BOOKIE SCLMLANG)

A.16 Process BookMaster Format to Create Printable Books

LANG=SCRIPT__ ; Member=SCLMLANG(SCRIPT)

Description

This language allows to process BookMaster sources to create LIST3820 format outputs to be printed.

For internal include libraries see C.4, “I$SCRIPT SCLMVIEW - Internal Documentation Libraries” on page 121.

Code Fragment 1 (Member=SCRIPT SCLMLANG)

```
************************************************************************
* Change Activities:
* 950619 JP created
************************************************************************
         FLMLANGL    LANG=SCRIPT,VERSION=D950619
*
         COPY I$SCRIPT
```

End of Code Fragment 1 (Member=SCRIPT SCLMLANG)

PARSER Translator Steps

For parser translator used see B.1.6, “P$BMASTR SCLMMACS - Parse of BookMaster Format” on page 107.

Code Fragment 2 (Member=SCRIPT SCLMLANG)

```
*
        COPY P$BMASTR
```

End of Code Fragment 2 (Member=SCRIPT SCLMLANG)

BUILD Translator Steps

For build translator used see B.2.2, “B$BOOKIE SCLMMACS - Create Printable Documentation” on page 111.

Code Fragment 3 (Member=SCRIPT SCLMLANG)

```
*
        COPY B$BOOKIE
```

End of Code Fragment 3 (Member=SCRIPT SCLMLANG)

COPY Translator Steps

For build translator used see B.3.1, “C$SHIPIT SCLMMACS - General Promote Exit” on page 118.

Code Fragment 4 (Member=SCRIPT SCLMLANG)

```
      *
              COPY C$SHIPIT
```

End of Code Fragment 4 (Member=SCRIPT SCLMLANG)

A.17 Create BookManager Format from BookMaster Format

LANG=BOOKMGR_ ; Member=SCLMLANG(BOOKMGR)

Description

This language allows to create BookManager BOOK format from BookMaster source.

For internal include libraries see C.4, “I$SCRIPT SCLMVIEW - Internal Documentation Libraries” on page 121.

Code Fragment 1 (Member=BOOKMGR SCLMLANG)

```
*************************************************************************
* Change Activities:
* 950619 JP created
*************************************************************************
        FLMLANGL    LANG=BOOKMGR,VERSION=D950619
*
        COPY I$SCRIPT
```

End of Code Fragment 1 (Member=BOOKMGR SCLMLANG)

PARSER Translator Steps

For parser translator used see B.1.6, “P$BMASTR SCLMMACS - Parse of BookMaster Format” on page 107.

Code Fragment 2 (Member=BOOKMGR SCLMLANG)

```
*
        COPY P$BMASTR
```

End of Code Fragment 2 (Member=BOOKMGR SCLMLANG)

BUILD Translator Steps

For build translator used see B.2.1, “B$BMANGR SCLMMACS - Create On-line Documentation” on page 108.

Code Fragment 3 (Member=BOOKMGR SCLMLANG)

```
*
        COPY B$BMANGR
```

End of Code Fragment 3 (Member=BOOKMGR SCLMLANG)

COPY Translator Steps

For build translator used see B.3.1, “C$SHIPIT SCLMMACS - General Promote Exit” on page 118.

Code Fragment 4 (Member=BOOKMGR SCLMLANG)

```
*
        COPY C$SHIPIT
```

End of Code Fragment 4 (Member=BOOKMGR SCLMLANG)

Appendix B. Translators

This appendix lists all the translators referred to in Appendix A, “SCLM Definitions and Functions for Supported Languages” on page 95 and some additional ones which might be of interest to the reader. Some are listed in detail and some just in the format of a user’s guide, because to list all would be beyond the scope of the document. If you need more information please contact the authors.

B.1 Parser Translators Reusable Parts

B.1.1 P$DTLC SCLMMACS - Parse DTL Source Members

PARSER Translator Steps

A Parser Translator to get accounting information about a DTL member.

B.1.2 P$ARCHDEF SCLMMACS - Parse for Architecture Definitions Members

PARSER Translator Steps

A Parser Translator to get accounting information about an ARCHDEF member. This is the ISPF provided parser FLMLSS. For details see the SCLM documentation.

B.1.3 P$TEXT SCLMMACS - Parse for Text Only Members

PARSER Translator Steps

A Parser Translator to get accounting information about a plain text member. It is the ISPF provided parser FLMLPGEN. For details see the SCLM documentation.

B.1.4 P$ASM SCLMMACS - Parse of Assembler Format

PARSER Translator Steps

Parse the source for include and macro dependencies.

B.1.5 P$RXPREP SCLMMACS - Parse for REXX to be Preprocessed

PARSER Translator Steps

A Parser Translator to get dependencies of REXX macros processed with the RXPREP program.

B.1.6 P$BMASTR SCLMMACS - Parse of BookMaster Format

PARSER Translator Steps

A Parser Translator to get dependencies of BookMaster and script source. This is an ISPF provided parser. For details see the SCLM documentation.

Code Fragment 1 (Member=P$BMASTR SCLMMACS)

```
************************************************************************
* Change Activities:
* 950622 JP1 Created
************************************************************************
```

End of Code Fragment 1 (Member=P$BMASTR SCLMMACS)

B.2 Build Translators

B.2.1 B$BMANGR SCLMMACS - Create On-line Documentation

BUILD Translator Steps

Code Fragment 1 (Member=B$BMANGR SCLMMACS)

```
************************************************************************
* Change Activities:
* 950622 JP Created
************************************************************************
```

End of Code Fragment 1 (Member=B$BMANGR SCLMMACS)

Format the source to a BOOK output file to be read by BookManager. The REXX exec BMBUILD is used.

It follows the architecture member keyword assignments:

PARM1 To specify BookManager parameters. No parameters are predefined.

Code Fragment 2 (Member=B$BMANGR SCLMMACS)

```
*                        |----+----+----+-|                          *
        FLMTRNSL  CALLNAM=¢BookManager     ¢,                          C
              FUNCTN=BUILD,                                            C
              CALLMETH=LINK,                                           *
              COMPILE=SCLMISP,                                         *
              VERSION=D950110,                                         *
              OPTIONS=¢BMBUILD INDD=TEXTIN,OUTDD=BOOKOUT¢,             *
              PARMKWD=PARM1,                                           *
              GOODRC=0,                                                *
              PORDER=1
*
*-----------------------------------------------------------------------
*  DDNAME Allocations
*-----------------------------------------------------------------------
```

End of Code Fragment 2 (Member=B$BMANGR SCLMMACS)

SINC The source used to generate a book for online reading.

Code Fragment 3 (Member=B$BMANGR SCLMMACS)

```
*    - TEXTIN DDNAME - Translator Input  (PS or member)
       FLMALLOC  IOTYPE=S,DDNAME=TEXTIN,                              C
             KEYREF=SINC,                                             C
             RECFM=VB,LRECL=255,RECNUM=4500,CATLG=Y
*
```

End of Code Fragment 3 (Member=B$BMANGR SCLMMACS)

OUT2 The created BOOK format.

Code Fragment 4 (Member=B$BMANGR SCLMMACS)

```
*    - BOOKOUT DDNAME - Translator Output (PS or member)
       FLMALLOC  IOTYPE=O,DDNAME=BOOKOUT,                             C
             KEYREF=OUT2,                                             C
             RECFM=FB,LRECL=4096,BLKSIZE=28672,RECNUM=6000,CATLG=Y
*
*
*    --- Textpart library chain
*          Corresponds to SCRIPT SEARCH() command
       FLMALLOC  IOTYPE=I,DDNAME=TEXTLIB,                             C
             KEYREF=SINC
*
*
*    --- DCF Macro Libraries
*          Corresponds to SCRIPT LIB() command
       FLMALLOC  IOTYPE=A,DDNAME=SCRPTLIB
        FLMCPYLB  PC.BOOK.SEOYMC20
        FLMCPYLB  TEMP.DCF.MACLIB
*
       FLMALLOC  IOTYPE=A,DDNAME=SSGML
        FLMCPYLB  PC.BOOK.SEOYMC40
*
       FLMALLOC  IOTYPE=A,DDNAME=BOOKMGML
        FLMCPYLB  TEMP.DCF.MACLIB
*
*    --- DCF Profiles
*
       FLMALLOC  IOTYPE=A,DDNAME=SSPROF
        FLMCPYLB  PC.BOOK.SEOYMC40(DSMPROF4)
*
       FLMALLOC  IOTYPE=A,DDNAME=BMPROF
        FLMCPYLB  TEMP.DCF.MACLIB(EDFPROF)
*
```

```
*    --- PUBLIC Graphic and Image Libraries
*
         FLMALLOC  IOTYPE=A,DDNAME=ADMGDF
          FLMCPYLB  NULLFILE
*
         FLMALLOC  IOTYPE=A,DDNAME=ADMIMG
          FLMCPYLB  NULLFILE
*
         FLMALLOC  IOTYPE=A,DDNAME=CCITTG4
          FLMCPYLB  NULLFILE
*
         FLMALLOC  IOTYPE=A,DDNAME=EOYCGM
          FLMCPYLB  NULLFILE
*
         FLMALLOC  IOTYPE=A,DDNAME=EPS
          FLMCPYLB  NULLFILE
*
         FLMALLOC  IOTYPE=A,DDNAME=PSEG3820
          FLMCPYLB  PC.SCLMSITE.PSEGLIB
          FLMCPYLB  PC.BOOK.PSEGLIB
*
         FLMALLOC  IOTYPE=A,DDNAME=PSEG38PP
          FLMCPYLB  PC.SCLMSITE.PSEGLIB
          FLMCPYLB  PC.BOOK.PSEGLIB
*
         FLMALLOC  IOTYPE=A,DDNAME=PSEG4250
          FLMCPYLB  NULLFILE
*
         FLMALLOC  IOTYPE=W,DDNAME=EHZFILES,                          C
               RECFM=VB,LRECL=255,BLKSIZE=6160
*
*    --- GDDM Symbol Libraries:
*
         FLMALLOC  IOTYPE=A,DDNAME=ADMSYMBL
          FLMCPYLB  PC.BOOK.GDDMSYM
*
*    --- Build options data set
*
         FLMALLOC  IOTYPE=A,DDNAME=BLDOPTND
          FLMCPYLB  PC.BOOK.SEOYIPRF(IBMBOOKS)
*
*    --- BookManager Message file
*
         FLMALLOC  IOTYPE=A,DDNAME=MSGFILE
          FLMCPYLB  PC.BOOK.SEOYBENU(EOYMSGS)
*
*    --- Just some output files
*
         FLMALLOC  IOTYPE=W,DDNAME=SYSERR,PRINT=Y
         FLMALLOC  IOTYPE=W,DDNAME=SYSPRINT,PRINT=Y
         FLMALLOC  IOTYPE=W,DDNAME=SYSMSG,PRINT=Y

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

End of Code Fragment 4 (Member=B$BMANGR SCLMMACS)

B.2.2 B$BOOKIE SCLMMACS - Create Printable Documentation

BUILD Translator Steps

Code Fragment 1 (Member=B$BOOKIE SCLMMACS)

```
*********************************************************************
* Change Activities:
* 950622 JP created
*********************************************************************
```

End of Code Fragment 1 (Member=B$BOOKIE SCLMMACS)

Language definition for SCRIPT Processor with BookMaster to create LIST3820 format output. The ISPF provided translator FLMTMSI is used.

It follows the architecture member keyword assignments:

PARM0 To specify BookMaster source file. Following parameters are predefined:

FP(2) see BookMaster documentation
M(ID TRACE) see BookMaster documentation
CH(X0T00395) see BookMaster documentation
DEV(3820A4) see BookMaster documentation
SYS(H YES S 1 T YES X YES) see BookMaster documentation
CO see BookMaster documentation

Code Fragment 2 (Member=B$BOOKIE SCLMMACS)

```
*
*********************************************************************
*                        |----+----+----+-|
         FLMTRNSL  CALLNAM=¢BookMaster     ¢,                        C
               FUNCTN=BUILD,                                          C
               COMPILE=FLMTMSI,                                       C
               VERSION=4.0,                                           C
               GOODRC=8,                                              C
               PORDER=3,                                              C
               PARMKWD=PARM0,                                         C
               OPTIONS=¢@@FLMUID/I,FP(2),M(ID TRACE),CH(X0T00395),DEV(3C
               820A4),SYS(H YES S 1 T YES X YES),CO¢
*
*--------------------------------------------------------------------
*  DDNAME Allocations (using DDNAMELIST substitution)
*--------------------------------------------------------------------
```

End of Code Fragment 2 (Member=B$BOOKIE SCLMMACS)

SINC The source used to be formatted

Code Fragment 3 (Member=B$BOOKIE SCLMMACS)

```
*  1 - TEXTIN DDNAME - Translator Input  (PS or member)
*             coresponds to file name or SOURCDD option
        FLMALLOC  IOTYPE=S,DDNAME=TEXTIN,                          C
             KEYREF=SINC,                                          C
             RECFM=VB,LRECL=255,RECNUM=4500,CATLG=Y
*
```

End of Code Fragment 3 (Member=B$BOOKIE SCLMMACS)

OUT1 The output generated in LIST3820 format. It has a language of SHIPIT so that it can be distributed during a promote.

Code Fragment 4 (Member=B$BOOKIE SCLMMACS)

```
*  2 - TEXTOUT DDNAME - Translator Output (PS or member)
*             coresponds to DDNAME=SCRPTFIL or FILE( ) option
        FLMALLOC  IOTYPE=O,DDNAME=TEXTOUT,                         C
             KEYREF=OUT1,LANG=SHIPIT,                              C
             RECFM=VBM,LRECL=8205,RECNUM=6000,CATLG=Y
*  3 - TEXTMAC DDNAME - POs containing the needed macros
*             coresponds to DDNAME=SCRPTLIB or LIB( ) option
        FLMALLOC  IOTYPE=A,DDNAME=TEXTMAC
         FLMCPYLB  TEMP.DCF.MACLIB
*  4 - TEXTPROF DDNAME - BookMaster Profile
*                      (PS or member imbedded before primary)
*             coresponds to DDNAME=SCRPTPRO or PROFILE( ) option
        FLMALLOC  IOTYPE=A,DDNAME=TEXTPROF
         FLMCPYLB  TEMP.DCF.MACLIB(EDFPROF)
*  5 - TEXTMSGS DDNAME - Messages (to put out the messages produced)
        FLMALLOC  IOTYPE=W,DDNAME=TEXTMSGS,                        C
             RECFM=FBA,LRECL=133,RECNUM=1000,PRINT=Y
*
*  The following DDNAMEs are not substituted. They are predefined
*       by the script processor
*
*  --- TEXTLIB DDNAME - The imbedd and append Library  (PO)
*             coresponds to SEARCH( ) option
        FLMALLOC  IOTYPE=I,DDNAME=TEXTLIB,                         C
             KEYREF=SINC
*
*  --- SCRPTSEG DDNAME - PSEG libaries  (PO)
*             coresponds to SEGLIB( ) option
        FLMALLOC  IOTYPE=A,DDNAME=SCRPTSEG
         FLMCPYLB  PC.SCLMSITE.PSEGLIB
*
*  --- SCRPTFNT DDNAME - Font libraries  (PO)
*             coresponds to FONTLIB( ) option
        FLMALLOC  IOTYPE=A,DDNAME=SCRPTFNT
         FLMCPYLB  TEMP.MAIN.FONT3820
*
```

End of Code Fragment 4 (Member=B$BOOKIE SCLMMACS)

B.2.3 B$COPY SCLMMACS - Copy to SCLM Output Type

BUILD Translator Steps

Copy a source member to a distribution library

SINC The source to be copied
OUT0 The distribution library
PARM0 Additional parameters to be provided

B.2.4 B$DTLC SCLMMACS - Process DTL Source

BUILD Translator Steps

This step is used to process DTL source to create ISPF type output such as panels, messages and tables from DTL source.

SINC The DTL source member as input to be processed
PARM1 Additional parameters to the ISPDTLC step. The KEYAPPL= parameter should be specified at least.
OUT1 The ISPF panel created and to be saved
OUT2 The ISPF message created and to be saved
OUT3 The ISPF table created and to be saved
LIST The warning and error messages produced for member.

Right now if several outputs are produced with the same member name, then the last created messages are saved and the others are overwritten. In this case you should not have tags which create the same output member names from the same DTL input member.

B.2.5 B$LE370 SCLMMACS - Create a Load Module

BUILD Translator Steps

Code Fragment 1 (Member=B$LE370 SCLMMACS)

```
*********************************************************************
* Change Activities:
* 950622 JP Created
*********************************************************************
```

End of Code Fragment 1 (Member=B$LE370 SCLMMACS)

Invoke the Linkage editor IEWL.

It follows the architecture member keyword assignments:

PARM0 To specify Linkage Editor parameters. The following parameters are predefined:

DCBS Needed according to SCLM documentation
LIST To provide a listing for error investigation
MAP To get a module map for error investigation

LOAD The load module to be generated from the objects. The output has the language LOAD so that during promote it can be distributed to a production system.

For external references see C.3, “E#LE370 SCLMVIEW - External Load Libraries” on page 120.

LMAP The Linkmap listing. Will also go to the .LISTxx file.

B.2.6 B$RXPREP SCLMMACS - Preprocess a REXX Part

BUILD Translator Steps

Generate the executable REXX file invoking RXPREP program.

PARM0 Additional parameters for the RXPREP step

SINC The REXX main program as input to be processed

LIST Messages produced during processing

Copy input source to a PO member for next processing step.

Generate the REXX documentation files in BookMaster format using REXXFMT.

OUT1 The compiled REXX command file (not yet implemented)

OUT2 The created documentation file

B.2.7 B$SCLMOD SCLMMACS - Compile and Process SCLM Project Definition

BUILD Translator Steps

Code Fragment 1 (Member=B$SCLMOD SCLMMACS)

```
************************************************************************
* Change Activities:
* 950622 JP Created
************************************************************************
```

End of Code Fragment 1 (Member=B$SCLMOD SCLMMACS)

Create SCLM project object and BookMaster format from the SCLM source.

It calls REXX2LLD, which itself calls the Assembler to create the object output.

Depending on the parameters provided by an architecture member, the allocation information to allocate additional PDS will be produced also.

Code Fragment 2 (Member=B$SCLMOD SCLMMACS)

```
* -- Define Build Translator(s)
*
          FLMTRNSL   CALLNAM=¢SCLMMOD  Process¢,                    *
               VERSION=1.0,                                         *
               FUNCTN=BUILD,                                        *
               CALLMETH=LINK,                                       *
               COMPILE=SCLMISP,                                     *
               OPTIONS=¢REXX2LLD @@FLMPRJ.@@FLMGRP.@@FLMTYP(@@FLMMBR)  *
               (DDIN(SYSIN),DDOUT(REXX2LLD),DDJCL(ALLOCJCL),        *
               +ASM<NOTERM,NODECK,OBJECT,XREF(FULL),ESD,RLD,LIST,RENT>,*
               +SCLM,DDSCLM(SCLMINFO),+CODE,DDSLIB(SYSLIB),+ALLOC¢,  *
               PARMKWD=PARM0,                                       *
               GOODRC=0,                                            *
               PORDER=1
```

End of Code Fragment 2 (Member=B$SCLMOD SCLMMACS)

It follows the architecture member keyword assignments:

Code Fragment 3 (Member=B$SCLMOD SCLMMACS)

```
*  Allocations for the Translator (Not positional, Explicit DDNAMES)
*-----------------------------------------------------------------------
*
*  REXX2LLD related DDNAMES:
*
```

End of Code Fragment 3 (Member=B$SCLMOD SCLMMACS)

OUT1 The BookMaster format to be used in a document. The language SCRIPT is associated with it to get dependency information for those generated parts. They are used as imbed members.

Code Fragment 4 (Member=B$SCLMOD SCLMMACS)

```
*  --REXX2LLD-- for REXX2LLD exec output
          FLMALLOC  IOTYPE=O,DDNAME=REXX2LLD,                       *
               KEYREF=OUT1,LANG=SCRIPT,                             *
               RECFM=VB,LRECL=255,RECNUM=9000
*
```

End of Code Fragment 4 (Member=B$SCLMOD SCLMMACS)

OUT2 The allocation JCL to create new PDSs. The language SHIPIT is associated with it to allow the generated members to be distributed during promote via the shipit mechanism.

Code Fragment 5 (Member=B$SCLMOD SCLMMACS)

```
*  --ALLOCJCL-- for allocation PDS JCL output
          FLMALLOC  IOTYPE=O,DDNAME=ALLOCJCL,                    *
              KEYREF=OUT2,LANG=SHIPIT,                           *
              RECFM=FB,LRECL=80,RECNUM=9000
*
*  --SCLMINFO-- for SCLMINFO exec output
          FLMALLOC  IOTYPE=W,DDNAME=SCLMINFO,                    *
              RECFM=VB,LRECL=255,RECNUM=9000
*
*  Assembly related DDNAMES:
*
*
```

End of Code Fragment 5 (Member=B$SCLMOD SCLMMACS)

SINC The source to be processed by this language

Code Fragment 6 (Member=B$SCLMOD SCLMMACS)

```
*  --SYSIN   --  Input to be processed
          FLMALLOC  IOTYPE=S,DDNAME=SYSIN,                       *
              KEYREF=SINC,                                       *
              RECFM=FB,LRECL=80,RECNUM=9000
*
*  --SYSLIB  -- for assembler
          FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,                      *
              KEYREF=SINC
*
```

End of Code Fragment 6 (Member=B$SCLMOD SCLMMACS)

OBJ The object deck resulting from the assembly of the input source

Code Fragment 7 (Member=B$SCLMOD SCLMMACS)

```
*  --SYSLIN  -- for assembler
          FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,                      *
              KEYREF=OBJ,                                        *
              RECFM=FB,LRECL=80,RECNUM=15000
*
```

End of Code Fragment 7 (Member=B$SCLMOD SCLMMACS)

LIST The assembly listing

Code Fragment 8 (Member=B$SCLMOD SCLMMACS)

```
*  --SYSPRINT-- for assembler
          FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,                          *
              KEYREF=LIST,                                             *
              RECFM=FBM,LRECL=121,RECNUM=60000,PRINT=Y
*
*  --SYSUT1  -- for assembler
          FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,                            *
              RECFM=FB,LRECL=80,RECNUM=15000
*
*  --SYSPUNCH-- for assembler
          FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
           FLMCPYLB NULLFILE
*
*  --SYSTERM -- for assembler
          FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
           FLMCPYLB NULLFILE
```

End of Code Fragment 8 (Member=B$SCLMOD SCLMMACS)

B.2.8 B$SCLMAC SCLMMACS - Process SCLM Project Definition

BUILD Translator Steps

Create BookMaster format from the SCLM source using REXX2LLD.

It follows the architecture member keyword assignments:

SINC The source to be processed by this language

OUT1 The BookMaster format to be used in a document. The output stored has the language SCRIPT assigned to resolve dependencies.

B.3 Copy Translators

B.3.1 C$SHIPIT SCLMMACS - General Promote Exit

COPY Translator Steps

Code Fragment 1 (Member=C$SHIPIT SCLMMACS)

```
************************************************************************
* Change Activities
* 950619 JP Created
************************************************************************
```

End of Code Fragment 1 (Member=C$SHIPIT SCLMMACS)

A Copy Translator to distribute members during promote according to a list provided in PDS @@FLMPRJ.PROJDEFS.SHIPLIST. For details about the mechanism see the description of the SHIPIT exec.

Code Fragment 2 (Member=C$SHIPIT SCLMMACS)

```
*
* -- Copy  Translator(s)     |----+----+----+-|
          FLMTRNSL   CALLNAM=¢Ship a part     ¢,                        *
               FUNCTN=COPY,                                             *
               CALLMETH=LINK,                                           *
               COMPILE=SCLMISP,                                         *
               OPTIONS=¢%SHIPIT @@FLMPRJ,@@FLMGRP,@@FLMTOG,@@FLMTYP,    *
               @@FLMMBR=>¢,                                             *
               VERSION=D940113,                                         *
               GOODRC=0,                                                *
               PDSDATA=Y,                                               *
               PORDER=1
```

End of Code Fragment 2 (Member=C$SHIPIT SCLMMACS)

Appendix C. Include Library Definitions

This chapter lists all the include library definitions we refer to in this document and shows how we use such definitions.

C.1 E#SCLM SCLMVIEW - External SCLM Macro Libraries

Code Fragment 1 (Member=E#SCLM SCLMMACS)

```
************************************************************************
*  Start of ¢SCLMMACS(E#SCLM)¢
************************************************************************
*
*  Change Activities see $hist tags
*
*  950622 JP Created
*  951120 JP Put in correct ISPF 4.1 macro libraries
************************************************************************
```

End of Code Fragment 1 (Member=E#SCLM SCLMMACS)

External library chain to be used

No dependency checking will be done for members in these libraries.

The library chain is:

SYS4.ISPF.V410.SISPMACS	SCLM Macros of ISPF 4.1.0
SYS1.MACLIB	System Macros

Code Fragment 2 (Member=E#SCLM SCLMMACS)

```
*        V     V
*---+----1----+----2----+----3----+----4----+----5----+----6----+----7-
          MACRO
&LABEL    E#SCLM   &LBMACRO=FLMCPYLB
.*
&LABEL    &LBMACRO  SYS4.ISPF.V410.SISPMACS
          &LBMACRO  SYS1.MACLIB
          MEND
************************************************************************
*  End of  SCLMMACS(E#SCLM)
************************************************************************
```

End of Code Fragment 2 (Member=E#SCLM SCLMMACS)

C.2 I$SCLM SCLMVIEW - Internal SCLM Macro Libraries

Code Fragment 1 (Member=I$SCLM SCLMMACS)

```
*********************************************************************
*  Change Activities:
*  950622 JP Created
*********************************************************************
```

End of Code Fragment 1 (Member=I$SCLM SCLMMACS)

SCLM controlled libraries to be accessed

@@FLMTYP	The input source type
@@FLMETP	The extended type
SCLMVIEW	The view parts
SCLMLANG	The language parts
SCLMMACS	The macro and translator parts

Code Fragment 2 (Member=I$SCLM SCLMMACS)

```
FLMINCLS TYPES=(@@FLMTYP,@@FLMETP,SCLMVIEW,SCLMLANG,SCLMMACS)
```

End of Code Fragment 2 (Member=I$SCLM SCLMMACS)

C.3 E#LE370 SCLMVIEW - External Load Libraries

Code Fragment 1 (Member=E#LE370 SCLMMACS)

```
*********************************************************************
*  Change Activities:
*  950622 JP Created
*********************************************************************
```

End of Code Fragment 1 (Member=E#LE370 SCLMMACS)

External library chain to be used.

No dependency checking will be done for members in these libraries.

The library chain is:

SYS4.ISPF.V410.SISPLOAD	ISPF System objects

Code Fragment 2 (Member=E#LE370 SCLMMACS)

```
*        V     V
*---+----1----+----2----+----3----+----4----+----5----+----6----+----7-
          MACRO
&LABEL    E#LE370  &LBMACRO=FLMCPYLB
.*
&LABEL    &LBMACRO  SYS4.ISPF.V410.SISPLOAD   IT provided
          MEND
```

End of Code Fragment 2 (Member=E#LE370 SCLMMACS)

C.4 I$SCRIPT SCLMVIEW - Internal Documentation Libraries

Code Fragment 1 (Member=I$SCRIPT SCLMMACS)

```
************************************************************************
*  Change Activities
*  950622 JP Created
************************************************************************
```

End of Code Fragment 1 (Member=I$SCRIPT SCLMMACS)

SCLM controlled libraries to be accessed

@@**FLMTYP**	The input source type
@@**FLMETP**	The extended type
SCLMGMLO	The generated SCLM source
BMMACROS	The local available BookMaster macros
BMSTYLES	The local available BookMaster styles

Code Fragment 2 (Member=I$SCRIPT SCLMMACS)

```
        FLMINCLS TYPES=(@@FLMTYP,@@FLMETP,SCLMGMLO,BMMACROS,BMSTYLES)
```

End of Code Fragment 2 (Member=I$SCRIPT SCLMMACS)

Appendix D. Translator Functions and Utilities

This chapter lists all the major translator functions used by our build and promote translators and some utilities we provide in our environment. For the most only the users guide information is provided because printing the complete listings would extend the scope of this documentation. However, the reader may contact the authors for more detail.

D.1 SCLMCPP - Invoke the MVS C++ Compiler

```
/* REXX */
/*********************************************************************/
/* Description:                                                      */
/*    Exec to call the C++ compilation exec under TSO.               */
/*    Called by SCLM CPP translator during execution.                */
/*                                                                   */
/* Change Activity:                                                  */
/*    1.0 GB 01 MAR 95 Initial Version                               */
/*    1.1 GB 01 MAY 95 New Compiler Version                          */
/*    1.2 GB 20 JUL 95 Replace USERLIB by USERPATH for improved      */
/*                        flexibility (bilingual headers in .MAC)    */
/*    1.3 GB 22 SEP 95 User lib/lng in SYSPATH                       */
/*    1.4 GB 26 SEP 95 Support all macros in SYSPATH with            */
/*                        NOUSERPATH option, Building Blocks Rel. 2  */
/*                                                                   */
/*********************************************************************/
Trace O
Parse Upper Arg Flmmbr†,†Arguments
Lng=†SYS1.CMVS.V310†
Lib=†SYS1.LE.V150†
Bbl=†RMF.BBXX20†
Flmmbr = Strip(Flmmbr,†B†)

Address †TSO†
savemsg = MSG()
xxxxmsg = MSG(†OFF†)
†FREE  FI(SYSPRINT)†
xxxxmsg = MSG(savemsg)
†ALLOC FI(SYSMSGS)  DA(¢†Lng†.SCBC3MSG(EDCMSGE)¢) SHR REU†
†ALLOC FI(SYSXMSGS) DA(¢†Lng†.SCBC3MSG(CBCMSGE)¢) SHR REU†

†ALLOC FI(PDSIN) NEW SP(10 10) TR LRECL(255) RECFM(V B) DIR(1)     REUSE†
†ALLOC FI(OPTS)  NEW SP(1 1)   TR LRECL(80)  RECFM(F B) BLKS(3120) REUSE†

Address †ISPEXEC† †CONTROL ERRORS RETURN†
Address †ISPEXEC† †LMINIT DATAID(CPP) DDNAME(CPPIN) ENQ(SHR)†
Address †ISPEXEC† †LMINIT DATAID(PDS) DDNAME(PDSIN) ENQ(SHR)†
Address †ISPEXEC† †LMCOPY FROMID(†cpp†) TODATAID(†pds†)† ,
  †REPLACE TOMEM(†Flmmbr†)†
Address †ISPEXEC† †LMFREE DATAID(†cpp†)†
Address †ISPEXEC† †LMFREE DATAID(†pds†)†
```

```
†LISTDDN PDSIN    BACK CLIST†; Indsn   = Lddsn1†(††flmmbr†)†
†LISTDDN CPPLIN   BACK CLIST†; Lindsn  = Lddsn1
†LISTDDN CPPULIB  BACK CLIST†; Userlibs= Ldstring
†LISTDDN CPPCPRT  BACK CLIST†; CprtDsn = Lddsn1

Opts = † † Translate(Arguments,† †,†,†)
Select
  When (Pos(† NOSOURCE†,Opts) <> 0)
  Then Opts=Replace(Opts,† NOSOURCE†,† SOURCE(¢†Cprtdsn†¢)†)
  When (Pos(† SOURCE†,Opts) <> 0)
  Then Opts=Replace(Opts,† SOURCE†,† SOURCE(¢†Cprtdsn†¢)†)
  Otherwise Opts=Opts† SOURCE(¢†Cprtdsn†¢)†
End

Select
  When (Pos(† NOLIST†,Opts) <> 0) Then Nop
  When (Pos(† LIST†,Opts) <> 0)
  Then Opts=Replace(Opts,† LIST†,† LIST(¢†Cprtdsn†¢)†)
  Otherwise Opts=Opts† LIST(¢†Cprtdsn†¢)†
End

Select
  When (Pos(† NOOBJECT†,Opts) <> 0)
  Then Opts=Replace(Opts,† NOOBJECT†,† OBJECT(¢†Lindsn†¢)†)
  When (Pos(† OBJECT†,Opts) <> 0)
  Then Opts=Replace(Opts,† OBJECT†,† OBJECT(¢†Lindsn†¢)†)
  Otherwise Opts=Opts† OBJECT(¢†Lindsn†¢)†
End

If (Pos(† NOUSERPATH†,Opts) <> 0) Then Do
   Opts=Replace(Opts,† NOUSERPATH†,† †)
   NoUser = 1
End
Else NoUser = 0

Userpath = ††
Do i = 1 To Words(Userlibs)
   Parse Value Subword(Userlibs,i,1) With †¢†Ulib†¢†
   Parse Value Reverse(Ulib) With †.†Ulib
   Ulib = Translate(Reverse(Ulib),†/†,†.†)
   If i > 1 Then Userpath = Userpath || †,†
   Userpath = Userpath || †/† || Ulib
End
libpp = Translate(lib,†/†,†.†)
lngpp = Translate(lng,†/†,†.†)
bblpp = Translate(bbl,†/†,†.†)

opt.0 = Words(Opts)
Do i = 1 To opt.0
   opt.i = Subword(Opts,i,1)
/* Say †OPTFILE Line(†Right(i,2,†0†)†)=† opt.i */
End
†EXECIO† opt.0 †DISKW OPTS (FINIS STEM OPT.†
Opts = †OPTFILE(DD:OPTS)†

Cmd = †CBC310PP /FAST† Lng Lib †¢†Indsn†¢† Opts †NOTERMINAL†
```

```
If NoUser
   Then Cmd = Cmd ,
      †SYSPATH(/†bblpp†,/†libpp†/SCEEH,/†lngpp†/SCLB3H,†Userpath†)†
   Else Cmd = Cmd †SYSPATH(/†libpp†/SCEEH,/†lngpp†/SCLB3H)† ,
      †USERPATH(†Userpath†)†


Address †TSO† Cmd
xrc = Rc
†FREE FI(SYSMSGS,SYSXMSGS,PDSIN,OPTS)†

/*-------------------------------------------------------------------*/
/* Incorrect RECFM/LRECL/BLKSIZE of C++ Listing data set is changed: */
/* Read Listing, delete data set, rewrite with correct attributes    */
/*-------------------------------------------------------------------*/
†FREE FILE(CPPCPRT)†
†ALLOC FI(XXTEMP) DA(¢†cprtdsn†¢) SHR REUSE†
†EXECIO * DISKR XXTEMP (FINIS STEM F.†
†FREE FI(XXTEMP)†
x = MSG(†OFF†); †DELETE ¢†cprtdsn†¢†; x = MSG(x)
†ALLOC FI(CPPCPRT) DA(¢†cprtdsn†¢) NEW SP(10 10) TR DIR(0) LRECL(137)†,
   † BLKSIZE(0) RECFM(V B A) DSORG(PS)†
†EXECIO† f.0 †DISKW CPPCPRT (FINIS STEM F.†
Drop f.

/*-------------------------------------------------------------------*/
/* Incorrect RECFM/LRECL/BLKSIZE of C++ Object data set is changed:  */
/* Read Listing, delete data set, rewrite with correct attributes    */
/*-------------------------------------------------------------------*/
†FREE FI(CPPLIN)†
†ALLOC FI(XXTEMP) DA(¢†lindsn†¢) SHR REUSE†
†EXECIO * DISKR XXTEMP (FINIS STEM F.†
†FREE FI(XXTEMP)†
x = MSG(†OFF†); †DELETE ¢†lindsn†¢†; x = MSG(x)
†ALLOC FI(CPPLIN) DA(¢†lindsn†¢) NEW SP(10 10) TR DIR(0) LRECL(80)†,
   † BLKSIZE(3200) RECFM(F B) DSORG(PS)†
†EXECIO† f.0 †DISKW CPPLIN (FINIS STEM F.†
Drop f.

Exit xrc

/*-------------------------------------------------------------------*/
/* Replaces a substring in a given input string by a new string      */
/* Invocation:  result = REPLACE(string,old,new)                     */
/* with:        string = input string                                */
/*              old    = substr of string which should be replaced   */
/*              new    = replacement of old                         */
/*-------------------------------------------------------------------*/
```

```
Replace: Procedure;
   If Arg() < 3 Then Return ††           /* Invalid arguments ?     */
   str = Arg(1)                          /* String to be searched   */
   old = Arg(2)                          /* Pattern to be searched  */
   new = Arg(3)                          /* Replace string          */
   lold = Length(old)                    /* Length of old substr    */
   lnew = Length(new)                    /* Length of new substr    */
   spos = 1                              /* Start position search   */
   pold = Pos(old,str)                   /* Find pattern position   */
   Do While (pold > 0)                   /* Found pattern !         */
      left = Substr(str,1,pold+spos-2)   /* Left substring          */
      rght = Substr(str,pold+lold+spos-1) /* Right substring        */
      str = left || new || rght          /* Build new string        */
      spos = pold+lnew+spos-1            /* New start position      */
      pold = Pos(old,Substr(str,spos))   /* Find pattern position   */
   End;
   Return str
```

D.2 MAKEARCD - Generate Architecture Definitions from Models

This exec is used to generate architecture members for source members which are subject to migration to SCLM. How to use the exec and which information must be available is described below.

```
MAKEARCD  migrate_type  mig_info_member
```

migrate_type
This is a name from the second column in the file mig_info_member as seen in Figure 24 on page 127.

mig_info_member
This is the name of the member as in Figure 24 on page 127.

```
* Migration information used to migrate HCD to SCLM
*
*
<SCLM_project   > HCD
<MIG_group      > MIG
<SCLM_group     > BASE
*
<dist_lists     > HCD.PROJDEFS.MEMINFO
<model_archdef  > HCD.PROJDEFS.MODELS.ARCHDEF
<report         > HCD.PROJDEFS.MIGMSGS
*
*Status    |Migrate |ARCHDEF   |Target  |Distlib   |process|grouping           |ARCHDEF |
*          |type    |Model     |lib type|type      |type   |type_name          |EXEC    |
*========= |========|========= |========|==========|=======|==========         |========|
 ADF 941126|PWR     |PWR01     |-/-     |PHASE     |LINKDEF| PRODDEF(PWR)      |VSELINK |
 ADF 941126|PLI     |PLI       |MOD     |=>CBDJ0001|COMPDEF| LINKDEF           |LNK2ARCD|
 ADF 950811|MACEXT  |MACEXT    |MAC     |1:AMODGEN |MACDEF | PRODDEF(MACEXT  ) |MAKEARCD|
 ADF 950811|MACINT  |MACINT    |MAC     |1:AMACLIB |MACDEF | PRODDEF(MACINT  ) |MAKEARCD|
 ADF 941126|CLI     |CLIST     |CLIST   |0:ACBDCLST|MISCDEF| PRODDEF(CLISTS  ) |MAKEARCD|
 ADF 950811|JCL     |JCL       |JCL     |=>JCL     |MISCDEF| PRODDEF(JCL     ) |MAKEARCD|
 ADF 950811|SAMPLE  |SAMPLE    |JCL     |1:ASAMPLIB|MISCDEF| PRODDEF(SAMPLE  ) |MAKEARCD|
 ADF 941126|DTLHLP  |DTLHLP    |HELPS   |O:ACBDHENU|HELPDEF| DISPDEF(HELPS   ) |MAKEARCD|
 ADF 941126|DTLPNL32|DTLPNL32  |PANELS  |1:ACBDPENU|PANLDEF| DISPDEF(PANEL32 ) |MAKEARCD|
 ADF 941126|DTLPNL41|DTLPNL41  |PANELS  |1:ACBDPENU|PANLDEF| DISPDEF(PANEL41 ) |MAKEARCD|
 ADF 941126|ISPFPNL |ISPFPNLS  |PANELS  |0:ACBDPENU|PANLDEF| DISPDEF(PANELISP) |MAKEARCD|
 ADF 941126|PANELS  |ISPFPNLS  |PANELS  |0:ACBDPENU|PANLDEF| DISPDEF(HELPSISP) |MAKEARCD|
 ADF 941126|INCFPNL |DTLMAC    |DTLMAC  |-/-       |-/-    | DISPDEF(DTLMACS1) |MAKEARCD|
 ADF 941126|INCHLP  |DTLMAC    |DTLMAC  |-/-       |-/-    | DISPDEF(DTLMACS2) |MAKEARCD|
 ADF 941126|ISPFMSG |ISPFMSGS  |MSGS    |0:ACBDMENU|MSGSDEF| DISPDEF(MSGSISPF) |MAKEARCD|
 ADF 950208|MSGSRCA |MSGSCRE   |MSGS    |2:ACBDHENU|MSGSDEF| DISPDEF(MSGSA   ) |MAKEARCD|
 ADF 950811|TAB     |TABLES    |TABLES  |1:ACBDTENU|TABLDEF| DISPDEF(TABLES  ) |MAKEARCD|
```

Figure 24. MAKEARCD: Example of <mig_info_member>

Lines starting with an * are treated as comments.

You have to define some global values such as:

<SCLM_project >
: The HLQ of the project containing all the data

<MIG_group >
: The second qualifier to hold all the source to be migrated. It could be the same as an SCLM controlled data set.

<SCLM_group >
: The second qualifier of an SCLM controlled data set to where all the source will be migrated and architecture definition files created.

<dist_lists >
: A fully qualified PDS name, but without quotes which may contain member lists to overwrite data provided in the columns below. For the layout of such members refer to the example in Figure 26 on page 131.

<model_archdef >
: A fully qualified PDS name, but without quotes which contains the model members to create the architecture members for the members stored in *<SCLM_project>.<mig_group>.<mig_typ>*. An example of a model member can be seen in Figure 25 on page 130.

<model_archdef >
: A fully qualified PDS name, but without quotes which contains the members with msgs generated during the process for later analysis and documentation.

The meaning and use of the columns are as follows:

Status Just some information for you to state the work you did against the data described in this line.

Migrate type The third qualifier of the migration source PDS containing the source members. This makes the PDS name *<SCLM_project>.<MIG_group>.<migrate_type>*

ARCHDEF model The name of the member of the PDS specified in global info <model_archdef> containing the model to create the architecture member for the members stored in the migrate type.

Target lib type The type to where the member should go to be in an SCLM controlled library. This makes normally the PDS name. *<SCLM_project>.<SCLM_group>.<migrate_type>* where the source will be stored. This will be used to overwrite the placeholder <sinc_type> in a model member of a SINC statement.

Distlib type This information is the distribution information to be used to create the architecture member. Following formats are available:

- =>member

  This refers to another member to take member based information to create the architecture definition member.

  If the “ARCHDEF exec” is LNK2ARCD it refers to the Linkage Edit JCL containing the information to create the LEC architecture controls.

  If the “ARCHDEF exec” is MAKEARCD it refers to a member of the PDS specified in the global info <disl_lists>. For an example see Figure 26 on page 131.

- n:member
- -/-

process type The third qualifier to hold the SCLM CC members.

grouping type name
The third qualifier to hold the SCLM LEC members or the HL members to group the CC members.

ARCHDEF Exec The exec that should be invoked to do the generation of the architecture members.

To explain how the exec is used an example is provided.

Assume you have the following source PDS to be migrated to SCLM controlled PDSs with the name:

<SCLM_project>.<mig_group>.<mig_typ>.

For the example the source to store into SCLM is located in:

HCD.MIG.ASM	HCD.MIG.PLI
HCD.MIG.MACEXT	HCD.MIG.MACINT
HCD.MIG.CLI	HCD.MIG.JCL
HCD.MIG.SAMPLE	HCD.MIG.DTLHLP

HCD.MIG.DTLPNL32
HCD.MIG.DTLPNL41
HCD.MIG.ISPFPNL
HCD.MIG.PANELS
HCD.MIG.INCFPNL
HCD.MIG.INCHLP
HCD.MIG.ISPFMSG
HCD.MIG.MSGSRCA
HCD.MIG.TAB

Each type contains members of the same SCLM language and potentially the same corresponding architecture definition.

To migrate the sources in *<SCLM_project>.<mig_group>.<mig_typ>* to SCLM and to create the associated architecture members in *<SCLM_project>.<SCLM_group>.<process_type>* proceed as follows:

1. Setup
   a. First create a member in the primary control file, let's say HCD.MIGCNTL.MIGINFO(MIGINFO) as the one shown in Figure 24 on page 127.
   b. Then create the model architecture members as defined in column three of Figure 24 on page 127. For examples of model members see Figure 25 on page 130.
   c. If you refer to member lists in the primary control file then create those members. For an example see Figure 26 on page 131.
2. Execution

   It is assumed that your target data set is an SCLM controlled editable group.

   For each line or selective line do the following as is necessary:

   a. Copy members from *<SCLM_project>.<mig_group>.<migrate_type>* to the PDS *<SCLM_project>.<SCLM_group>.<target_lib_type>*

      This is not done by the exec. It must be done with copy utilities provided. For example copy HCD.MIG.MACINT to HCD.BASE.MACINT.

   b. Migrate all members copied with the language to be associated. For example copy HCDV5.MIG.MACINT to HCD.BASE.MACINT

      Currently it is assumed that all members of one migrate_type have the same language. There is no support by the exec to do the migration. The SCLM dialog to migrate members is used instead.

   c. Create the architecture members for each member

      This is currently supported differently.

      1) For PLI and ASM of MVS products the link JCL job provided is used as input currently for the LNK2ARCD exec. The call of LNK2ARD will be integrated in this exec to have one common interface for all architecture definition generations.

      2) For all the others issue this exec which takes the ARCHDEF model provided in *<SCLM_project>.PROJDEFS.ARCHDEF.MODELS(<language>)*, produces the archdef and stores it into *<SCLM_project>.<SCLM_group>.<process_type>*

         In addition it adds the name of the produced process_type member into the list member of *<SCLM_project>.<SCLM_group>.<grouping_type_name>*

For example:

```
MAKEARCD MACINT HCD.MIGCNTL.MIGINFO(MIGINFO)
```

This will then create architecture members into type MACDEF.

It will format the following model statements (OUTx is OUT0 to OUT9) by replacing the <variables> and alignment of the words.

```
SINC   <member> <sinc_type> commentary
OBJ    <member> <distlib>  commentary
LOAD   <module> <distlib>  commentary
OUTx   <member> <distlib>
LIST   <member> <type>
COPY   <member> <type>    commentary
```

```
*
* Member SCLM1.MIGCNTL.MODELS.ARCHDEF(ASM)
*
* Compile Assembler Source
*
SINC     <member> MOD            -- ASM Source
OUT4     <member> LISTASM        -- ASM/H listing
OBJ      <member> <distlib>      -- Object deck
COPY     ASM$     COMPDEF        -- Global ASM options
COPY     PRE$     COMPDEF        -- Global Prelink options
*
```

Figure 25 (Part 1 of 3). Architecture Model Member Examples

```
*
* Member SCLM1.MIGCNTL.MODELS.ARCHDEF(MSGSCRC0)
*
* Info to create Message parts from HCD message source
*
 SINC   <member> MSGS       -- The HCD message source
*
*    For empty messages OUT0 and OUT1 must be commented out
*
*OUT0   <member> SCBDHENU  -- The final HELP linked module for testing
*OUT1   <member> ACBDHENU  -- The prelinked HELP module to be shipped
 OUT2   <member> IMBEDS    -- BookMaster imbed file
 OUT3   <member> DTLMSGS   -- DTL message format
 LIST   <member> DTLMLOG   -- Error and Warning messages produced
*
* Parameters:
*
 COPY   DTL$MSG  DISPDEF  *  Common parameters for HCD message members
*
```

Figure 25 (Part 2 of 3). Architecture Model Member Examples

```
*
* Member SCLM1.MIGCNTL.MODELS.ARCHDEF(VSELNK  )
*
* Model to create LINKDEFs for VSE
*
*
 LOAD   <module> <distlib>   -- The phase to be produced
 OUT1   <module> ACBDHENU  -- The prelinked HELP module to be shipped
 OUT2   <module> IMBEDS    -- BookMaster imbed file
 OUT3   <module> DTLMSGS   -- DTL message format
 LIST   <module> DTLMLOG   -- Error and warning messages produced
*
* Parameters:
*
 COPY   DTL$MSG  DISPDEF  *  Common parameters for HCD message members
*
```

Figure 25 (Part 3 of 3). Architecture Model Member Examples

The model members contain some variables such as:

<member> The name of the member to be processed

<distlib> The primary distlib to move the primary output. If one source has several outputs to put in several distlib types additional info must be provided through a member mapping list. This member mapping list is also necessary if not all members in a <sclm_mig> group do not go to the same <distlib>.

n: means all go to same distlib

=> means refer to the PDS member *<SCLM_project>.PROJDEFS.MEMINFO(<distlib_type>)*

```
* Member migration information used to migrate HCD to SCLM
*
*
*Status    |Migrate |Archdef   |Target  |Distlib   |process|grouping          |ARCHDEF |
*          |member  |Model     |lib type|type      |type   |type_name         |EXEC    |
*========= |========|========= |========|==========|=======|==========        |========|
 MIG 941021|CBDJCMPR|JCL       |        |0:APROCLIB|       | PRODDEF(JCL     ) |        |
 MIG 941021|CBDJIMPT|JCL       |        |0:APROCLIB|       | PRODDEF(JCL     ) |        |
 MIG 941021|CBDJIOCP|JCL       |        |0:APROCLIB|       | PRODDEF(JCL     ) |        |
 MIG 941024|CBDJRPTS|JCL       |        |0:APROCLIB|       | PRODDEF(JCL     ) |        |
 MIG 941025|CBDJ0001|JCL       |        |0:APROCLIB|       | PRODDEF(JCL     ) |        |
 redo      |CBDJ0002|JCL       |        |0:APROCLIB|       | PRODDEF(JCL     ) |        |
 redo      |CBDSALIO|JCL       |        |0:APROCLIB|       | PRODDEF(JCL     ) |        |
 redo      |CBDSCNT2|JCL       |        |0:APROCLIB|       | PRODDEF(JCL     ) |        |
 redo      |CBDSCNT3|JCL       |        |0:APROCLIB|       | PRODDEF(JCL     ) |        |
 redo      |CBDSCPIO|JCL       |        |0:APROCLIB|       | PRODDEF(JCL     ) |        |
 MIG 941025|CBDSEXPA|JCL       |        |0:APROCLIB|       | PRODDEF(JCL     ) |        |
 MIG 941025|CBDSEXPU|JCL       |        |0:APROCLIB|       | PRODDEF(JCL     ) |        |
 MIG 941026|CBDSUDT |SAMPLE    |        |0:ASAMPLIB|       | PRODDEF(SAMPLE  ) |        |
 MIG 941026|CBDSUIM |SAMPLE    |        |0:ASAMPLIB|       | PRODDEF(SAMPLE  ) |        |
```

Figure 26. MAKEARCD: Example of <dist_lists> Member. Reference = > JCL of Figure 24 on page 127.

D.3 LNK2ARCD - Generate Architecture Definitions from LINK JCL

Create from a link JCL job the Architecture Definition members to compile the source and to link the modules.

The exec creates:

- LEC Archdefs for Linkage Editor build
- CC Archdefs for compile build
- HL Archdefs for application build

The invocation syntax is

```
LNK2ARCD input_profile
```

input_profile The member containing the parameters to be used by the exec to create the architecture definitions. A sample of such a member and the description of the parameters is shown in Figure 27 on page 133.

```
*-------------------------------------------------------------------
*
* Information to be used by the LNK2ARCD EXEC to create
* architecture definition members
*
*-------------------------------------------------------------------
* INDSN    : <the PDS name where the LINK JCL member is stored>
* INMEM    : <the member name, if not specified, FMID name is used>
* FMID     : <the FMID related to the LinkJCL>
* MODINFO  : <A member which contains mapping info for module list>
*
INDSN    : BPRA.TEST.LINKJCL
INMEM    :
FMID     : HTE24D2J
MODINFO  : TSO1111.PROJDEFS.MIGINFO(MODINFOS)
*-------------------------------------------------------------------
*
* PROJECT  : <the highlevel qualifier to store the archdefs>
* GROUP    : <the second    qualifier to store the archdefs>
* CCTYPE   : <the third     qualifier to store the compile archdefs>
* LECTYPE  : <the third     qualifier to store the link    archdefs>
* PRODTYPE : <the third     qualifier to store the product archdefs>
*
PROJECT  : BPRA
GROUP    : TSOTEST
CCTYPE   : COMPDEF
LECTYPE  : LINKDEF
PRODTYPE : PRODDEF
*-------------------------------------------------------------------
*
*        Seek for the statement that indicates a link step
* JCLIN: START= {word_pos},{start_pos in word}!{search_string} & {...}
*        Seek for the statement that contains the step name info
* JCLIN: STEP = {prefix_string}<{just a comment}>{postfix string}
*        Seek for the statement that contains the link parameters
* JCLIN: PARMS= {prefix_string}<{just a comment}>{postfix string}
*        Seek for the statement that contains the load lib type name
* JCLIN: LOAD = {prefix_string}<{just a comment}>{postfix string}
*
*
JCLIN: START= 2,1!EXEC & 3,1!LINKS
JCLIN: START= 2,1!EXEC & 3,1!PGM=IEWL
JCLIN: STEP = //<step> EXEC LINKS,
JCLIN: STEP = //<step> EXEC PGM=IEWL
JCLIN: PARMS= PARM=¢<parms>¢
JCLIN: LOAD = NAME=<load_lib>,
```

Figure 27 (Part 1 of 3). LNK2ARCD: Example of <input_profile>

```
*-------------------------------------------------------------------
*
* List of things to be generated in compile archdefs for assembler
* programs
*
* ASM: {archdef keyword} = {associated type} {associated comment}
*
ASM: SINC = MOD       -- ASM Source
ASM: LIST = LISTASMH  -- ASM/H listing
ASM: OBJ  = OBJ       -- Object deck
*-------------------------------------------------------------------
*
* List of things to be generated in compile archdefs for PLX programs
*
* PLX: {archdef keyword} = {associated type} {associated comment}
*
PLX: SINC = MOD       -- Source         --
PLX: LIST = LISTPLX   -- PL/X listing   --
PLX: OBJ  = OBJ       -- Object deck    --
PLX: OUT1 = SYSLOGIC  -- SYSLOGIC file  --
PLX: OUT2 = ASM       -- Generated code --
PLX: OUT3 = LISTASMH  -- ASMH Listing   --
*-------------------------------------------------------------------
*
* List of things to be generated in link archdefs
*
* LEC: {archdef keyword} = {associated type} {associated comment}
*
LEC: LMAP = LINKMAP   -- Linkage editor listing --
LEC: OUT1 = SLDUSE    -- SLD/VSD USE file       --
*-------------------------------------------------------------------
* Information to be used to select from input according to given
* patterns subapplications and create archdefs for it.
*
*          Specifies subappls are to be created
* SUBAPPL: NAME = {name of subappl member} ! {commentary associated}
*
*          Specifies which modules go to a subappl
* SUBAPPL: SELECT= {name of subappl member} ! {condition} & ...
*                possible {conditions} are:
*                                LLIB   ={load lib type}
*                                MNAME  ={module name  }
*                                MPREFix={module prefix}
*
*          Contains all modules which do not fit in any SELECT
* SUBAPPL: MISC  = {name of suppapl member} ! {commentary associated}
*
```

Figure 27 (Part 2 of 3). LNK2ARCD: Example of <input_profile>

```
* A sample follows:
*
* SUBAPPL: NAME  = RMFLPAG3 ! Subappl LPAG3
* SUBAPPL: SELECT= RMFLPAG3 ! MPREFIX=LPAG3 & LLIB= PALIB
*
* SUBAPPL: NAME  = RMFLNK12 ! Subappl LNK12
* SUBAPPL: SELECT= RMFLNK12 ! LLIB= LINKLIB
*
* SUBAPPL: NAME  = RMFLNKR3 ! Subappl LNKR3
* SUBAPPL: SELECT= RMFLNKR3 ! MNAME  =RMFWDM
*
* SUBAPPL: MISC  = RMFMISC  ! Subappl miscellaneous
*
*--------------------------------------------------------------------
```

Figure 27 (Part 3 of 3). LNK2ARCD: Example of <input_profile>. Sample of information file and description of its usage

D.4 SDYNINCL - Dynamic Include Technique

Track dynamic includes based on the monitor files produced by PLX compiler and assembler exits. The usage in an SCLM translator step is shown in Figure 28 on page 136.

```
SDYNINCL  -H  hlq                          -G  SCLM_group  -T  SCLM_type
                      -P  SCLM_project

-M  SCLM_member  -I  dynamic_include_pointer  -IN  include_libs

                                                              (  /
  -DDASM  asminfo_ddname          -DDPLX  plxinfo_ddname
  -DDADATA  sysadata_ddname       -DDLGC  syslogic_ddname


  )
```

Parameters:

hlq High level qualifier of project

SCLM_project The alternate project or per default same as hlq

SCLM_group The SCLM group of the member to be processed

SCLM_type The SCLM type of the member to be processed

SCLM_member The name of the member to be processed

dynamic_include_pointer
The dynamic include pointer provided by SCLM to add the includes found

include_libs The list of macro libraries defined to FLMINCLS to be used during compile. Needed to verify if includes found are part of those libraries.

asminfo_ddname The provided assembler information through the assembler exit program tracing the includes used.

sysadata_ddname The SYSADATA file (currently not supported). If supported it will replace the DDASM as the preferred solution.

plxinfo_ddname The provided PLX information via a compile exit tracking the includes used.

syslogic_ddname The SYSLOGIC file (currently not supported). If supported it will replace the DDPLX as the preferred solution.

```
         MACRO
         B#DYNI    &DD_PLX=,                                          C
               &DD_ASM=,                                              C
               &DD_ADATA=,                                            C
               &DD_LGC=,                                              C
               &MAC_1=MAC,                                            C
               &MAC_2=,                                               C
               &MAC_3=,                                               C
               &MAC_4=,                                               C
               &MAC_5=
.*
```

Figure 28 (Part 1 of 6). SDYNINCL - Usage in a SCLM Translator Step Macro

```
 MNOTE *,¢&SYSLIB_MEMBER.: Parameters passed to macro:      ¢
 MNOTE *,¢&SYSLIB_MEMBER.:                                  ¢
 MNOTE *,¢&SYSLIB_MEMBER.:  DD_PLX        = &DD_PLX.        ¢
 MNOTE *,¢&SYSLIB_MEMBER.:  DD_ASM        = &DD_ASM.        ¢
 MNOTE *,¢&SYSLIB_MEMBER.:  DD_ADATA      = &DD_ADATA.      ¢
 MNOTE *,¢&SYSLIB_MEMBER.:  DD_LGC        = &DD_LGC.        ¢
 MNOTE *,¢&SYSLIB_MEMBER.:  MAC_1         = &MAC_1.         ¢
 MNOTE *,¢&SYSLIB_MEMBER.:  MAC_2         = &MAC_2.         ¢
 MNOTE *,¢&SYSLIB_MEMBER.:  MAC_3         = &MAC_3.         ¢
 MNOTE *,¢&SYSLIB_MEMBER.:  MAC_4         = &MAC_4.         ¢
 MNOTE *,¢&SYSLIB_MEMBER.:  MAC_5         = &MAC_5.         ¢
 MNOTE *,¢&SYSLIB_MEMBER.:                                  ¢
```

Figure 28 (Part 2 of 6). SDYNINCL - Usage in a SCLM Translator Step Macro

```
.*
.*
.*  MHELP 1+16    Used for debugging macro invocations if uncommented
.*                   It shows the macro invocation and the parameters
.*
.*
.*
 MNOTE *,¢&SYSLIB_MEMBER.: Append and pack files given by DDNAMEs    ¢
.*
*
```

Figure 28 (Part 3 of 6). SDYNINCL - Usage in a SCLM Translator Step Macro

```
          FLMTRNSL   CALLNAM=¢Dyn. Incl.   ¢,                          C
               FUNCTN=BUILD,                                           C
               CALLMETH=LINK,                                          C
               COMPILE=SCLMISP,                                        C
               OPTIONS=¢SDYNINCL -H @@FLMPRJ -P @@FLMALT -G @@FLMGRP   C
               -T @@FLMTYP -M @@FLMMBR -I @@FLMINC                     C
               -IN &MAC_1. &MAC_2. &MAC_3. &MAC_4. &MAC_5.             C
               -DDASM &DD_ASM. -DDADATA &DD_ADATA. -DDPLX &DD_PLX.     C
               -DDLGC &DD_LGC. ( /¢,                                   C
               PORDER=1
*
.*
```

Figure 28 (Part 4 of 6). SDYNINCL - Usage in a SCLM Translator Step Macro

```
.*MHELP 128     Used for debugging macro invocations if uncommented
.*                  It ends all macro traces
  MNOTE *,¢&SYSLIB_MEMBER.:                                  ¢
  MNOTE *,¢&SYSLIB_MEMBER.: End of Macro                     ¢
  MNOTE *,¢&SYSLIB_MEMBER.:                                  ¢
.*
          MEND
```

Figure 28 (Part 5 of 6). SDYNINCL - Usage in a SCLM Translator Step Macro

DD_PLX The ddname of PLX monitor output.
DD_ASM The ddname of ASM monitor output.
DD_ADATA The ddname of Logic ASM output.
DD_LGC The ddname of Syslogic output.
MAC_1 Type of macro library 1, checked for dynamic includes
MAC_2 Type of macro library 2, checked for dynamic includes
MAC_3 Type of macro library 3, checked for dynamic includes
MAC_4 Type of macro library 4, checked for dynamic includes
MAC_5 Type of macro library 5, checked for dynamic includes

Figure 28 (Part 6 of 6). SDYNINCL - Usage in a SCLM Translator Step Macro

D.5 REXX2LLD - Generate BookMaster Format from Sources

This program will strip the included design information out of a REXX type source file or Assembler source. If desired also the correlated source code can be shown as intercepted.

It is assumed, that the stripped file is then formatted with BookMaster and then printed, to get the best results.

You may try this exec on itself, because this documentation you read right now is part of the program and therefore created from it. The exec runs on both VM or MVS.

Invocation Examples

In the following examples VM is used as the environment. The input files are either:

```
SCLMINFO EXEC A          or
SCLMINFO CODEPRTX A
```

The created output file is SCLMINFO REXX2LLD A.

For the MVS environment the syntax is the same in spite of the file names used. The mapping is the following if BPRA is the high level qualifier and REXX the second qualifier. Both may differ but not the rest of the mapping.

```
SCLMINFO EXEC A        ¢BPRA.REXX.EXEC(SCLMINFO)¢
SCLMINFO CODEPRTX A    ¢BPRA.REXX.CODEPRTX(SCLMINFO)¢
SCLMINFO SCLMINFO A    ¢BPRA.SCLMINFO.SCLMINFO¢
```

Example 1

```
SCLMINFO SCLMINFO EXEC A
```

This will create the design document SCLMINFO SCLMINFO for the exec SCLMINFO EXEC A.

Example 2

```
SCLMINFO SCLMINFO EXEC A ( +CODE
```

This will create the design document SCLMINFO SCLMINFO. It includes the code fragments of the exec SCLMINFO EXEC A.

Example 3

```
CODEPRTX  SCLMINFO EXEC A
SCLMINFO SCLMINFO CODEPRTX A
```

The program CODEPRTX will create the document SCLMINFO CODEPRTX A.

SCLMINFO will then create the design document SCLMINFO SCLMINFO for the exec SCLMINFO EXEC A. It includes the code fragments of the exec SCLMINFO EXEC A in the CODEPRTX style together with the index created by CODEPRTX.

Design indicators in the source lines

REXX source code style

To indicate the design in the source code you must use the following design indicators which are extensions of the comment delimiters.

Table 7. Indicators for REXX and PL/xx like Source Code

Start indicator	End indicator	Type of design information within the indicators
/*$$	$$*/	Information which belongs into the users guide of the program or a program package.
/ * > >	< < * /	Information which belongs in the program level design documentation of the program or a program package.
/*	*/	Just the normal code commentary. These will not be stripped out as design information. They will always belong to the code part section.

Note: Up to now you must code each design delimiter on a separate line and not put any text behind or before a delimiter.

This rule is in effect, to make the original implementation straight forward.

In the future it may be relaxed at least that the start delimiter may contain text. The next step will be that a start delimiter and an end delimiter may be on the same line.

An example of how a code section would look including such a delimiter is:

Sample 1

```
/*REXX: */
/*$$
  How to use the program

  ⋮
  This procedure will

  ⋮
$$*/
<<*/
  main_body:
/*>>
  Initialize system dependent information which triggers system
  dependent processing.
<<*/
    parse source !.env . . . . .
    !.system.cms = †CMS†     /* The OS environment where the exec is used */

  ⋮
```

End of Sample 1

An example of how a code section would look using BookMaster tags including such a delimiter is:

Sample 2

```
/*REXX: */
/*$$
:h1.How to use the program
:p.This procedure will
:p.
  ⋮
$$*/
<<*/
  main_body:
/*>>
:p.Initialize system dependent information which triggers system
dependent processing.
<<*/
    parse source !.env . . . . .
    !.system.cms = †CMS†     /* The OS environment where the exec is used */

  ⋮
```

End of Sample 2

Assembler source code style

Table 8. Indicators for Assembler

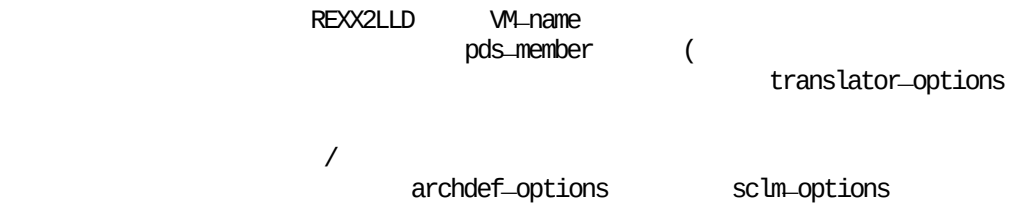
Start indicator	End indicator	Type of design information within the indicators
*$>	*$<	Information which belongs into the users guide of the program or a program package.
*>>	*<<	Information which belongs in the program level design documentation of the program or a program package.
*	*	Just the normal code commentary. These will not be stripped out as design information. They will always belong to the code part section.

Note: Up to now you must code each design delimiter on a separate line and not put any text behind or before a delimiter.

This rule is in effect, to make the original implementation straight forward.

In the future it may be relaxed at least that the start delimiter may contain text. The next step will be that a start delimiter and an end delimiter may be on the same line.

Invocation Syntax



VM_name:
fn ft fm

translator_options:
primary_options
primary_sclm_options
sclm_options

PDS_member
(Full_pds_qualifier(member)) with or without surrounding quotes

When running on MVS this syntax is checked and used. The last qualifier of the full_pds_qualifier corresponds to the VM file type. The member name corresponds to the VM file name.

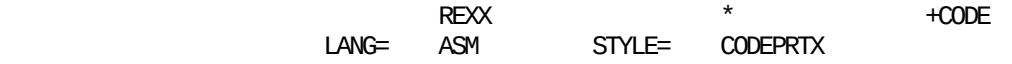
VM_name
When running on VM this syntax is checked and used.

fn (String). The file name.
fm (String). The file mode.
ft (String). The file type.

options
(String). If the no options are found then the complete design information is extracted without the code.

RETURN
(Integer). The return code.

primary_options:



archdef_options:
primary_options

The valid common options are:

+CODE
(String). The option to trigger the code fragments should be included in the design document. This option will be set automatically if the file type is CODEPRTX.

LANG
The language to extract information. Section markers depend on it.

REXX REXX-like languages. Is the default

ASM Assembler-like languages

STYLE

Kind of format to scan

***** Just take what is defined in LANG keyword. Is the default

CODEPRTX The CODEPRTX format

design_types:

```
->UG
->DES
```

design_types

(Keyword). Determines down to which level, from the top, will be extracted. The level hierarchy will be *UG-DES*.

The last level is the CODE level, which is handled by a separate option

->UG User's Guide information. Just the top level. Information that is relevant for the user's guide of the program documentation.

->DES Program Level Design. From the top level to the Design. Information that shows the program level design information.

SCLM primary options

primary_sclm_options:

```
P= sclm_project G= sclm_group T= sclm_type M= sclm_member
```

sclm_project

The SCLM project

sclm_group

The SCLM group

sclm_type

The SCLM type

sclm_member

The SCLM member name

sclm_options:

```
+SCLM
      DDSCLM( sclminfo_ddname )

+JCL
+ALLOC    DDJCL( jcl_ddname )

+ASM< assembly_options >      file_options
```

The valid SCLM related options are:

+SCLM

(String). The option to trigger that a file contains SCLM information and it should be extracted and added to the end of the document.

sclminfo_ddname
(String). Will use the given name as ddname for the temporary SCLMINFO file. The allocation for the file must be done before this exec is called. Otherwise a default name userid.REXX2LLD.SCLMINFO with a DDNAME SCLMINFO will be used for the temporary allocation.

+JCL
(String). Says that JCL for allocations of PDS should be created.

jcl_ddname
The JCL should be written under this DDNAME.

assembly_options
(String). Indicates that for this invocation the assembler should be called for this member. The assembler related allocation must be done before invocation of this exec. The options listed here are passed without verification to the assembler.

file_options:

```
DDIN( input_ddname )    DDOUT( output_ddname )
                        OUTPDS( pds_name )
```

The valid file options are:

OUTPDS
(String) (TSO only) The option tells where to store the output from the exec. The given PDS name must be a partition organized file. If this option is not specified, the output on MVS is directed to a sequential file with the name 'userid.member_name.REXX2LLD'.

input_ddname
(String) (CMS and TSO) The option specifies the input DDNAME to be used. The pds_name given if any is ignored in this case. The input member must be already allocated to the DDNAME specified.

output_ddname
(String) (CMS and TSO) The option specifies the output DDNAME to be used. The output member must be already allocated to the DDNAME specified.

D.6 SCLMINFO - Generate Data Set Allocation Information

This exec will scan the SCLM definitions (FLMxxx macro statements) and extract the information to:

- generate allocation definitions either in JCL JOB format or as a definition file to be used by the SALLOC exec.
- generate documentation parts to enhance the project documentation.
- generate RACF definitions to control the projects (currently not implemented).
- potential other information needed later related to the project definitions.

The exec runs on either VM or MVS.

Invocation Syntax

```
SCLMINFO      VM_name
              pds_member          (    primary_options
              OS/2_identifier

    /    archdef_options
```

VM_name:

```
fn  ft  fm
```

PDS_member

(Full_pds_qualifier(member)). When running on MVS this syntax is checked and used. The last qualifier of the full_pds_qualifier corresponds to the VM file type. The member name corresponds to the VM file name.

VM_name

When running on VM this syntax is checked an used.

fn (String). The file name.
fm (String). The file mode.
ft (String). The file type.

OS/2 identifier

The format drive path file_id, where drive and path may not be there.

primary_options:

```
JCL
ALLOC        RACF      BOOK      DDIN(  input_ddname  )
NOJCL
NOALLOC

DDLIB(  syslib_ddname  )        DDOUT(  output_ddname  )

                                DDJCL(  alloc_ddname  )

                                PS(  ps_name  )

                                +ASM<  assembly_parms  >
```

archdef_options:

```
JCL
ALLOC
NOJCL
NOALLOC
RACF
BOOK
```

options Options to trigger the parsing and output to create:

(NO)JCL Create the job to allocate data sets not yet defined.

(NO)ALLOC Create the data set allocation information to be done off-line. This overrules JCL if both are given.

	RACF	Create the RACF job to define the authorization for the groups.
	BOOK	Create the BookMaster tags to document the information in a more readable way.
	assembly_parms	Triggers that the member should be processed by the assembler if it contains the FLMABEG macro. The assembly_parms are the options to be used for the assembler step.
options	File options	
input_ddname	Use a ddname for input instead of input file name	
syslib_ddname	Use a ddname for copy or include libraries	
output_ddname	Use a ddname for output instead of PS name or default	
alloc_ddname	Use a ddname for allocation information created triggered by options JCL or ALLOC	
ps_name	Use the specified name for the output file	

Valid statements in project definition source to be parsed by SCLMINFO:

```
*ALLOCDEF   INIT:
            VSAM:   VSAM_specs
            TYPE:   TYPE_specs
```

INIT:	Can be used to reset default values
VSAM:	Must be used with the FLMCNTRL and FLMALTC macros
TYPE:	Must be used with the FLMTYPE macros

VSAM_specs:

```
    ACCTCYL   primary_alloc  secondary_alloc
    VERSCYL
    EXPOCYL
```

-ACCTCYL	Defines the number of primary_alloc and secondary_alloc in cylinder units for the accounting data sets.
-VERSCYL	Defines the number of primary_alloc and secondary_alloc in cylinder units for the auditing data sets.
-EXPOCYL	Defines the number of primary_alloc and secondary_alloc in cylinder units for the export data sets.

TYPE_specs:

```
    -NOSTORCLAS
    -STORCLAS  storage_class    -DCB  recfm  lrecl
                                                 blksize
                                                       -PDSE
                                                       -PDS
  -MAXALLOC  max_size    TRK    -DIRBLKS  directory_blocks
                         CYL
```

-STORCLAS, -NOSTORCLAS
Defines the storage_class to be used

-DCB Defines the record format recfm, the logical record length lrecl and optionally, the block size to be used.

-MAXALLOC Defines the maximum size (max_size) of the data set in tracks or cylinders.

-DIRBLKS Defines the directory blocks to be used in case it is a PDS type data set.

-PDS, PDSE Defines if the data set should be allocated as a PDS or a PDSE.

Data Set Allocation Definition produced by SCLMINFO

```
Option to create JCL was switched off
START: -HLQ SCLM2
VSAM: -VERS -Name SCLM1.PROJDEF$.AUDIT  -Cyl 4 1
VSAM: -ACCT -Name SCLM1.PROJDEF$.ACCOUNT1.COM  -Cyl 4 1
VSAM: -ACCT -Name SCLM1.PROJDEF$.ACCOUNT2.COM  -Cyl 4 1
VSAM: -VERS -Name SCLM1.PROJDEF$.AUDIT.COM  -Cyl 4 1
VSAM: -ACCT -Name SCLM2.PROJDEF$.ACCOUNT1.VSE  -Cyl 4 1
VSAM: -ACCT -Name SCLM2.PROJDEF$.ACCOUNT2.VSE  -Cyl 4 1
VSAM: -VERS -Name SCLM2.PROJDEF$.AUDIT.VSE  -Cyl 4 1
VERPDS: -CNTL @@FLMDSN.VERSION
DSNAME: -ALTC #COM      SCLM1.@@FLMGRP.@@FLMTYP
VERPDS: -ALTC #COM      @@FLMDSN.VERSION
VERPDS: -ALTC #VSE      @@FLMDSN.VERSION
Groups: -Names  -ALTC #COM( COMMON) -ALTC #VSE( VSE$ VSE)
Type: -TYPE SCLMDEFP -DCB FB 80 3120 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE SCLMDEFL -DCB FB 80 3120 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE SCLMDEFC -DCB FB 80 3120 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE SCLMDEFV -DCB FB 80 3120 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE SCLMDEFS -DCB FB 80 3120 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE SCLMDEFD -DCB FB 80 3120 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE SCLMVIEW -DCB FB 80 23200 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE SCLMLANG -DCB FB 80 23200 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE SCLMMACS -DCB FB 80 23200 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE SCLMJCL -DCB FB 80 23200 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE SCLMDOCS -DCB VB 255 27998 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE SCLMGMLI -DCB VB 255 27998 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE SCLMGMLO -DCB VB 255 27998 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE SCLMOBJ -DCB FB 80 3120 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE SCLMLIST -DCB FB 121 6050 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE SCLMLOAD -DCB U 0 23200 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE SCLMLMAP -DCB VBA 137 3120 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE VSE -DCB FB 80 23200 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE REXXDEFP -DCB FB 80 23200 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE REXXDEFC -DCB FB 80 23200 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE REXXDEFD -DCB FB 80 23200 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE REXXMOD -DCB FB 80 23200 -PDSTYPE PDSE -MAXSIZE 10 -UNITS  TRK
Type: -TYPE REXXMAC -DCB FB 80 23200 -PDSTYPE PDSE -MAXSIZE 10 -UNITS  TRK
Type: -TYPE REXX -DCB FB 80 23200 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE REXXDOCS -DCB FB 80 23200 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE REXXGMLI -DCB FB 80 23200 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE REXXGMLO -DCB FB 80 23200 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE REXXBOOK -DCB FB 80 23200 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE REXX3820 -DCB FB 80 23200 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE BMDEF -DCB FB 80 3120 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
```

```
Type: -TYPE BMSCRIPT -DCB VB 255 27998 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE BMIMBEDS -DCB VB 255 27998 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE BMMACROS -DCB VB 255 27998 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE BMSTYLES -DCB VB 255 27998 -PDSTYPE PDSE -MAXSIZE 10 -UNITS TRK
Type: -TYPE BMBOOK -DCB FB 4096 28672 -PDSTYPE PDSE -MAXSIZE 10 -UNITS  TRK
Type: -TYPE BM3820 -DCB VBM 8205 8209 -PDSTYPE PDSE -MAXSIZE 10 -UNITS  TRK
Version: -COND *       *        NO
Version: -COND VSE$    SCLMDEFP YES
Version: -COND VSE$    SCLMDEFC YES
Version: -COND VSE$    SCLMDEFV YES
Version: -COND VSE$    SCLMDEFD YES
Version: -COND VSE$    VSE      YES
Version: -COND VSE$    SCLMVIEW YES
Version: -COND VSE$    SCLMLANG YES
Version: -COND VSE$    SCLMMACS YES
END:
```

D.7 SALLOC - Allocate Data Sets

This program will do all the allocation for an SCLM controlled project. The information is provided by an allocation information member, which will be normally generated by the exec SCLMINFO during a compile of an SCLM project. For more details see also the SCLMINFO exec.

You call the exec with:

```
SALLOC  data set_member
                          -QUERY                ON
                                    -BLKSIZE    OFF
```

-QUERY If specified then only a query and report is done. No allocation actions are done.

-BLKSIZE If set to off then no blocksize allocation will be done even if specified later on.

The program will first check if the data set to be allocated already exists. If so, no allocation will be performed, otherwise the data set will be allocated.

Statements of allocation definition file:

```
//NL//
 *  commentary
                          //NL//
   START_specs    //NL//     *  commentary          END_specs
                             VSAM_specs
                             VERPDS_specs
                             DSNAME_specs
                             GROUPS_specs
                             TYPE_specs
                             VERSION_specs
```

//NL// Start a new line

commentary A line with comments starting with an asterisk

START_specs Indicates that now the allocation definitions start. For options see the following syntax diagrams.

VSAM_specs VSAM data set allocation specification. For options see the following syntax diagrams.

VERPDS_specs Versioning PDS allocation specification. For options see the following syntax diagrams.

DSNAME_specs data set mapping specification. For options see the following syntax diagrams.

GROUPS_specs group related allocation specification. For options see the following syntax diagrams.

TYPE_specs type related allocation specification. For options see the following syntax diagrams.

VERSION_specs
Selection of version data sets to be allocated. For options see the following syntax diagrams.

END_specs End of allocation specification. For options see the following syntax diagrams.

END of allocation definitions
This statement ends the allocation definitions. Even though it is possible to have several START, END sequences in one member, currently only the first is used for processing.

END_specs:

```
END:
```

START of allocation definitions
This statement starts the allocation definitions. It is possible to specify parameters to either set defaults or influence the processing of the allocation definitions.

START_specs:

```
START:
          -HLQ  high_level_qualifier      -REORG    GROUPS

              OFF                  ON
   -QUERY     ON         -ALLOC    OFF
```

high_level_qualifier
Use this as the default high-level qualifier if not otherwise defined.

GROUPS If specified then it indicates that the GROUP REORG parameter is active. If not specified no reorg will be done.

query_switch If set to on then only a query would be done regarding what action is to be taken. No alloc and no reorg are done in this case.

alloc_switch If set to off then no allocation is done.

Allocate SCLM control data sets
This statement allows to specify which SCLM control data sets should be allocated. ACCT stands for accounting, EXPO for export, VERS for auditing and XREF for the cross reference.

VSAM_specs:

```
VSAM:    -ACCT    -NAME  data set_name  -CYL  primary_units
         -EXPO
         -VERS
         -XREF

   secondary_units
```

data set_name
Name of the VSAM data set to be allocated

primary_units Units to use for primary allocation

secondary_units
Units to use for secondary allocation

GROUP allocation definitions statement
This statement allows to specify that SCLM group related information is to be used to create data set names.

GROUPS_specs:

```
GROUPS:

          -NAMES                        -REORG
                  group_name                    group_name


     -ALTC  altc_name  (    group_name    )
```

-NAMES List of group names not defined via an ALTC option

-ALTC List of group names defined via an altc_name

-REORG If a reorganization is active then the list of defined groups will be reorganized.

DSNAME allocation definitions statement
This statement allows to specify SCLM data set patterns to be used to create the data set names. Those patterns are defined in the FLMCNTRL and FLMALTC macros if the statements are generated via the SCLMINFO exec.

DSNAME_specs:

```
DSNAME:  -CNTL  data set_specs
         -ALTC  altc_name  data set_specs
```

-CNTL Specifies the data set_specs to be used for the groups with no altc_names.

-ALTC Specifies the data set_specs to be used for the groups with the altc_name reference.

VERSION allocation definitions statement
This statement allows to specify SCLM group, type selection for versioning. The pattern for versioning is taken from the FLMCNTL or FLMALTC macro if the SCLMINFO is used to generate the statements.

VERSION_specs:

```
VERSION:  -COND    *              *              NO
                   group_name     type_name      YES
```

group_name Name of an SCLM group, or any group if * is specified

type_name Name of an SCLM type, or any type if * is specified

NO, YES Indicate if the group, type combination is to be version or not. If so then the version data set will be allocated for this combination.

VERPDS allocation definitions statement

This statement allows to specify SCLM version data set patterns to be used. The patterns are taken by FLMCNTRL and FLMALTC macros if statements are generated by the SCLMINFO exec.

VERPDS_specs:

```
VERPDS:   -CNTL              data set_name_pattern
          -ALTC  altc_name
```

altc_name Name of an ALTC option related to a group

data set_name_pattern

Name of a data set pattern to be used to create the allocation data set names.

TYPE allocation definitions statement

This statement allows to specify SCLM type related allocation information.

TYPE_specs:

```
TYPE:                                  -TYPE  type_name
        -HLQ  Highlevel_qualifier      -QUAL  qualifiers

-DCB  recfm  lrecl                -MAXSIZE  maximum_allocation
                    blocksize

         TRK
-UNITS   CYL
                  -STORCLAS  storage_class

  -LIKE  data set_name

           PDSE
-PDSTYPE   PDS
                     -DIR  directory_blocks
```

-TYPE Type_name to be used

-HLQ highlevel_qualifier to be used

-QUAL A given qualifier to be used to create data set name

-DCB Allocation parameters to do the allocation. Record format (recfm) and logical record length (lrecl) are required, blocksize is optional.

-MAXSIZE The maximum size in allocation units to be used to compute the number of primary and secondary extents so that the full extent would cover a data set.

-UNIT The units to be used to allocate the data set, either tracks or cylinders.

-LIKE Use a data set_name as model for allocation

-STORCLAS The storage class to be used for allocation to force the allocation to certain devices.

-PDSTYPE Define if PDS or PDSE type data sets should be used. For PDS the -DIR parameters is required.

-DIR Directory blocks to be allocated for PDS data sets.

D.8 Allocation Definitions for Data Set Reorganization

A sample of the generated data set allocation definition is shown in D.6, “SCLMINFO - Generate Data Set Allocation Information” on page 143.

```
* Use this to reorg files under hlq HCD.
START:-HLQ HCD -REORG GROUP
GROUPS: -NAMES PLAY PLAY1 PLAY1T PLAY100
      : -REORG                   PLAY100
TYPE: -TYPE ACBDCLST
TYPE: -TYPE ACBDHENU
TYPE: -TYPE ACBDJCL1
TYPE: -TYPE ACBDMENU
TYPE: -TYPE ACBDMOD1
TYPE: -TYPE ACBDMOD2
TYPE: -TYPE ACBDPENU
TYPE: -TYPE ACBDTENU
TYPE: -TYPE AMODGEN
TYPE: -TYPE APROCLIB
TYPE: -TYPE LINKDEF
TYPE: -TYPE LINKLIB
TYPE: -TYPE LINKMAP
TYPE: -TYPE LISTASM
TYPE: -TYPE MAC
TYPE: -TYPE MISCDEF
TYPE: -TYPE MOD
TYPE: -TYPE MSGS
TYPE: -TYPE MSGSDEF
TYPE: -TYPE MSGSMAC
* Make sure that the next are reorganized as PDS and not as PDSE
TYPE: -TYPE MSGSMOD    -PDSTYPE PDS
TYPE: -TYPE PREP32I    -PDSTYPE PDS
TYPE: -TYPE PREP32L    -PDSTYPE PDS
END:
```

D.9 SCLMSITE - Basic SCLM Project Environment Setup

This exec will do allocations for SCLM projects, with a basic support.

If a project needs specific allocations, the user must issue these himself and in addition this may be called by a specific allocation procedure for such an environment.

In the latter case, the procedures must ensure that this procedure will not be called several times from several places.

The invocation is:

SCLMSITE PROJECTS(proj1,proj2,...)

where projn is a valid project defined to the exec.

Actions performed:

1. Check active ISPF-Release and do allocations and actions to prepare the common SCLM environment for all projects.
2. Depending on user IDs do allocations for testing and debugging.
3. Depending on the specified projects calling this exec do the allocations and all the project related actions.
4. Take special action when this is running in the background.

Project specific actions:

1. Check if an SCLM project for the active ISPF/SCLM release exists for the project.
2. Do project specific allocations and actions.
3. Allocate project.PROJDEFS.REXX

D.10 DTLC - ISPF DTL Dialog Development

This exec is used to process DTL source and create ISPF output. It was primarily written to be used as an SCLM BUILD translator step. However it can also be used in TSO foreground and TSO batch. For its use refer to Figure 29 on page 155, Figure 30 on page 156 and Figure 31 on page 157.

This translator is able to process all DTL tags and create the appropriate ISPF outputs, such as ISPF panels, ISPF messages and ISPF tables.

```
DTLC    input_member_spec      ddname_mapping
 processing_exits    (                     /
                          standard_parms        ,

 archdef_parms      EXITS<    processing_exits    >

 archdef_parms      )
```

standard_parms
ISPDTLC parameters for all DTL members. This is the place to put all common parameters for all possible invocation types of DTLC.

archdef_parms
ISPDTLC parameters for just one member specified in its archdef member. This is the place which is used during an SCLM process to get member dependent parameters.

```
input_member_spec:
   P=project  G=group  T=type  M=member_name
```

project Name of project currently active

group Name of group currently active

type Name of type currently active

member Name of the DTL member to be processed.

```
ddname_mapping:
   DDPROF=profile_ddname  DDDTL=dtl_ddname  DDINC=include_ddname
   DDPAN=panel_members_ddname  DDMSG=message_members_ddname
   DDTAB=table_members_ddname  DDLOG=log_ddname
```

profile_ddname
DDname to create the Profile data set which is needed for ISPDTLC to know the ddname filename mapping. Must be preallocated and sequential.

dtl_ddname
DDname of the PDS to be used to find the DTL member to be processed. Must be preallocated and PO.

include_ddname
DDname of the PDS(s) to get the chain of DTL PDSs to resolve DTL entity processing. Must be preallocated and PO. If none is actually necessary, just use the same data set name as for the dtl_ddname.

panel_members_ddname
DDname of the PDS in which to put the produced ISPF panels. Must be preallocated and PO. It is assumed that before DTLC is called no members are in the PO.

message_members_ddname
DDname of the PDS in which to put the produced ISPF messages. Must be preallocated and PO. It is assumed that before DTLC is called no members are in the PO.

table_members_ddname
DDname of the PDS in which to put the produced ISPF tables. Must be preallocated and PO. It is assumed that before DTLC is called no members are in the PO.

log_ddname
DDname of sequential data set to write the log messages. Must be preallocated and sequential.

```
processing_exits:

     POSTPAN=panel_post_exit      ISPDTLC=explicit_name

     PREPROC=pre_exit      BACKPAN=panel_back_exit
```

pre_exit The fully qualified name of the preprocessing exec member to be called, which prepares the input member for DTL processing with the ISPF provided ISPDTLC exec.

panel_post_exit
The fully qualified name of the postprocessing exec member to be called.

panel_back_exit
The fully qualified name of the backend processing exec member to be called. This exec is called after the postprocessing exit if specified.

explicit_name The fully qualified name of the ISPDTLC exec to be called.

```
          FLMTRNSL   CALLNAM=¢DTL Source      ¢,                          *
               FUNCTN=BUILD,                                             *
               CALLMETH=LINK,                                            *
               COMPILE=SCLMISP,                                          *
               VERSION=V3R5,                                             *
               PORDER=1,                                                 *
               GOODRC=0,                                                 *
               OPTIONS=¢DTLC P=@@FLMPRJ,G=@@FLMGRP,T=@@FLMTYP,           *
               M=@@FLMMBR,   DDPRF=DTLPROF,DDGML=DTLGMLI,DDLOG=DTLLOGO,*
               DDINC=DTLINCI,DDPAN=DTLPANO,DDMSG=DTLMSGO,DDTAB=DTLTABO *
               (DISK NOPREP REPLACE CUAATTR NOPANEL MSGSUPP PORTSUPP/¢,*
               PARMKWD=PARM1
*
* 1       Input Data Set: DTL Source
          FLMALLOC  IOTYPE=S,DDNAME=DTLGMLI,                             *
               KEYREF=SINC,                                              *
               RECFM=VB,LRECL=255,RECNUM=5000
*
* 2       Include Data Sets (POs)
          FLMALLOC  IOTYPE=I,DDNAME=DTLINCI,                             *
               KEYREF=SINC
*
* 3       Output Data Set: ISPF Panel (PO)
          FLMALLOC  IOTYPE=P,DDNAME=DTLPANO,CATLG=Y,                     *
               KEYREF=OUT1,                                              *
               RECFM=FB,LRECL=80,BLKSIZE=23200,DIRBLKS=5,RECNUM=5000
*
* 4       Output Data Set: ISPF messages (PO)
          FLMALLOC  IOTYPE=P,DDNAME=DTLMSGO,CATLG=Y,                     *
               KEYREF=OUT2,                                              *
               RECFM=FB,LRECL=80,BLKSIZE=23200,DIRBLKS=5,RECNUM=5000
*
* 5       Output Data Set: ISPF tables (PO)
          FLMALLOC  IOTYPE=P,DDNAME=DTLTABO,CATLG=Y,                     *
               KEYREF=OUT4,                                              *
               RECFM=FB,LRECL=80,BLKSIZE=23200,DIRBLKS=5,RECNUM=5000
*
* 6       Error and message file (PS)
          FLMALLOC  IOTYPE=O,DDNAME=DTLLOGO,CATLG=Y,PRINT=Y,             *
               KEYREF=OUT3,                                              *
               RECFM=FB,LRECL=121,RECNUM=9000
*
* 7       Profile to provide I/O info to ISPDTLC (PS)
          FLMALLOC  IOTYPE=W,DDNAME=DTLPROF,CATLG=Y,                     *
               RECFM=FB,LRECL=80,RECNUM=900
*
```

Figure 29. Sample SCLM Definition

```
//BPRADTLC JOB (DE01111,,),BPRA,USER=BPRA,NOTIFY=BPRA,CLASS=S,         *
//         MSGCLASS=H,MSGLEVEL=(1,1)
//*MAIN SYSTEM=JGLOBAL
//*-------------------------------------------------------------------*
//BUILD EXEC PGM=IKJEFT01,REGION=4096K,TIME=1439,DYNAMNBR=200
//*******************************************************************
//* ISPF/PDF LIBRARIES TO ACCESS
//*******************************************************************
//*
//ISPMLIB DD DSN=SYS1.ISRMENU,DISP=SHR  ISPF/PDF MSGS
//         DD DSN=SYS1.ISPMENU,DISP=SHR  ISPF MESSAGES
//*
//ISPSLIB DD DSN=SYS1.ISRSENU,DISP=SHR  PDF SKELS
//         DD DSN=SYS1.ISPSLIB,DISP=SHR  ISPF SKELS
//*
//ISPPLIB DD DSN=SYS1.ISRPENU,DISP=SHR  PDF PANELS
//         DD DSN=SYS1.ISPPENU,DISP=SHR  ISPF PANELS
//*
//ISPTLIB DD DSN=SYS1.ISRTLIB,DISP=SHR  PDF TABLES
//         DD DSN=SYS1.ISPTENU,DISP=SHR  ISPF TABLES
//*
//*******************************************************************
//* ISPF/PDF NEEDED FILES TO RUN
//*******************************************************************
//*
//ISPTABL DD UNIT=SYSALLDA,DISP=(NEW,PASS),SPACE=(CYL,(1,1,5)),
//         DCB=(LRECL=80,BLKSIZE=19040,DSORG=PO,RECFM=FB),
//         DSN=                    TEMPORARY TABLE LIBRARY
//*
//ISPPROF DD UNIT=SYSALLDA,DISP=(NEW,PASS),SPACE=(CYL,(1,1,5)),
//         DCB=(LRECL=80,BLKSIZE=19040,DSORG=PO,RECFM=FB),
//         DSN=                    TEMPORARY TABLE LIBRARY
//*
//ISPLOG  DD SYSOUT=*,
//         DCB=(LRECL=120,BLKSIZE=2400,DSORG=PS,RECFM=FB)
//*
//*******************************************************************
//* SOME OTHER FILES NEEDED
//*******************************************************************
//SYSPRINT DD SYSOUT=(*)
/*
//*-------------------------------------------------------------------
//* TSO OUTPUT FILE
//*-------------------------------------------------------------------
//SYSTSPRT DD SYSOUT=(*)
/*
//*******************************************************************
//* REXX LIBRARIES WHERE NEEDED EXEC RESIDES
//*******************************************************************
//SYSEXEC DD DSN=TRYSCLM.BPRA.REXX,DISP=SHR The translator DTLC
//     DD DSN=SYS1.ISPEXEC,DISP=SHR        The ISPDTLC exec
//*
```

Figure 30 (Part 1 of 2). Sample Batch Invocation

```
//*********************************************************************
//* THE FILES NEEDED FOR DTL PROCESSING
//*********************************************************************
//* Inputs
//DTLGMLI DD DSN=BPRA.PLAY1.DTL,DISP=SHR
//DTLINCI DD DSN=BPRA.PLAY1.DTL,DISP=SHR
//* Outputs
//DTLPANO DD DSN=BPRA.PLAY1.HELPPAN,DISP=SHR
//DTLMSGO DD DSN=BPRA.PLAY1.SRMFMENU,DISP=SHR
//DTLTABO DD DSN=BPRA.PLAY1.SRMFTENU,DISP=SHR
//DTLLOGO DD DSN=BPRA.PLAY1.DTLLOG,DISP=SHR
//* Workfiles
//DTLPROF DD UNIT=SYSALLDA,DISP=(NEW,PASS),SPACE=(CYL,(1,1,0)),
//        DCB=(LRECL=80,BLKSIZE=19040,DSORG=PS,RECFM=FB),
//        DSN=BPRA.PLAY1.PROFILE      TEMPORARY TABLE LIBRARY
//*
//*--------------------------------------------------------------------
//* TSO INPUT FILE
//* THE INVOCATION OF THE DTL PROCESS
//*--------------------------------------------------------------------
//SYSTSIN DD *
  LAF
  ISPSTART CMD(DTLC  P=BPRA    ,G=PLAY1   ,T=DTL     ,M=ERBKEYS,       -
         DDGML=DTLGMLI,DDINC=DTLINCI,DDLOG=DTLLOGO,DDPRF=DTLPROF,     -
         DDPAN=DTLPANO,DDMSG=DTLMSGO,DDTAB=DTLTABO,                   -
         POSTPAN=PRODUCT.TOOLS.REXX(POSTPAN),
         ISPDTLC=ISPF.V41.REXX(ISPDTLC)
  (DISK NOPREP REPLACE CUAATTR NOPANEL MSGSUPP PORTSUPP KEYLAPPL=ERB/  -
              )
/*
```

Figure 30 (Part 2 of 2). Sample Batch Invocation. If called in batch all files must be preallocated and the output files should be empty. If not then postprocessing will also work on already existing members and messages from exec are not correct or files which are already postprocessed will be processed again which may lead to wrong formats.

```
⋮
sclm_info  =†P=X G=Y T=Z M=ABC†
ddin_info  =†DDPROF=DTLPROF DDDTL=DTLGMLI DDINC=DTLINCI†
ddout_info =†DDPAN=DTLPANO  DDMSG=DTLMSGO DDTAB=DTLTABO DDLOG=DTLLOGO†
parms      =†DISK NOPREP REPLACE CUAATTR NOPANEL NOMSGSUPP KEYLAPPL=YYY†

call DTLC sclm_info ddin_info ddout_info †(† ispdtlc_parms †/†
⋮
```

Figure 31. Example how to Invoke DTLC in TSO Foreground (from an EXEC). To process the DTL source 'X.Y.Z(ABC)' with preallocated data sets you may use the shown code fragments.

D.11 Panel Postprocessing Exit

In addition after calling the ISPDTLC exec of ISPF to create ISPF output from DTL, it calls a panel postprocessing exit if at least one panel is found in the panel output data set.

The exit can be activated explicitly using the keyword POSTPAN described in the invocation syntax.

For more details about the exit refer to Figure 32.

The exec is prepared to process one DTL member and postprocess created outputs. SCLM 3.5 does not support moving more than one output to SCLM controlled PDS after successful processing. This is possible with ISPF 4.1.

If the exit is called then its syntax for invocation will be:

```
EXEC ¢exit_name¢ ¢member data set_name                ¢ EXEC
                                                  dtl_member
```

exit_name	Fully qualified name of the exec without quotes
member	Member in data set_name to be postprocessed.
dtl_member	Member name of DTL source from where member was generated.
data set_name	Fully qualified name of the PDS containing the members to be postprocessed, however without quotes.

Figure 32. Post Processing Exit

D.12 Panel Backend Processing Exit

In addition after calling the ISPDTLC exec of ISPF to create ISPF output from DTL, it calls a panel backend processing exit if at least one panel is found in the panel output data set to do project specific treatment of the panel output coming from ISPDTLC or the postprocessing exit if specified.

To activate the exit it must be explicitly specified using the keyword BACKPAN described in the invocation syntax. The exec takes one member and replaces it by the processed output.

If the exit is called then its syntax for invocation will be:

```
EXEC ¢exit_name¢ ¢ P= project , G= group , T= type ,
DA= data set_name , M= member_name
                                    , WI= input_work_type
                                                    ¢ EXEC
  , WO= output_work_type     , WO= log_work_type
```

exit_name Fully qualified name of the exec without quotes or %exec_name to search for an exec in the search chain

project The SCLM project name of source member to be processed

group The SCLM group name of source member to be processed

type The SCLM type name of source member to be processed

data set_name Fully qualified name of the PDS containing the member to be postprocessed, however without quotes.

member Member in data set_name to be postprocessed.

input_work_type
The type to copy the member to be processed from data set_name.

output_work_type
The type to store the member after processed From here it is copied back to data set_name.

log_work_type The log output created in the backend process for this member.

Figure 33. Backend Processing Exit

Appendix E. VSAM Control File Utility Jobs

This chapter lists some useful examples to analyze and/or reorganize VSAM control files for optimization or problems.

```
//USR1VSAM JOB (DE11111,,),USER1,NOTIFY=USR1,CLASS=A,USER=USR1,
//             REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* USR2VSAM JOB (DE22222,,),USER2,NOTIFY=USR2,CLASS=A,USER=USR2,
//*              REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* USR1VSAM JOB (DE11111,,),USER1,NOTIFY=USR1,CLASS=A,USER=USR1,
//*              REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* ================================================================
//* MEMBER: VSE.PROJDEFS.VSAMCNTL(LISTCAT)
//* ================================================================
//*
//*  Use this job to get the account file information about its size
//*  and general status.
//*
//* ================================================================
//*
//LISTC   EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=H
//SYSIN     DD *
  LISTC ENT(¢VSE.PROJDEF$.ACCOUNT1.BAS¢) ALL
  LISTC ENT(¢VSE.PROJDEF$.ACCOUNT2.BAS¢) ALL
  LISTC ENT(¢VSE.PROJDEF$.ACCOUNT1.INT¢) ALL
  LISTC ENT(¢VSE.PROJDEF$.ACCOUNT2.INT¢) ALL
/*
//
```

Figure 34. VSAM LISTCAT - Show VSAM Statistics

```
//USR1VSAM JOB (DE11111,,),USER1,NOTIFY=USR1,CLASS=A,USER=USR1,
//             REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* USR2VSAM JOB (DE22222,,),USER2,NOTIFY=USR2,CLASS=A,USER=USR2,
//*              REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* USR1VSAM JOB (DE11111,,),USER1,NOTIFY=USR1,CLASS=A,USER=USR1,
//*              REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* ================================================================
//* MEMBER: VSE.PROJDEFS.VSAMCNTL(LISTVTOC)
//* ================================================================
//*
//*  Use this job to get the VTOC information of the account files
//*
//*    Check the volume name is still correct before submitting
//*    the job
//*
//* ================================================================
//*
//STEP1    EXEC PGM=IEHLIST
//SYSPRINT DD SYSOUT=*
//DD2      DD UNIT=DASD,VOL=SER=SMS002,DISP=SHR
//SYSIN    DD *
   LISTVTOC FORMAT,VOL=DASD=SMS002,                                  *
              DSNAME=(VSE.PROJDEF$.ACCOUNT1.BAS)
/*
```

Figure 35. VSAM List VTOC - Show VTOC Information for VSAM Data Sets

```
//USR1VSAM JOB (DE11111,,),USER1,NOTIFY=USR1,CLASS=A,USER=USR1,
//             REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* USR2VSAM JOB (DE22222,,),USER2,NOTIFY=USR2,CLASS=A,USER=USR2,
//*              REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* USR1VSAM JOB (DE11111,,),USER1,NOTIFY=USR1,CLASS=A,USER=USR1,
//*              REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* ================================================================
//* Member: VSE.PROJDEFS.VSAMCNTL(BAS#1VER)
//* ================================================================
//*
//*  Use this job to check if the SCLM control files are still
//*  OK or in error. The job should return codes of 0, otherwise
//*  the corresponding account data set is incorrect.
//*
//*  To run this job the account files should not be in use!!!!!!
//*
//*  If one is incorrect then it must be restored with the correct one.
//*  If both account files are corrupted then contact the SCLM
//*  support group to discuss the appropriate actions to take.
//*
//*  To do the restore, edit job BAS#2SAV, follow its
//*  instruction and submit the job.
//* ================================================================
//*
//STEP1   EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=H
//SYSIN     DD *
     VERIFY  DATASET(VSE.PROJDEF$.ACCOUNT1.BAS)
     EXAMINE NAME(VSE.PROJDEF$.ACCOUNT1.BAS) -
       INDEXTEST -
       DATATEST

     VERIFY  DATASET(VSE.PROJDEF$.ACCOUNT2.BAS)
     EXAMINE NAME(VSE.PROJDEF$.ACCOUNT2.BAS) -
       INDEXTEST -
       DATATEST

/*
```

Figure 36 (Part 1 of 2). VSAM - Check Data Sets for Correctness

```
//USR1VSAM JOB (DE11111,,),USER1,NOTIFY=USR1,CLASS=A,USER=USR1,
//             REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* USR2VSAM JOB (DE22222,,),USER2,NOTIFY=USR2,CLASS=A,USER=USR2,
//*              REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* USR1VSAM JOB (DE11111,,),USER1,NOTIFY=USR1,CLASS=A,USER=USR1,
//*              REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* ================================================================
//* Member: VSE.PROJDEFS.VSAMCNTL(BAS#3RE1)
//* ================================================================
//*
//* Use this job to save the current account files in a backup copy
//*
//*   - to do the restore later
//*   - to keep a backup for any case if needed
//*
//* Existing backup copies get deleted first in this job !!!!!
//*
//* After running this job successfully, you may proceed to recover
//* the corrupted account file by editing
//*
//*   BAS#3RE1 to recovery ACCOUNT1
//*   BAS#3RE2 to recovery ACCOUNT2
//*
//* Modify the job according to the instructions and submit it.
//*
//* ================================================================
//*
//STEP1   EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=H
//SYSIN     DD *
       DELETE ¢VSE.PROJDEF$.ACCOUNT1.BASOLD¢
       DELETE ¢VSE.PROJDEF$.ACCOUNT2.BASOLD¢

       DEFINE  CLUSTER ( NAME(VSE.PROJDEF$.ACCOUNT1.BASOLD)-
                         MODEL(VSE.PROJDEF$.ACCOUNT1.BAS)   -
                        )
       DEFINE  CLUSTER ( NAME(VSE.PROJDEF$.ACCOUNT2.BASOLD)-
                         MODEL(VSE.PROJDEF$.ACCOUNT2.BAS)   -
                        )
       REPRO INDATASET(VSE.PROJDEF$.ACCOUNT1.BAS) -
             OUTDATASET(VSE.PROJDEF$.ACCOUNT1.BASOLD)
       REPRO INDATASET(VSE.PROJDEF$.ACCOUNT2.BAS) -
             OUTDATASET(VSE.PROJDEF$.ACCOUNT2.BASOLD)
/*
```

Figure 36 (Part 2 of 2). VSAM - Check Data Sets for Correctness

```
//USR1VSAM JOB (DE11111,,),USER1,NOTIFY=USR1,CLASS=A,USER=USR1,
//             REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* USR2VSAM JOB (DE22222,,),USER2,NOTIFY=USR2,CLASS=A,USER=USR2,
//*              REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* USR1VSAM JOB (DE11111,,),USER1,NOTIFY=USR1,CLASS=A,USER=USR1,
//*              REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* ====================================================================
//* Member: VSE.PROJDEFS.VSAMCNTL(BAS#3RE2)
//* ====================================================================
//*
//*
//* Use this job to recover the corrupted ACCOUNT2 from ACCOUNT1!!!!
//*
//* Be sure you executed successfully the
//*           jobs BAS#1VER and BAS#2SAV beforehand!!!!!!
//*
//* ====================================================================
//*
//STEP1   EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=H
//SYSIN     DD *
     DELETE ¢VSE.PROJDEF$.ACCOUNT2.BAS¢

     DEFINE  CLUSTER ( NAME(VSE.PROJDEF$.ACCOUNT2.BAS)   -
                      MODEL(VSE.PROJDEF$.ACCOUNT1.BAS)   -

     REPRO INDATASET(VSE.PROJDEF$.ACCOUNT1.BAS) -
          OUTDATASET(VSE.PROJDEF$.ACCOUNT2.BAS)

/*
```

Figure 37. VSAM - Recover Backup Account File from Primary

```
//USR1VSAM JOB (DE11111,,),USER1,NOTIFY=USR1,CLASS=A,USER=USR1,
//             REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* USR2VSAM JOB (DE22222,,),USER2,NOTIFY=USR2,CLASS=A,USER=USR2,
//*              REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* USR1VSAM JOB (DE11111,,),USER1,NOTIFY=USR1,CLASS=A,USER=USR1,
//*              REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* ================================================================
//* MEMBER: VSE.PROJDEFS.VSAMCNTL(VSAM2SEQ)
//* ================================================================
//*
//*  Use this job to create a sequential file from a VSAM File
//*
//*    Check the volume name is still correct before submitting
//*    the job
//*
//* ================================================================
//*
//*   Job Procedure section used by EXEC statements at end of this
//*   file. Follow the instructions there.
//*
//SAVEVSAM PROC ALTC=,DATE=,HLQ=,NUM=
//SAVE     EXEC PGM=IDCAMS
//VSAM     DD   DISP=SHR,DSN=&HLQ..PROJDEF$.&NUM..&ALTC.
//SEQU     DD   DISP=(NEW,CATLG),UNIT=SYSDA,SPACE=(TRK,(300,300),RLSE),
//         DSN=&HLQ..PROJDEF$.ACCOUNT&NUM..&ALTC..SEQ&DATE,
//         DCB=(LRECL=32004,BLKSIZE=32008,RECFM=VB,DSORG=PS)
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   DISP=(OLD,PASS),DSN=*.INPUT.SYSUT2
//         PEND
//*
//INPUT    EXEC PGM=IEBGENER
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   DUMMY
//SYSUT2   DD   DISP=(NEW,PASS),SPACE=(TRK,(1,1)),UNIT=SYSDA,
//         DCB=(LRECL=80,RECFM=F,DSORG=PS)
//SYSUT1   DD   *
  REPRO OUTFILE(SEQU) INFILE(VSAM)
//*
//*  End of the procedure section
//*
//* ---------------------------------------------------------
//*   Update the DATE= parameter and remove comments
//*   before submitting the job.
//*
//* ---------------------------------------------------------
//*        EXEC SAVEVSAM,DATE=NOV02,HLQ=VSE,ALTC=BAS,NUM=1
//*        EXEC SAVEVSAM,DATE=NOV02,HLQ=VSE,ALTC=BAS,NUM=2
//*        EXEC SAVEVSAM,DATE=OCT17,HLQ=VSE,ALTC=INT,NUM=1
//*        EXEC SAVEVSAM,DATE=OCT17,HLQ=VSE,ALTC=INT,NUM=2
//
```

Figure 38. VSAM - Unload VSAM to Sequential Data Set

```
//USR1VSAM JOB (DE11111,,),USER1,NOTIFY=USR1,CLASS=A,USER=USR1,
//             REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* USR2VSAM JOB (DE22222,,),USER2,NOTIFY=USR2,CLASS=A,USER=USR2,
//*              REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* USR1VSAM JOB (DE11111,,),USER1,NOTIFY=USR1,CLASS=A,USER=USR1,
//*              REGION=4096K,MSGLEVEL=(1,1),MSGCLASS=H
//*
//* ====================================================================
//* MEMBER: VSE.PROJDEFS.VSAMCNTL(SEQ2VSAM)
//* ====================================================================
//*
//*  Use this job to restore a sequential file back to a VSAM File
//*
//*    Check the volume name is still correct before submitting
//*    the job and change the INPUT name and the data set name for
//*    output
//*
//* ====================================================================
//*
//*   Job Procedure section used by EXEC statements at end of this
//*   file. Follow the instructions there.
//*
//*
//SEQ2VSAM EXEC PGM=IDCAMS
//SYSPRINT DD   SYSOUT=*
//INPUT    DD   DISP=SHR,DSN=VSE.PROJDEF$.ACCOUNT1.BAS.SEQAPR25
//SYSIN    DD   *
  DELETE ¢VSE.PROJDEF$.ACCOUNT1.BAS¢ CLUSTER

  DEFINE CLUSTER                                    -
    (NAME(¢VSE.PROJDEF$.ACCOUNT1.BAS¢)              -
     CYLINDERS(300 20)                              -
     VOLUMES(SMS010)                                -
     KEYS(26 0)                                     -
     IMBED                                          -
     RECORDSIZE(264 32000)                          -
     SHAREOPTIONS(4,3)                              -
     SPEED                                          -
     UNIQUE                                         -
     SPANNED                                        -
    )                                               -
  INDEX                                             -
    (NAME(¢VSE.PROJDEF$.ACCOUNT1.BAS.INDEX¢)        -
    )                                               -
  DATA                                              -
    (NAME(¢VSE.PROJDEF$.ACCOUNT1.BAS.DATA¢)         -
     CISZ(2048)                                     -
     FREESPACE(50 50)                               -
    )

  REPRO INFILE(INPUT) OUTDATASET(¢VSE.PROJDEF$.ACCOUNT1.BAS¢)
```

Figure 39. VSAM - Load Sequential Data Set to VSAM Data Set

Appendix F. VSE Project View under HLQ SCLM2 in Source Edit Format

The following shows member **SCLM1.SCLM2.SCLM2(VSE)** in assembly format. All BookMaster tags are included as assembler comments for administration and documentation purposes.

```
*>>
*.*********************************************************************
*.se member=¢SCLM2(VSE)¢
*.*********************************************************************
*<<
*********************************************************************
*  Change activities:
*  950801 JP Created
*********************************************************************
*$>
*.*____________________________________________________________________
*:h2 id=pvse.VSE Project View under HLQ SCLM2
*$<
         TITLE   ¢VSE project view under HLQ SCLM2 on system BOESCLM¢
SCLM2    FLMABEG
*
*$>
*:p.This project definition describes project definition view under
*HLQ SCLM2 on the BOESCLM machine provided by MVS System Support.
*:p.The VSE project is used to
*:ul compact.
*:li.Create product administration projects for HLQ SCLM2.
*:eul.
*.*____________________________________________________________________
*:h2.Overall Project Definitions
*$<
*
      COPY C@
      COPY C@#COM
*$>
*.im C@
*.im C@#COM
*$<
*$>
*:h3.Control Data Set for VSE SCLM Project Development Hierarchy Section
*$<
*
#VSE     FLMALTC  ACCT=SCLM2.PROJDEF$.ACCOUNT1.VSE,                    *
               ACCT2=SCLM2.PROJDEF$.ACCOUNT2.VSE,                     *
               VERS=SCLM2.PROJDEF$.AUDIT.VSE,                         *
               VERPDS=@@FLMDSN.VERSION
*$>
*.*____________________________________________________________________
*:h2.VSE Hierarchy
*:fig.
*:figcap.VSE Hierarchy
```

```
*:xmp.
*   .-Common----&ellip.
*  /
*  | see :hdref refid=gvcom form=numonly.
*  \
*   ¢-----------&ellip.
*           &Au.
*          .-Development----&ellip.
*         /
*         | see :hdref refid=gvvse form=numonly.
*         \
*          ¢----------------&ellip.
*
*:exmp.
*:efig.
*$<
*JCLGEN XLOCS 1
        COPY G@#COM      The common hierarchy
*
*$>
*.im G@#COM
*$<
*$>
*:h3 id=gvvsed.Development Work Hierarchy Section for SCLM2 HLQ Views
*:fig.
*:figcap.Development Work hierarchy Section for SCLM2 HLQ Views
*:xmp.
* |  (...)          COMMON
* |                  &Au.
* ¢-------------------------
*                    &Lv.
* .-------------------------
* |                  &Lv.
* |  (VSE)          VSE$
* |                  &Au.
* |  (VSE)          VSE
* |
* ¢-------------------------
*
*:exmp.
*:efig.
*.*
*:p.This hierarchy section is used to develop SCLM product development
*definitions for the VSE project on HLQ SCLM2 for HLQ VSE on the
*BOESCLM system.
*.*
*:dl tsize=¢10m¢.
*:dt.VSE$
*:dd.The group contains parts that are in production for projects of
*HLQ VSE. Basically stored under VSE.PROJDEFS.*.
*.*
*:dt.VSE
*:dd.This group contains parts that are in update mode by the member
*of the VSE development or integration team who is responsible for the
*development of VSE SCLM release development environment(s).
*.*
*:edl.
*$<
*.*
```

```
VSE$     FLMGROUP ALTC=#VSE,AC=(VSE),KEY=Y,PROMOTE=COMMON
VSE      FLMGROUP ALTC=#VSE,AC=(VSE,TEMP),KEY=Y,PROMOTE=VSE$
*
*.*-------------------------------------------------------------------
*
     COPY T@      The global definition for all types
*$>
*.im T@
*$<
     COPY T@SCLM  The SCLM Project Development Types
*
*JCLGEN DCB   FB,80,23200
VSE      FLMTYPE   Unique type referred in docu as <hlq> type
*$>
*.im T@SCLM
*$<
     COPY T@REXX  The REXX Development Types
*$>
*.im T@REXX
*$<
     COPY T@BOOK  The Documentation Development Types
*$>
*.im T@BOOK
*$<
*$>
*.*-------------------------------------------------------------------
*:h3 id=vvvsev.Versioning of members
*:p.Keep versions of members in group VSE$ for the following
*input types:
*:p.
*.layout 4
*:ul compact.
*:li.SCLMDEFP
*:li.SCLMDEFC
*:li.SCLMDEFV
*:li.SCLMDEFD
*:li.VSE
*:li.SCLMVIEW
*:li.SCLMLANG
*:li.SCLMMACS
*:eul.
*.layout
*:p.
*$<
*
         FLMATVER VERSION=NO,GROUP=*,TYPE=*
*
         FLMATVER VERSION=YES,GROUP=VSE$,TYPE=SCLMDEFP
         FLMATVER VERSION=YES,GROUP=VSE$,TYPE=SCLMDEFC
         FLMATVER VERSION=YES,GROUP=VSE$,TYPE=SCLMDEFV
         FLMATVER VERSION=YES,GROUP=VSE$,TYPE=SCLMDEFD
*
         FLMATVER VERSION=YES,GROUP=VSE$,TYPE=VSE
         FLMATVER VERSION=YES,GROUP=VSE$,TYPE=SCLMVIEW
         FLMATVER VERSION=YES,GROUP=VSE$,TYPE=SCLMLANG
         FLMATVER VERSION=YES,GROUP=VSE$,TYPE=SCLMMACS
*
*$>
```

```
*$<
*
         COPY     L@                    All standard Languages
*
*>>
*.*_____________________________________________________________________
*<<
*
         FLMAEND
*>>
*.*********************************************************************
*.* End of  ¢SCLM2(VSE)¢
*.*********************************************************************
*<<
```

Appendix G. Sample Meta Project Documentation (Formatted Output)

This chapter shows how member **SCLM1.SCLM2.SCLM2(VSE)** has been documented as part of the project.

The following text only shows the documentation section of the project. The implementation sections are shown in Appendix H, “Sample Meta Project Implementation” on page 187. If you compare Appendix F, “VSE Project View under HLQ SCLM2 in Source Edit Format” on page 169 with this text and H.1, “Member VSE SCLM2” on page 187 you will see how the sections relate to each other and how it is possible to get from one source different views in different documentation.

D In order to differentiate between the user’s guide and the administration guide
D we added revision bars to the administration guide as shown in this paragraph.

Project environment definition for

```
System    MVS2
HLQ       SCLM2
Project   VSE

(Administrator Guide)
(VSE Developers Guide)
```

Creation Information

```
System .... MVS2
HLQ ....... SCLM1
Project ... SCLM2
Document  SCLM1.*.SCLMDOCS(#VSE), Printed 96-10-23 3:26 a.m.
```

This document describes the SCLM project definition environment for product **VSE**. It’s located on the HLQ **SCLM2** with the name **VSE**.

Under this SCLM project are all the definitions made for the **VSE** development environment under SCLM control.

The documentation for this final product development environment will be created under HLQ **SCLM2** SCLM project **VSE**.

This document serves two purposes:

- Gives advice on how to define and set up the product development environment controlled by SCLM
- Describes the project definitions of HLQ **SCLM2** project **VSE** to document the available group hierarchy, the types and the languages to define the product development project definitions.

G.1 Creation of a Product Project Definition

G.1.1 Setup to Access SCLM Project SCLM2 (VSE)

When you first start you will normally have requested a new project name under HLQ **SCLM2** from SCLM Support Group on System **MVS2**.

If not then do so first.

Otherwise, log on to your user ID on **MVS2** and make sure that you have the following line in your startup procedure for ISPF (for example, userid.MVS.CLIST(DEFAULT)):

```
EXEC ¢PC.$ASIS$.ALLOC(SCLMSITE)¢ ¢PROJECTS(SCLM2)¢ EXEC
```

This enables you to access all programs and resources to run the SCLM project **VSE** under HLQ **SCLM2**.

Now at this point you are in daily operation mode, as seen below.

G.1.1.1 Access to SCLM Project HLQ SCLM2 (VSE)

After you logged on you access your SCLM environment by option 10.

There, in the lower section of the panel, you enter:

```
Project     SCLM2
Alternate   VSE
Group       VSE
```

You normally will work with option 3.1, so enter this in the command line and press ENTER.

In this panel you should know the types available to this project definition. You will find those in this document.

You will also find in this document the group hierarchy and its use. So please refer to these sections for information.

In the type SCLMDOCS you should find at least the following members:

VSE# The master document for the administration

VSE$ The master document for the developers

VSE@ The master body for both the administration and developers guide.

If you start with the project definition you will find them as fill-in skeletons which will evolve later on to your product development project definition, where you find:

- the documentation of the project definition
- the implementation of the project definition
- and if you wish any agreements or process descriptions that apply to the project that are necessary to run the product development process and that can't be automated by the provided SCLM functions. These should be guidelines and rules for integrators, project leaders, developers or testers of the product to be developed.

If you have no experience in setting up such projects you are advised to consult SCLM Support Group to give you assistance doing the setup. We cannot cover all possible aspects of each product because of the uniqueness of each product. The products may be totally different or differ only in certain areas. It is important to have everyone concerned involved during the definition phase of the project. Using the experience of the support group will save you time and frustration in the long run, as a redefinition of a working system can be, at best, very time and resource consuming.

Now refer to the skeletons mentioned.

G.1.1.2 HLQ SCLM2....

In general member in the data sets are provided via an SCLM project.

The source of this document is stored in the meta project **SCLM2** on **MVS2**. The outputs are also generated there but also distributed to the **System MVS2**.

If not provided via the SCLM project, then the members are listed or it is explained explicitly how the members are updated.

Data Set* SCLM2*.PROJDEFS.LOAD: This will contain all the project views available under HLQ **SCLM2**.

Data Set* SCLM2*.PROJDEFS.LIST3820: This will contain the description in a LIST3820 format for printing of the project definitions stored in **SCLM2**.PROJDEFS.LOAD. #project members contain the administrator guide and $project members contain the users guide.

Data Set* SCLM2*.PROJDEFS.BOOK: This will contain the description in a BookManager format for online reading of the project definitions stored in **SCLM2**.PROJDEFS.LOAD. Same naming conventions as for LIST3820 members.

Data Set* SCLM2*.PROJDEFS.ALLOCDEF: Contains the allocation jobs or definitions to allocate SCLM controlled data sets and those needed to have a complete development environment. Most of the members will be generated.

Data Set* SCLM2*.PROJDEFS.RACFDEF: Updated manually. Contains the RACF definitions to control the access to SCLM controlled data sets and those needed to have a complete development environment.

Data Set* SCLM2*.PROJDEFS.REXX: Updated from the **SCLM2** project itself. Contains **SCLM2** specific REXX programs.

Data Set* SCLM2*.PROJDEFS.SHIPLIST: Updated manually. Contains the shipping list information used by the SHIPIT exec in PC.SCLMSITE.REXX to transfer members during promote from the **SCLM2** controlled library to a different place.

At least the following member must exist:

SCLM2 The root for the shipping information scanned by the REXX exec SHIPIT. It may contain include statements.

Data Set* SCLM2*.PROJDEFS.SHIPLOG: Updated via the SHIPIT exec. Contains log information created during the use of the SHIPIT exec.

Data Set* SCLM2*.PROJDEFS.INFOS: Update by hand. Contains the logon news for **SCLM2** users, and other useful information.

At least the following member must exist:

NEWS The news information to be shown during logon of an **SCLM2** user. Update done by hand.

G.2 VSE Project View under HLQ SCLM2

This project definition describes the view under HLQ SCLM2 on the BOESCLM machine provided by MVS System Support.

The VSE project is used to create product administration projects for HLQ SCLM2.

G.3 Overall Project Definitions

G.3.1 General SCLM Project Control and Exit Definitions

The FLMCNTRL is only used to define global project parameters. All group related definitions are defined using the FLMALTC macro and the ALTC= parameter of the FLMGROUP macro.

OPTOVER= Overwrites of translator parameters with PARMx

It is allowed to overwrite the parameters in architecture definition members.

VIOUNIT= VIO Definitions

Important

The VIO should be placed on a scratch DISK, preferably also in CACHE. If not, then with a very big MAXVIO the available storage for other things will be reduced and GETMAIN and other strange abends, for example (80A), may occur.

See also the Project Manager's Guide manual under "VIOUNIT control option" and "VIO limit".

MAXVIO= Maximum number of records to go to VIOUNIT.

MAXLINE= Number of lines to be produced in the SCLM reports.

VERS=, VERPDS=
If versioning is to be done FLMCNTRL requires these.

***JCLGEN DEFLOCS**
The default allocation values to be used for allocating the SCLM controlled data sets.

G.3.2 Control Data Sets for Common Hierarchy Section

All types of the common groups are shared across all SCLM1 and SCLM2 projects. So the account file will always be under HLQ SCLM1.

However, the groups can only be updated from the SCLM1 project and are read-only from the SCLM2 projects.

G.4 VSE Hierarchy

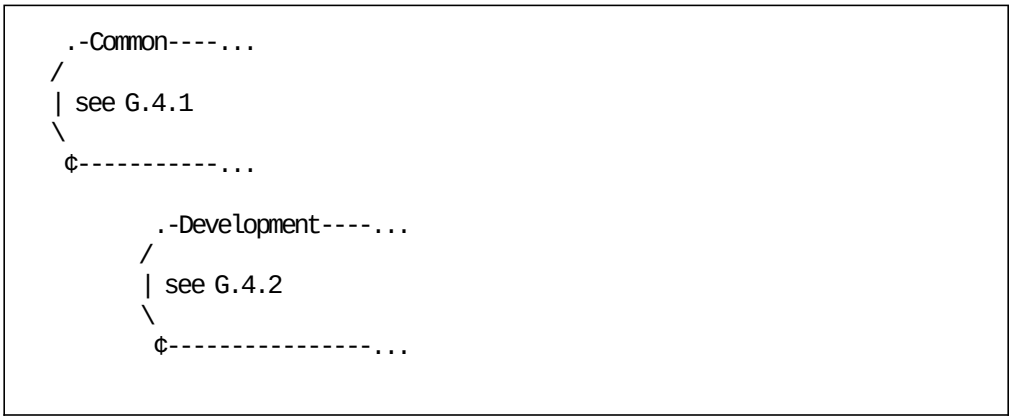


Figure 40. VSE Hierarchy

G.4.1 COMMON Hierarchy Section for Reuse of Parts

Figure 41. Common Work Hierarchy

The SCLM Support Group uses the hierarchy section to provide parts that will be commonly used and shared across different project or work units within the same project.

COMMON The group that contains members to be used across all project definitions to create project definitions. It is at least available for HLQ SCLM1 and HLQ SCLM2 projects. However can only be updated by HLQ SCLM1 projects. This is ensured by using the authorization code COMMON only for the common groups. Additionally it is protected by RACF layers.

G.4.2 Development Work Hierarchy Section for SCLM2 HLQ Views

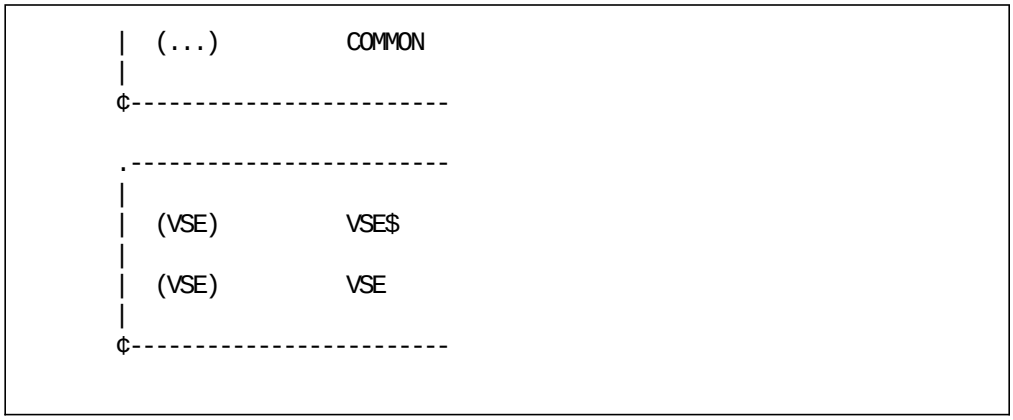


Figure 42. Development Work Hierarchy Section for SCLM2 HLQ Views

This hierarchy section is used to develop SCLM product development definitions for the VSE project on HLQ SCLM2 for HLQ VSE on the BOESCLM system.

VSE$ The group contains parts that are in production for projects of HLQ VSE. Basically stored under VSE.PROJDEFS.*.

VSE This group contains parts that are in update mode by the member of the VSE development or integration team who is responsible for the development of VSE SCLM release development environment(s).

G.5 Development Task Related Types

G.5.1 Definitions

The following chapters will describe the development tasks and the types used for those development tasks. Figure 43 on page 179 shows the syntax to define the relationship between the types and languages used in the project.

Syntax	Description
>>-	Start of process definition
-<<	End of process definition
/arch_type/	SCLM controlled architecture type (HL, LEC, CC, Gen)
|data_type|	SCLM controlled type of data members used as input or output of a language
+xxxxxx+	SCLM controlled types to be used as includes in the language
{xxxxxx}	Reference to types of data members which are created in a previous build and now referenced in an LEC architecture members.
(language)=>	Language attached to architecture members
(language)->	Language attached to data members to create outputs from one input data member

Figure 43. Process Syntax Diagram Notation

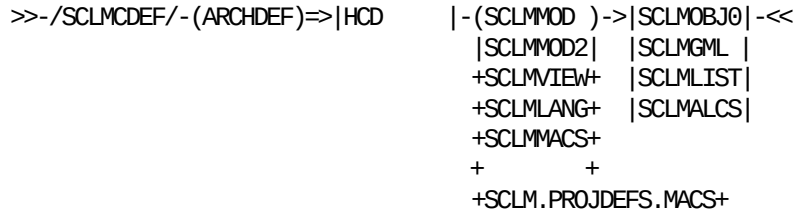
How to read the syntax is described in the following two examples:

Example 1

```
>>-/SCLMPDEF/-(ARCHDEF)=>
>>-/SCLMLDEF/-(ARCHDEF)=>{SCLMOBJ0}-(LE370   )->|SCLMLOAD|-<<
                         {SCLMOBJ1}             |SCLMLMAP|
```

This specifies that there is an architecture hierarchy SCLMPDEF which contains SCLMLDEF, both with the language ARCHDEF. The SCLMLDEF contains direct or indirect references to members stored in SCLMOBJ0 and SCLMOBJ1. Associated to the build issued to members of type SCLMLDEF is the language LE370 which may produce members of the types SCLMLOAD and/or SCLMLMAP.

Example 2



This specifies that there is an architecture type containing members with the language ARCHDEF. The SCLMCDEF contains a reference to members stored in type HCD. The associated language for a build of such members in type HCD is SCLMMOD. This language includes members from the extended type SCLMMOD2 and via FLMINCLS from types SCLMVIEW, SCLMLANG, SCLMMACS and the data set SCLM.PROJDEFS.MACS. Possible output goes into types SCLMOBJ0, SCLMGML, SCLMLIST, SCLMALCS.

G.5.2 Defaults

D

Maximum allocation

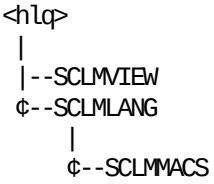
D The default maximum allocation for all types is set to 10 tracks.

G.5.3 Develop SCLM Projects

This project definition provides types and languages to edit and maintain SCLM project definitions to create SCLM projects which are themselves allowed to work in an SCLM controlled environment.

G.5.3.1 SCLM Project Creation

It is assumed that the following include chain of types is used to create the SCLM project definition:



where <hlq> is the type of the project definition to be created.

The use of types is explained in Table 9 on page 181.

G.5.3.2 Documentation of Project Definitions

It is assumed that the following imbed hierarchy of types is used to create the SCLM project documentation:

```
SCLMDOCS
 |   (#<hlq>)  The Administration Guide
 |   ($<hlq>)  The User¢s Guide
 |   (...   )  Other Master Documents
 |
 |--SCLMGMLI   Imbed members to the master documents
 |--SCLMGMLO   Outputs from SCLM assembler members used as imbeds
 ¢--........   Via the documentation languages there are other
               types available from where information may be
               included
```

The use of types is explained in Table 9 and in Table 13 on page 184.

G.5.3.3 Languages Used

Table 9. Language and Process Diagrams for SCLM Project Development. For notation used see Figure 43 on page 179.

Language name	**Process diagram**
ARCHDEF	A.3, “Process Architecture Definition” on page 99

LLE370	A.8, “Link Objects to an Executable Load Module” on page 101
	`>>-/SCLMDEFP/-(ARCHDEF)=>`
	`>>-/SCLMDEFL/-(ARCHDEF)=>{SCLMDEFC}-(LE370 )->|SCLMLOAD|-<<`
	`+SCLMOBJ + |SCLMLMAP|`

SCLMMOD	A.6, “Process SCLM Project Definitions” on page 100
	`>>-/SCLMDEFC/-(ARCHDEF)=>|<hlq> |-(SCLMMOD )->|SCLMOBJ |-<<`
	`+SCLMVIEW+ |SCLMGMLO|`
	`+SCLMLANG+ |SCLMLIST|`
	`+SCLMMACS+ |SCLMJCL |`
	`+ +`
	`+SCLM.ISPF.V41.SISPMACS+`
	For include chains see C.2, “I$SCLM SCLMVIEW - Internal SCLM Macro Libraries” on page 120 and C.1, “E#SCLM SCLMVIEW - External SCLM Macro Libraries” on page 119.

SCLMVIEW	A.7, “Process SCLM Project Parts” on page 100 `>>-/SCLMDEFV/-(ARCHDEF)=>|SCLMVIEW|-(SCLMMAC )->|SCLMGMLO|-<<`

SCLMACS	A.7, “Process SCLM Project Parts” on page 100
	`>>-/SCLMDEFS/-(ARCHDEF)=>|SCLMLANG|-(SCLMMAC )->|SCLMGMLO|-<<`
	`|SCLMMACS|`

SCRIPT
BOOKIE
BOOKMGR
	`>>-/SCLMDEFD/-(ARCHDEF)=>|SCLMDOCS|-(... )->|BM3820 |-<<`
	`|SCLMDOCS| |BMBOOK |`
	`|SCLMGMLI|`
	`+... +`
	`+... +`
	For more details see G.5.5, “Develop BookMaster and BookManager Output” on page 184

G.5.3.4 Types Provided

To develop SCLM project definition the following types are necessary.

Table 10 (Page 1 of 2). Types Used to Develop SCLM Project Definitions. For language processes see Table 9 on page 181 and Table 13 on page 184.

Type (ext.)	Description	Output from Lang.(s)	Input to Lang.(s)
	Process Control types		
SCLMDEFP	Overall SCLM Project definitions to create one logical connected set of outputs in one build.		ARCHDEF
SCLMDEFL	The project definition link controls to create the load modules. Member names are the same as in type <hlq>		ARCHDEF
SCLMDEFC	The project definition compile controls to create the objects. Member names are the same as in type <hlq>		ARCHDEF
SCLMDEFV	The project section definition controls. Names are the same as in type SCLMVIEW		ARCHDEF
SCLMDEFD	The project documentation process controls.		ARCHDEF
	Project Definition types		
<hlq>	SCLM project views for target highlevel qualifier. Name corresponds to the highlevel qualifier for which a SCLM setup is created. Member names in this type either map to the hlq or the group names which must be edit types in special cases.		SCLMMOD
SCLMVIEW	SCLM project related section definitions. These can be reused and combined to create different views on the same project. Naming convention used: **member Contains** **A@...** Authorization codes (FLMAGRP) **C@...** Control data set definitions (FLMCNTRL, FLMALTC) **G@...** Set of Group definitions (FLMGROUP) **L@...** Set of language definitions. Contains either reference to members of SCLMLANG via COPY or macro reference. **T@...** Set of Type definitions (FLMTYPE) **V@...** Set of Versioning definitions (FLMTVER) **I@...** Include libraries of SCLM types (FLMINCLS) **E@...** Include libraries of external data sets (FLMSYSLB) where @ is used if the member is included by COPY and # is used if the member is referenced as an assembler macro.		SCLMMAC
SCLMLANG	SCLM project language definitions. These are defined for each project using own or predefined translators from SCLMMACS.		SCLMMAC
SCLMMACS	SCLM common definitions such as standard translator, customizable translator steps. Projects should normally have no need to modify members in this type. They will be provided by the SCLM Support Group Naming convention used: **member Contains** **P$...** Parser translators (FLMTRNSL FUNCTN=PARSE) **B$...** Build translators (FLMTRNSL FUNCTN=BUILD) **C$...** Copy translators (FLMTRNSL FUNCTN=COPY) Where $ is used if the member is included by COPY. # is used if the member is referenced as an assembler macro.		SCLMMAC
	Project Documentation types		
SCLMDOCS (SCLMGMLI)	Bookie format of SCLM master documents		SCRIPT BOOKIE BOOKMGR
SCLMGMLI	Bookie format of documentation imbed members		SCRIPT BOOKIE BOOKMGR

Table 10 (Page 2 of 2). Types Used to Develop SCLM Project Definitions. For language processes see Table 9 on page 181 and Table 13 on page 184.

Type (ext.)	Description	Output from Lang.(s)	Input to Lang.(s)
SCLMGMLO (BMSCRIPT)	Bookie format of SCLM definition files generated from SCLM assembler formats. Projects should never modify members of this type.	SCLMMOD SCLMMAC	SCRIPT BOOKIE BOOKMGR
Project Compile/Link Output types			
SCLMOBJ	The object deck from project assembly.	SCLMMOD	LE370
SCLMLIST	The assembler listing from project assembly.	SCLMMOD	
SCLMJCL	Generated allocation information to create data sets for project	SCLMMOD	
SCLMLOAD	The project load module	LE370	
SCLMLMAP	The project load module link map	LE370	

G.5.4 Develop REXX Programs and Documentation

G.5.4.1 Languages Used

Table 11. Language and Process Diagrams to Create REXX Programs. For notation used see Figure 43 on page 179.

Language name	Process diagram
ARCHDEF	A.3, “Process Architecture Definition” on page 99.

REXXMOD	A.10, “Process REXX Modules” on page 101.
	`>>-|REXXDEFP|-(ARCHDEF)=>`
	`>>-|REXXDEFC|-(ARCHDEF)=>|REXXMOD|-(REXXMOD)->|REXX |-<<`
	`|REXXGML |`

REXXMAC	A.11, “Process REXX Macros” on page 102.
	`>>-/REXXDEFP/-(ARCHDEF)=>`
	`>>-/REXXDEFC/-(ARCHDEF)=>|REXXMAC|-(REXXMAC)->|REXXGML |-<<`

...	`>>-/REXXDEFP/-(ARCHDEF)=>`
	`>>-/REXXDEFD/-(ARCHDEF)=>|REXXDOCS|-(... )->|REXX3820|-<<`
	`|REXXGMLO| |REXXBOOK|`
	`+... +`
	`+... +`
	For more details see G.5.5, “Develop BookMaster and BookManager Output” on page 184

G.5.4.2 Types Provided

To develop general purpose documentation the following types are necessary.

Table 12. Types Used to Develop Documentation Outputs. For related language process see Table 13.

Type (ext.)	Description	Output from Lang.(s)	Input to Lang.(s)
	REXX Process Control types		
REXXDEFP	REXX part process control Connects all REXXDEFC and REXXDEFD parts to one unit for processing to get a running REXX program and related documentation		ARCHDEF
REXXDEFC	REXX program process controls		ARCHDEF
REXXDEFD	REXX document process controls		ARCHDEF
	REXX Program Development file types		
REXXMOD (REXXMAC)	REXX Master source files		REXXMOD
REXXMAC	REXX Macros which are resolved by the RXPREP translator		REXXMAC
	Rexx Program Document development file types		
REXXDOCS (REXXGMLO)	The Master documents for REXX programs		SCRIPT BOOKMGR BOOKIE
REXXGMLI	Imbed files to REXXDOCS (not yet in imbed chain!!)		IMBED
REXXGMLO	Imbed files to REXXDOCS generated from REXXMOD of REXXMAC members		IMBED
	Document output format file types		
REXX3820	Formatted SCRIPT/BOOKMASTER file for printing	SCRIPT BOOKIE	
REXXBOOK	Formatted SCRIPT/BOOKMASTER file for online viewing with BookManager	BOOKMGR BOOKIE	

G.5.5 Develop BookMaster and BookManager Output

G.5.5.1 Languages Used

Table 13 (Page 1 of 2). Language and Process Diagrams to Document Development. For notation used see Figure 43 on page 179.

Language name	Process diagram
ARCHDEF	A.3, “Process Architecture Definition” on page 99
SCRIPT	A.16, “Process BookMaster Format to Create Printable Books” on page 104 >>-/BMDEF /-(ARCHDEF)=>|BMSCRIPT|-(SCRIPT)->|BM3820 |-<< |BMIMBEDS| +SCLMGMLO+ +BMMACROS+ +BMSTYLES+ For include library see C.4, “I$SCRIPT SCLMVIEW - Internal Documentation Libraries” on page 121.

Table 13 (Page 2 of 2). Language and Process Diagrams to Document Development. For notation used see Figure 43 on page 179.

Language name	Process diagram

BOOKMGR	A.17, "Create BookManager Format from BookMaster Format" on page 105
	`>>-/BMDEF /-(ARCHDEF)=>|BMSCRIPT|-(BOOKMGR )->|BMBOOK |-<<`
	`|BMIMBEDS|`
	`+SCLMGMLO+`
	`+BMMACROS+`
	`+BMSTYLES+`
	For include library see C.4, "I$SCRIPT SCLMVIEW - Internal Documentation Libraries" on page 121.

BOOKIE	A.15, "Process BookMaster Format to Create Printable or Online Books" on page 103
	`>>-/BMDEF /-(ARCHDEF)=>|BMSCRIPT|-(BOOKIE )->|BM3820 |-<<`
	`|BMIMBEDS| |BMBOOK |`
	`+SCLMGMLO+`
	`+BMMACROS+`
	`+BMSTYLES+`
	For include library see C.4, "I$SCRIPT SCLMVIEW - Internal Documentation Libraries" on page 121.

G.5.5.2 Types Provided

To develop general purpose documentation the following types are necessary.

Table 14. Types Used to Develop Documentation Outputs. For related language process see Table 13 on page 184.

Type (ext.)	Description	Output from Lang.(s)	Input to Lang.(s)
	Document Process Control types		
BMDEF ARCHDEF	BookMaster process controls		ARCHDEF
	Document Development file types		
BMSCRIPT (BMIMBEDS)	BookMaster source		SCRIPT BOOKIE
BMIMBEDS	BookMaster source imbeds		SCRIPT
	Document productivity file types		
BMMACROS	BookMaster macros provided for all projects		
BMSTYLES	BookMaster styles provided for all projects		SCRIPT
	Document output format file types		
BM3820	Formatted SCRIPT/BOOKMASTER file for printing	SCRIPT	
BMBOOK	Formatted SCRIPT/BOOKMASTER file for online viewing	BOOKMGR BOOKIE	

*

G.5.6 Versioning of Members

Keep versions of members in group VSE$ for the following input types:

SCLMDEFP	SCLMDEFC	SCLMDEFV	SCLMDEFD
VSE	SCLMVIEW	SCLMLANG	SCLMMACS

G.5.6.1 Languages Used in the Project Definition

Note to the reader

Please refer to Appendix A, “SCLM Definitions and Functions for Supported Languages” on page 95 for the languages used.

Appendix H. Sample Meta Project Implementation

This chapter lists the members for the user documentation shown in G.2, “VSE Project View under HLQ SCLM2” on page 176. It illustrates the implementation sections only by using the .SETDVCF CODE ON flag that is for conditional formatting and results from the output of the exec REXX2LLD.

The source Appendix F, “VSE Project View under HLQ SCLM2 in Source Edit Format” on page 169 was input to the REXX2LLD program. This then created one file which is shown in this document under H.1, “Member VSE SCLM2” and G.2, “VSE Project View under HLQ SCLM2” on page 176. If the reader compares those sections he will see how they relates to each other.

H.1 Member VSE SCLM2

Code Fragment 1 (Member=VSE SCLM2)

```
*********************************************************************
*  Change activities:
*  950801 JP Created
*********************************************************************
```

End of Code Fragment 1 (Member=VSE SCLM2)

Code Fragment 2 (Member=VSE SCLM2)

```
         TITLE   ¢VSE project view under HLQ SCLM2 on system BOESCLM¢
SCLM2    FLMABEG
*
```

End of Code Fragment 2 (Member=VSE SCLM2)

Code Fragment 3 (Member=VSE SCLM2)

```
*
      COPY C@
      COPY C@#COM
```

End of Code Fragment 3 (Member=VSE SCLM2)

Code Fragment 4 (Member=VSE SCLM2)

```
*
#VSE     FLMALTC  ACCT=SCLM2.PROJDEF$.ACCOUNT1.VSE,                  *
                  ACCT2=SCLM2.PROJDEF$.ACCOUNT2.VSE,                *
                  VERS=SCLM2.PROJDEF$.AUDIT.VSE,                    *
                  VERPDS=@@FLMDSN.VERSION
```

End of Code Fragment 4 (Member=VSE SCLM2)

Code Fragment 5 (Member=VSE SCLM2)

```
*JCLGEN XLOCS 1
        COPY G@#COM      The common hierarchy
*
```

End of Code Fragment 5 (Member=VSE SCLM2)

Code Fragment 6 (Member=VSE SCLM2)

```
*-------------------------------------------------------------------
VSE$     FLMGROUP ALTC=#VSE,AC=(VSE),KEY=Y,PROMOTE=COMMON
VSE      FLMGROUP ALTC=#VSE,AC=(VSE,TEMP),KEY=Y,PROMOTE=VSE$
*
*-------------------------------------------------------------------
*
      COPY T@      The global definition for all types
```

End of Code Fragment 6 (Member=VSE SCLM2)

Code Fragment 7 (Member=VSE SCLM2)

```
      COPY T@SCLM  The SCLM Project Development Types
*
*JCLGEN DCB   FB,80,23200
VSE      FLMTYPE   Unique type referred in docu as <hlq> type
```

End of Code Fragment 7 (Member=VSE SCLM2)

Code Fragment 8 (Member=VSE SCLM2)

```
      COPY T@REXX  The REXX Development Types
```

End of Code Fragment 8 (Member=VSE SCLM2)

Code Fragment 9 (Member=VSE SCLM2)

```
      COPY T@BOOK  The Documentation Development Types
```

End of Code Fragment 9 (Member=VSE SCLM2)

Code Fragment 10 (Member=VSE SCLM2)

```
*
         FLMATVER VERSION=NO,GROUP=*,TYPE=*
*
         FLMATVER VERSION=YES,GROUP=VSE$,TYPE=SCLMDEFP
         FLMATVER VERSION=YES,GROUP=VSE$,TYPE=SCLMDEFC
         FLMATVER VERSION=YES,GROUP=VSE$,TYPE=SCLMDEFV
         FLMATVER VERSION=YES,GROUP=VSE$,TYPE=SCLMDEFD
*
         FLMATVER VERSION=YES,GROUP=VSE$,TYPE=VSE
         FLMATVER VERSION=YES,GROUP=VSE$,TYPE=SCLMVIEW
         FLMATVER VERSION=YES,GROUP=VSE$,TYPE=SCLMLANG
         FLMATVER VERSION=YES,GROUP=VSE$,TYPE=SCLMMACS
*
```

End of Code Fragment 10 (Member=VSE SCLM2)

Code Fragment 11 (Member=VSE SCLM2)

```
*
         COPY     L@                    All standard Languages
*
```

End of Code Fragment 11 (Member=VSE SCLM2)

Code Fragment 12 (Member=VSE SCLM2)

```
*
         FLMAEND
```

End of Code Fragment 12 (Member=VSE SCLM2)

H.2 Member C@ SCLMVIEW

Code Fragment 1 (Member=C@ SCLMVIEW)

```
*************************************************************************
*
* Change Activities:
* 950619 JP Created
*************************************************************************
```

End of Code Fragment 1 (Member=C@ SCLMVIEW)

Code Fragment 2 (Member=C@ SCLMVIEW)

```
*JCLGEN DEFLOCS 500,250,200
*
        FLMCNTRL OPTOVER=Y,         * PARMx overwrites allowed         *
              MAXVIO=200000000,     * VIO maximum number of records    *
              VIOUNIT=SYSDA,        * VIO unit to be used              *
              MAXLINE=55,           * # of lines in reports used       *
              VERS=SCLM1.PROJDEF$.AUDIT,                               *
              VERPDS=@@FLMDSN.VERSION
```

End of Code Fragment 2 (Member=C@ SCLMVIEW)

H.3 Member C@#COM SCLMVIEW

Code Fragment 1 (Member=C@#COM SCLMVIEW)

```
*************************************************************************
*
* Change Activities:
* 950619 JP Created
*************************************************************************
```

End of Code Fragment 1 (Member=C@#COM SCLMVIEW)

Code Fragment 2 (Member=C@#COM SCLMVIEW)

```
*
#COM    FLMALTC  DSNAME=SCLM1.@@FLMGRP.@@FLMTYP,                       *
              ACCT=SCLM1.PROJDEF$.ACCOUNT1.COM,                        *
              ACCT2=SCLM1.PROJDEF$.ACCOUNT2.COM,                       *
              VERS=SCLM1.PROJDEF$.AUDIT.COM,                           *
              VERPDS=@@FLMDSN.VERSION
```

End of Code Fragment 2 (Member=C@#COM SCLMVIEW)

H.4 Member G@#COM SCLMVIEW

Code Fragment 1 (Member=G@#COM SCLMVIEW)

```
*********************************************************************
* Change Activities:
* 950616 JP Created
*********************************************************************
```

End of Code Fragment 1 (Member=G@#COM SCLMVIEW)

Code Fragment 2 (Member=G@#COM SCLMVIEW)

```
*--------------------------------------------------------------------
COMMON   FLMGROUP ALTC=#COM,AC=(COMMON),KEY=Y
*--------------------------------------------------------------------
```

End of Code Fragment 2 (Member=G@#COM SCLMVIEW)

H.5 Member T@ SCLMVIEW

Code Fragment 1 (Member=T@ SCLMVIEW)

```
*********************************************************************
* Change Activities:
* 950619 JP Created
*********************************************************************
```

End of Code Fragment 1 (Member=T@ SCLMVIEW)

Code Fragment 2 (Member=T@ SCLMVIEW)

```
*JCLGEN MAXALLOC 10,TRK
```

End of Code Fragment 2 (Member=T@ SCLMVIEW)

H.6 Member T@SCLM SCLMVIEW

Code Fragment 1 (Member=T@SCLM SCLMVIEW)

```
*********************************************************************
* Change Activities:
* 950619 JP Created
*********************************************************************
```

End of Code Fragment 1 (Member=T@SCLM SCLMVIEW)

Code Fragment 2 (Member=T@SCLM SCLMVIEW)

```
*
*ALLOCDEF TYPE : -DCB FB,80,3120
SCLMDEFP FLMTYPE
SCLMDEFL FLMTYPE
SCLMDEFC FLMTYPE
SCLMDEFV FLMTYPE
SCLMDEFS FLMTYPE
SCLMDEFD FLMTYPE
*ALLOCDEF TYPE : -DCB FB,80,23200
*<hlq>   FLMTYPE  is defined outside of this member!!!!
SCLMVIEW FLMTYPE
SCLMLANG FLMTYPE
SCLMMACS FLMTYPE
SCLMJCL  FLMTYPE
*ALLOCDEF TYPE : -DCB VB,255,27998
SCLMDOCS FLMTYPE EXTEND=SCLMGMLI
SCLMGMLI FLMTYPE
SCLMGMLO FLMTYPE
*ALLOCDEF TYPE : -DCB FB,80,3120
SCLMOBJ  FLMTYPE
*ALLOCDEF TYPE : -DCB U,0,23200
SCLMLOAD FLMTYPE
*ALLOCDEF TYPE : -DCB FB,121,6050
*JCLGEN MAXALLOC 1,CYL
SCLMLIST FLMTYPE
*ALLOCDEF TYPE : -DCB VBA,137,3120
SCLMLMAP FLMTYPE
*JCLGEN MAXALLOC 10,TRK
*
```

End of Code Fragment 2 (Member=T@SCLM SCLMVIEW)

H.7 Member T@REXX SCLMVIEW

Code Fragment 1 (Member=T@REXX SCLMVIEW)

```
/*********************************************************************/
* Change Activities:
* 950823 JP1 Created
/*********************************************************************/
```

End of Code Fragment 1 (Member=T@REXX SCLMVIEW)

Code Fragment 2 (Member=T@REXX SCLMVIEW)

```
*--------------------------------------------------------------------
*
*
*ALLOCINFO TYPE : -DCB FB,80,3120
REXXDEFP FLMTYPE
REXXDEFC FLMTYPE
REXXDEFD FLMTYPE
*ALLOCINFO TYPE : -DCB VB,255,23476
REXXMOD  FLMTYPE  EXTEND=REXXMAC
REXXMAC  FLMTYPE
REXX     FLMTYPE
*ALLOCINFO TYPE : -DCB VB,255,27998
REXXDOCS FLMTYPE  EXTEND=REXXGMLO
REXXGMLI FLMTYPE
REXXGMLO FLMTYPE
*
*ALLOCINFO TYPE : -DCB FB,4096,28672
REXXBOOK FLMTYPE                    BookManager formatted output
*ALLOCINFO TYPE : -DCB VBM,8205,8209
REXX3820 FLMTYPE                    BookMaster formatted output
*
*--------------------------------------------------------------------
```

End of Code Fragment 2 (Member=T@REXX SCLMVIEW)

H.8 Member T@BOOK SCLMVIEW

Code Fragment 1 (Member=T@BOOK SCLMVIEW)

```
************************************************************************
* Change Activities:
* 950619 JP created
************************************************************************
```

End of Code Fragment 1 (Member=T@BOOK SCLMVIEW)

Code Fragment 2 (Member=T@BOOK SCLMVIEW)

```
*-----------------------------------------------------------------------
*
*ALLOCDEF TYPE : -DCB FB,80,3120
BMDEF    FLMTYPE
*ALLOCDEF TYPE : -DCB VB,255,27998
BMSCRIPT FLMTYPE EXTEND=BMIMBEDS
BMIMBEDS FLMTYPE
*
BMMACROS FLMTYPE
BMSTYLES FLMTYPE
*ALLOCDEF TYPE : -DCB FB,4096,28672
BMBOOK   FLMTYPE                      BookManager formatted output
*ALLOCDEF TYPE : -DCB VBM,8205,8209
BM3820   FLMTYPE                      BookMaster formatted output
*ALLOCDEF TYPE : -DCB VB,8204,8208
*BMPSEGS  FLMTYPE                      page segments for docs
*
*-----------------------------------------------------------------------
```

End of Code Fragment 2 (Member=T@BOOK SCLMVIEW)

H.9 Member L@ SCLMVIEW

Code Fragment 1 (Member=L@ SCLMVIEW)

```
*
         COPY    ARCHDEF             SCLM Architecture Language
         COPY    TEXT                SCLM Text Language
         COPY    SHIPIT              SHIP Item Language
*
         COPY    SCLMMOD             SCLM Language for modules
         COPY    SCLMMAC             SCLM Language for macros
         COPY    LE370               Linkage Editor Language
         COPY    LOAD                LOAD Module Language
*
         COPY    REXXMOD             REXX master source
         COPY    REXXMAC             REXX macro source
         COPY    REXX                REXX executable
*
         COPY    IMBED               BookMaster imbed files
         COPY    SCRIPT              BookMaster files Language
         COPY    BOOKMGR             BookManager files Language
         COPY    BOOKIE              BookManager + BookMaster
*
```

End of Code Fragment 1 (Member=L@ SCLMVIEW)

Appendix I. Special Notices

This publication is intended to help System programmers migrating their existing program development systems for VSE and MVS environments to a new platform using the System Configuration and Library Management package of ISPF Version 4 Release 2.

The information in this publication is not intended as a specification of any programming interfaces that are provided by ISPF Version 4 Release 2. See the PUBLICATIONS section of the IBM Programming Announcement for ISPF V4R2 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (″vendor″) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

The following document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

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

ACF/VTAM	AIX
APPN	BookManager
BookMaster	CICS
CUA	Current
DB2	DFSMS
GDDM	IBM
ILE	OS/2
PSL	RACF
RETAIN	RMF
SXM	VM/ESA
VSE/ESA	VTAM
XT	

The following terms are trademarks of other companies:

C-bus	Corollary, Inc.
DOS	Microsoft Corporation
HP	Hewlett-Packard Company
PC Direct	Ziff Communications Company (used by IBM Corporation under license)
SX	Intel Corporation
UNIX	X/Open Company Ltd. (registered trademark in the United States and other countries)
Windows, Windows 95 logo	Microsoft Corporation

Appendix J. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

J.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How To Get ITSO Redbooks” on page 201.

- *Software Configuration and Library Manager WSP/2 - Usage Guide*, GG24-3538
- *Version 4 of ISPF and SCLM - An Implementation Guide*, GG24-4407

J.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RISC System/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RISC System/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection (available soon)	SBOF-7250	SK2T-8042

J.3 Other Publications

These publications are also relevant as further information sources.

- *PSP, IBM Technical Disclosure Bulletin, Vol. 38 No. 03 March 1995, Maintain Design and Program Source Together as One Entity, Johann Pramberger*, GE893-0405
- *ISPF Version 4 Release 2.0 SCLM Developer's Guide*, SC34-4469
- *ISPF Version 4 Release 2.0 SCLM Project Manager's Guide*, SC34-4470
- *ISPF Version 4 Release 2.0 SCLM Reference*, SC34-4471

How To Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL http://www.redbooks.ibm.com.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get lists of redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Home Page on the World Wide Web**

  http://w3.itso.ibm.com/redbooks

- **IBM Direct Publications Catalog on the World Wide Web**

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.

- **ITSO4USA category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 415 855 43 29 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com
- **On the World Wide Web**

Redbooks Home Page	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

- **Please put me on the mailing list for updated versions of the IBM Redbook Catalog.**

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number _____
- Credit card number _____

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.

Glossary

A

access key. An identifier used to restrict access to a member.

accounting database. See accounting data set.

accounting data set. VSAM data set containing SCLM control information. For every member the data set contains at least an accounting record. Information stored in the accounting record of a member includes the language assigned to the member, its creation and change dates and times, whether the member has been built, and statistical and dependency information. Another type of record in the accounting data set consists of the building map.

accounting information. Accounting records, cross-reference records, intermediate records and build maps.

accounting record. An SCLM datarecord containing statistical, historical, and dependency information for a member under SCLM control.

alternate project definition. A project definition that provides a version of the project environment which differs from the default project definition. It can be used in place of a project definition. It is a member name of a load module in the project.PROJDEFS.LOAD data set. The name of this module must be explicitly specified if you are referring to it.

application. Software that performs a function for an end user.

API. Application Programming Interface

archdef. See architecture definition.

architecture. The organization of software components to form integrated applications.

architecture definition. The specification of relationships between software components of an application. These relationships can be explicitly affected via archdefs.

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 an hierarchy. They are a property of a member under SCLM control.

B

build. Build usually involves compile or link. Build means to apply the language stored in the member;s accounting record (and defined in the project definition) to the member. Toy can build individual source members or whole applications consisting of multiple source members referenced by an architecture definition. Building a linkage edit control architecture definition means compiling and link-editing the referenced members.

build map. Internal data record containing a complete analysis of the data base at the time of the build; it includes the names of all referenced members and the last change data and version number of each member.

build step. The execution of a build of an entity that generates a build map.

C

change code. Reason code associated the an update or modification to a member controlled by SCLM.

check in. Putting a member that has been brought to a file system outside a library back to the control library. The lock is removed in the accounting database of the library. The converse of check in is check out.

check out. Getting a member (a ″part″) to the file system outside of the library for modification. The member is locked in the library. The converse of check out is check in.

CLEAR. A library and packaging facility running on MVS which is used by IBM internally.

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 control (CC). A type of architecture definition to control source compilation.

component. See software component.

concurrent updates. Two programmers update the same member in different ways in different development groups.

conditional reference. An include reference construct that depends on information outside the scope of a single line.

configuration management. See software configuration management.

configuration management plan. See software configuration management plan.

control data. Data that contains statistical, include, dependency, status and tracking information about all SCLM controlled members.

copylib. A library containing include referenced source code.

D

data base. SCLM-controlled data sets for a project

database administrator. See project administrator.

ddname substitution list. A strong of ddnames allocated for the translator. The ddname substitution list is usually documented in the Programmer's Guide for compilers and linkage editors.

default project definition. The main project definition used by an SCLM project.

dependency. A software component necessary for the completion of another software component.

dependency information. Identifies software components that need another software component to complete successfully.

development group. A group that has no group that promotes into it.

development layer. Layer of an SCLM hierarchy consisting of development groups.

development life cycle. The process followed to create an application. The process starts at the program requirements gathering phase, moves to the design phase, the development phase, and continues to the release of the final product.

downward dependency. A dependency indicating a compilation unit which must be compiled after the current compilation unit is compiled.

drawdown. To copy a member or a compilation unit to a development group from the first appearance in a higher key or primary group in the library concatenation. Such a member will be locked at all higher levels against concurrent promote. Drawdown is done automatically when a member is edited.

dynamic include. An include for a source member that cannot be resolved until after the translator invocation.

Dynamic reference. A reference that involves a variable.

E

editable/non-editable. Source members (created by an edit session) are editable; members produced by a processor during a build are non-editable.

external library. A library which connects logically to the SCLM hierarchy. Groups and translators for these libraries can be defined using the SCLM project definition macros.

G

group. A set of project data sets with the same middle-level qualifier in the SCLM logical naming convention.

H

hierarchical view. A path of groups (concatenation) through the hierarchy. The path may start at any group in the hierarchy and follows the promote path to the topmost group in the hierarchy.

hierarchy. The organization of groups in a ranked order, where each group is subordinate to the one above it.

HL. High-level, a type of architecture definition. It contains references to other architecture definitions or compilable units.

I

include. A member that is required to complete a compile of the member that references it.

include-set. An include-set is used to associate an included member name with the type or types in the project which are searched to find a member with that name.

integrate. To merge two or more software components of an application into a single software application.

internal data set. A VSAM data set that contains internal data.

K

key group. A group into which data is moved(as opposed to copied) during promotion. Key groups are always allocated when you work with an SCLM hierarchy.

L

language definition. Languages in SCLM must be specified in the project definition. They define the processes that source members will undergo during build.

layer. A given tier of the hierarchy, made up of groups of equivalent rank.

level. See layer

library (MVS). A partitioned data set.

library management. SCLM uses hierarchies for library management. Members are moved between different hierarchy layers by SCLM′s promote and drawdown.

linkage edit control (LEC). A type of architecture definition to control link-editing of load modules.

LINKJCL. JCL delivered with an MVS product which is used during an SMPE controlled product installation to perform the final link of this product.

lock. To preclude other programs from updating a member (usually associated with drawdown).

M

MAKE. A technique typically used on OS/2 and AIX to automate compiles and links.

maximum promotable group. The topmost group to which a member can be promoted.

member. The discrete element of an SCLM database, representing a single data type of a software component.

metadata. This term is used for all data that is SCLM controlled but does not result in a deliverable such as source code; it consists of descriptive elements to drive the process of creation of deliverables. One type are architecture members. Others are self defined and are used as process parameters, i.e. for the packaging process.

meta project. Deservices an SCLM project used to create another SCLM project.

migrate. Registering software components in SCLM: this includes identifying the component language, and possibly the change code and authorization code.

migration. Introduction of software components into an SCLM database.

N

NCAL. Never call, a linkage editor option to create a load module without resolving external references.

nonkey group. A group in the SCLM hierarchy into which data is copied (as opposed to moved) during promotion. Nonkey groups are not always allocated by SCLM. An example is the Edit panel of SCLM where nonkey groups never show up. A nonkey group results in duplication of code after promotion.

P

packaging. Preparation of product parts as an entity to allow easy installation in a structured and predefined way.

parse. Synonym for analyzing in a formal way. In SCLM, parsing means to gather statistics of a member and determine include relationships.

part. Part is another word for member when used in conjunction with SCLM.

partitioned data set. See PDS.

PDS. Partitioned data set. All SCLM-controlled members are stored in partitioned data sets or partition data sets extended.

PDSE. Partitioned data set extended. See PDS.

phase. Module on a VSE/ system.

predecessor date/time. The last modified date/time stamp taken from the previous version of the current member.

predecessor version. The process of verifying that the previous version of a member has not changed.

predecessors. Previous versions of a member existing at a higher level within the same hierarchical view.

primary group. A key or nonkey group with two or more groups promoting into it that must be allocated when a hierarchy is to be accessed.

primary nonkey group. A nonkey group with more than one lower level group promoting into it.

primary project definition. See default project definition.

private library. A partitioned data set or partitioned data set extended belonging to a group in the development layer of the hierarchy.

PROJDEFS. Second prefix of the hlq.PROJDEFS.LOAD data set where SCLM looks for the project definition load modules. Other relevant project information can also be stored using this second qualifier.

project. An undertaking with predescribed objectives, magnitude, and duration. It consists of a set of partitioned data sets whose members are controlled by a project definition. The project definition specifies a hierarchy.

project administrator. The person who maintains an SCLM project.

project data base. All partition data sets controlled under the project definition referenced.

project definition. A customization of the SCLM product which defines the development environment for a specific project. It describes the project hierarchy, project database, languages defined, and control options. SCLM provided assembler macros are used to edit compile and link a project definition. The project definition is the member name of the load module 'project' in data set 'project.PROJDEFS.LOAD'. in all cases where no project definition name is specified, SCLM assumes that this load module is being referred to.

project definition data. Project definitions and language definitions which are used to create and control an SCLM project.

project environment. Information which makes up an SCLM project. There are three types of information:

- Project Definition Data
- User Application Data
- Control Data

project identifier. The high-level qualifier used by all data sets belonging to a particular project.

project partitioned data sets. MVS Partition Data Sets where user application data is stored.

promote. To move (or copy) members and associated internal data (accounting, status, cross-reference, dependency, and statistical) up through the hierarchy one group at a time.

promote path. The link between two groups along which data moves from one subordinate group in the hierarchy.

proxy. Something that stands for a real thing. In SCLM, proxies are used in CSP and DB2 support, where the "real thing" is a record in a VSAM file or a DB2 table that cannot be under SCLM control.

PWS. programmable work station

R

RACF. Resource access control facility.

rebase. A process to integrate parts changed in parallel to an ongoing release at certain points.

REXEC. A TCP/IP provided application to execute a command on a remote host.

S

scope. The set of members (include architecture definitions) which will be processed (verified, copied, compiled, purged, etc.) by build or promote.

service. An SCLM function available via a command or programming interface.

service parameter list.. The options supplied when invoking an SCLM service.

software component. A member of an application.

software configuration. The way a whole application fits together.

software configuration management. The method of controlling and integrating software components to produce high quality application. Provides a common point of integration for all planning and implementation activities for a project.

software configuration management plan. A formulized procedure for software configuration management.

subapplications. Separate parts of an application being developed within a project. Once the project is completed, the parts are integrated to form the final product.

syslib. A library containing source code not under SCLM control. No dependency information is maintained for member in a syslib.

T

text. Data present in its natural form (not translatable).

traceability. Capability to access and maintain records of information about a software component, including when the component was last changed and why.

translator. A software program invoked by SCLM for the purpose of parsing, compiling, verifying, purging,

copying, or any other user-defined translations of data.

translator step. Occurrence of a translator in a language definition; it can be of the type PARSE, BUILD, COPY, VERIFY or PURGE.

transparent. A transparent process is one that lets you do things without having to know about operational details. Transparent means hidden from the user. You can use data sets that have been allocated with flexible data set naming standards without having any knowledge about the actual data set name, for example.

type. The third qualifier of the SCLM naming convention for project partitioned data sets. Typically identifies the kind of data maintained for a project hierarchy. Examples of types are SOURCE, OBJECT and LOAD.

U

unlock. To make a member (formerly locked out) available for updating (usually associated with promote).

upward dependency. A dependency indicating a compilation unit that must be compiled before the current compilation unit is compiled.

user application data. User developed programs or data that are under SCLM control

W

workdef member. An architecture member that describes the scope of work that is done in one build step for potential integration into the next higher level for common system testing.

WSP/2. AD/Cycle Workstation Platform/2, the former PWS-based front end to SCLM. It is replaced with ISPF 4.2 through workstation build and edit interfaces that includes a Client/Server environment.

List of Abbreviations

APAR	Authorized Program Analysis Report
BMS	Basic Mapping Support
BPL	Basic Programming Language
CC	Compilation Control
CCID	Change Control IDentifier
CICS	Customer Information Control System
CLEAR	Control Library Environment and Resource System
DTL	Dialog Tag Language
DB2	IBM Database 2
FTP	File Transfer Program
GPO	Group Processing Option
GML	Generalized Markup Language
HCD	Hardware Configuration Definition
HL	High Level
HLQ	High Level Qualifier
IBM	International Business Machine Corporation
ITSO	International Technical Support Organization
LEC	Linkage Edit Control
MVS	Multiple Virtual Storage
NLS	National Language Support
OCO	Object Code Only
PDS	Partitioned Data Set
PDSE	Partitioned Data Set Extended
PTF	Program Temporary Fix
RACF	Resource Access Control Facility
RETAIN	REmote Technical Assistance Information Network (IBM)
REXEC	Remote EXECution
RMF	Remote Maintenance Facility
SCLM	System Configuration and Library Manager
SDF II	Screen Definition Facility II
SMP/E	System Modification Program Extended
SPE	Small Product Enhancement
TCP/IP	Transmission Control Protocol/Internet Protocol
TSO	Time Sharing Option
VPL	View Program Listings
VSE	Virtual Storage Extended
VSE/ESA	Virtual Storage Extended/Enterprise Systems Architecture

Index

Special Characters

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

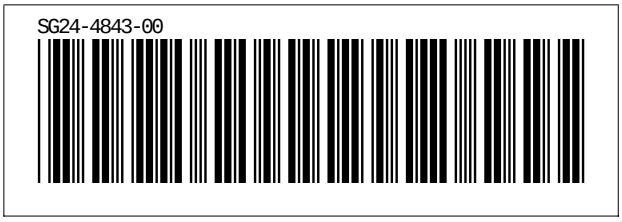
T

U

V

W

Printed in U.S.A.



Artwork Definitions

Table Definitions

Figures

Headings

Index Entries

List Items

id	File	Page	References
BPLTYPA	4843CH04	37	1
BPLTYPB	4843CH04	37	2
BPLTYPC	4843CH04	37	3
BPLTYPD	4843CH04	37	4
BPLTYPE	4843CH04	37	5
BPLLI1	4843CH05	52	4
			53

Revisions

id	File	Page	References
DES	4843VARS	i	173, 173, 176, 176, 176, 176, 180, 180

Tables

id	File	Page	References
BPLRAC2	4843CH03	23	1
			22
BPLTYP	4843CH04	38	2
			37, 37
BPLDOC1	4843CH07	71	3
			71
BPLDOC2	4843CH07	72	4
			71
BPLPKG1	4843CH08	79	5
			78, 82
BPLMODS	4843CH08	87	6
			87
PRCSCLM	4843AX07	181	9
			180, 181, 182, 183
PRCREXX	4843AX07	183	11
PRCBOOK	4843AX07	184	13
			181, 182, 183, 184, 185

Processing Options

Runtime values:

Document fileid	SG244843 SCRIPT
Document type	USERDOC
Document style	REDBOOK
Profile	EDFPRF30
Service Level	0029
SCRIPT/VS Release	4.0.0
Date	96.10.23
Time	03:26:39
Device	3820A
Number of Passes	4
Index	YES
SYSVAR D	YES
SYSVAR G	INLINE
SYSVAR S	OFFSET
SYSVAR X	YES

Formatting values used:

Annotation	NO
Cross reference listing	YES
Cross reference head prefix only	NO
Dialog	LABEL
Duplex	YES
DVCF conditions file	(none)
DVCF value 1	(none)
DVCF value 2	(none)
DVCF value 3	(none)
DVCF value 4	(none)
DVCF value 5	(none)
DVCF value 6	(none)
DVCF value 7	(none)
DVCF value 8	(none)
DVCF value 9	(none)
Explode	NO
Figure list on new page	YES
Figure/table number separation	YES
Folio-by-chapter	NO
Head 0 body text	Part
Head 1 body text	Chapter
Head 1 appendix text	Appendix
Hyphenation	NO
Justification	NO
Language	ENGL
Layout	OFF
Leader dots	YES
Master index	(none)
Partial TOC (maximum level)	4
Partial TOC (new page after)	INLINE
Print example id's	NO
Print cross reference page numbers	YES
Process value	(none)
Punctuation move characters	.,
Read cross-reference file	(none)
Running heading/footing rule	NONE
Show index entries	NO
Table of Contents (maximum level)	3
Table list on new page	YES
Title page (draft) alignment	RIGHT
Write cross-reference file	(none)

Imbed Trace

Page 0	4843SU
Page 0	4843VARS
Page 0	REDB$BOE
Page i	REDB$ED1
Page i	4843EDNO
Page i	REDB$ED2
Page xi	4843ABST
Page xi	4843ORG
Page xii	4843ACKS
Page xii	REDB$COM
Page xiv	4843MAIN
Page xiv	4843CH01
Page 6	4843CH02
Page 12	4843CH03
Page 33	4843CH04
Page 38	4843CH05
Page 53	4843CH06
Page 66	4843CH07
Page 72	4843CH08
Page 94	4843AX01
Page 106	4843AX02
Page 118	4843AX03
Page 121	4843AX04
Page 159	4843AX05
Page 167	4843AX06
Page 172	4843AX07
Page 186	4843AX08
Page 197	4843SPEC
Page 197	REDB$SPE
Page 198	4843TMKS
Page 198	4843BIBL
Page 199	REDB$BIB
Page 200	REDB$ORD
Page 203	4843GLOS
Page 209	4843ABRV