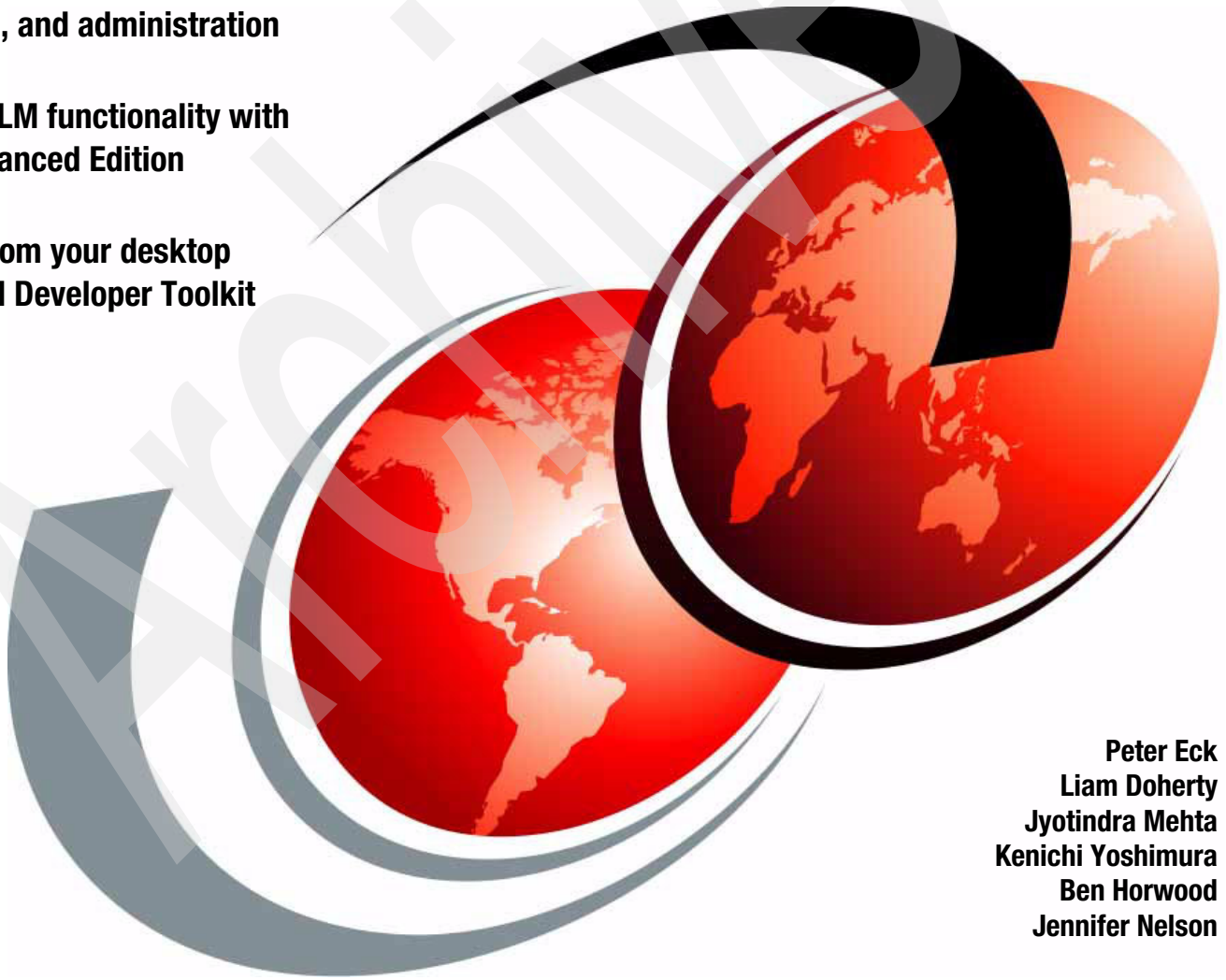
IBM

Getting Started with SCLM: A Practical Guide to SCLM and SCLM Advanced Edition

- Understand the basics of SCLM project setup, use, and administration
- Extend SCLM functionality with SCLM Advanced Edition
- Develop from your desktop with SCLM Developer Toolkit



Archived

Peter Eck
Liam Doherty
Jyotindra Mehta
Kenichi Yoshimura
Ben Horwood
Jennifer Nelson

ibm.com/redbooks

Redbooks

Archived

International Technical Support Organization

Getting Started with SCLM: A Practical Guide to SCLM and SCLM Advanced Edition

September 2007

Archived

SG24-7392-00

Note: Before using this information and the product it supports, read the information in “Notices” on page xiii.

Archived

First Edition (September 2007)

This edition applies to ISPF for Version 1 Release 8.0 of the licensed program z/OS (program number 5694-A01). Plus Version 2, Release 1 of SCLM Developer Toolkit (program number 5655-R37), Version 1, Release 1 of IBM Breeze for SCLM for z/OS (program number 5697-G58), Version 1, Release 1 of IBM Enhanced Access Control for SCLM for z/OS (program number 5697-H59) and Version 3, Release 1 of IBM SCLM Administrator Toolkit (program number 5697-N51)..

© Copyright International Business Machines Corporation 2007. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 2007. All rights reserved.

Archived

Archived

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

© Copyright IBM Corp. 2007. All rights reserved.

Trademarks

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

AIX®
AIX 5L™
C/370™
CICS®
Common User Access®
CUA®
DB2®
DFSMS™
DFSMS/MVS™
DFSMSdfp™
IBM®
IMS™
MVS™
OS/390®
RACF®
Rational®
Redbooks®
Redbooks (logo) ®
REXX™
S/370™
S/390®
System z™
Tivoli®
VTAM®
WebSphere®
z/OS®

The following terms are trademarks of other companies:

EJB, Java, JVM, J2EE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Excel, Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Pentium, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Archived

Preface

In this IBM® Redbooks® publication we describe and document a number of different aspects of the Software Configuration and Library Manager (SCLM).

Part 1 of the book focuses on setting up an SCLM project using commonly used languages such as COBOL and PL/I. Additionally, migration techniques are discussed for those customers considering migrating to SCLM from other vendor Software Configuration Management products.

Part 2 describes basic usage of SCLM functions such as Edit, Build, and Promote.

Part 3 goes a bit beyond the basics and looks at some of the newer functions that are being added to SCLM along with writing user exits and setting up SCLM for debugging with IBM Debug Tool.

Parts 4, 5, and 6 concentrate on the SCLM Advanced Edition products such as Breeze, Enhanced Access Control, SCLM Developer Toolkit, and SCLM Administrator Toolkit. These sections describe what these products are and how they can be set up and used to aid your application development.

This book is intended for project leaders, SCLM Library administrators, system programmers, and application programmers who are using, or are planning on using SCLM or one of the SCLM Advanced Edition products. The book is designed as a detailed introduction to SCLM and the SCLM Advanced Edition products.

The team that wrote this Redbooks publication

This Redbooks publication was produced by a team of specialists from around the world, as shown in Figure 1 on page xvi, Figure 2 on page xvii, and Figure 3 on page xvii, working at the International Technical Support Organization, Raleigh Center.

Peter Eck has been working with SCLM since 1999. He has led and participated in many conversions to SCLM. He has expertise in all areas of SCLM setup and use; project setup, language development, writing user exits, creating custom tools, and creating and presenting custom SCLM training. Overall, Peter has 20 years of programming and technical support experience in the IT industry.

Liam Doherty is a Software Engineer and SCLM architect in Perth, Australia. He has 25 years of experience with the MVS™ operating system both with IBM and other customers. Initially an application programmer in PL/I, IMS™, CICS®, and DB2®, he moved into more technical support roles such as IMS DBA, CICS System programmer, and SCLM Administrator before spending much of his career as a DB2 DBA. On moving to Perth, he took up a role at IBM that focused on SCLM and the SCLM additional products both from an architectural perspective and a customer facing support perspective. He was also one of the developers on SCLM Developer Toolkit. He attends the SHARE technical conference twice a year in the US to present presentations on both SCLM and ISPF.

© Copyright IBM Corp. 2007. All rights reserved.

Jyotindra Mehta is an IT specialist in Pune, India. He has 25 years of experience with application development, systems programming, and database administration on the MVS platform with IBM and other organizations. Initially, an application programmer in COBOL, ADABAS/NATURAL, and CICS, he moved into more technical support roles, such as MVS Systems programmer, DB2 DBA, and SCLM Administrator after joining IBM India. Jyo has led an SCLM implementation of an IBM Australia financial application on an IBM India mainframe server. He has led the z/OS® systems programming competency effort in the India GBS Global Delivery organization, and also provided consulting support to several customers on DB2 performance tuning. Jyo has written a technical paper on migration to SCLM, drawing on his experiences as an SCLM administrator, and also participates as an expert reviewer and presenter on IBM India's mainframe sharenets.

Kenichi Yoshimura is a software engineer in Australia Development Laboratory based in Perth, Australia. Since joining IBM in 2004, he has worked on several System z™ software products including IBM Fault Analyzer for z/OS and IBM SCLM Developer Toolkit. His main areas of expertise are Java™ on z/OS and Eclipse plug-in development. He holds a Master Degree in Engineering Science from The University of Melbourne.

Ben Horwood is a Software Engineer and a recent addition to the Australia Development Laboratory, Perth. He currently works in the ISPF/SCLM team and before IBM spent around 4 years developing applications for the Windows® platform. His current interests include Java and the world of z/OS.

Jennifer Nelson is a Product Specialist of the AD Tools with Rocket Software in Austin, Texas. She has 10 years experience in the mainframe industry working with DB2 and mainframe storage with large mainframe vendors. She is currently working as a Product Specialist with Rocket Software as a business partner with IBM assisting Technical Sales with potential and existing customers. She holds a BA in Political Science from the University of Texas at Austin.

Figure 1 From left, Peter Eck, Jyotindra Mehta, and Liam Doherty

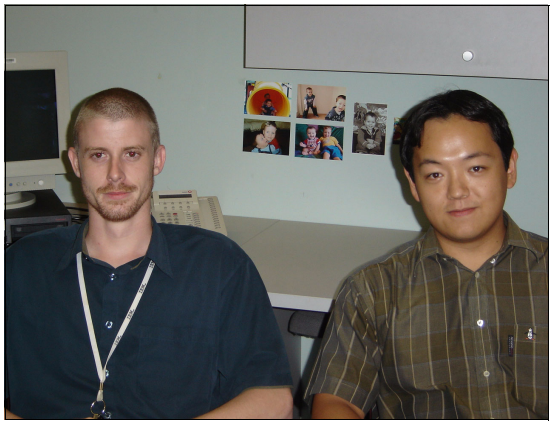


Figure 2 From left, Ben Horwood and Kenichi Yoshimura

Figure 3 Jennifer Nelson

Thanks to the following people for their contributions to this project:

John Cullen, IBM, System Programmer, Perth, Australia

Greg Henderson, IBM, System Programmer, Perth, Australia

Grant Sutherland, IBM, Lead Architect ISPF/SCLM, Perth, Australia

John Philp, IBM, SCLM Software Engineer, Perth, Australia

Paul Meaney, IBM, SCLM Developer Toolkit Software Engineer, Perth, Australia

Larry England, IBM, Development Engineer, Silicon Valley Laboratory

Chris Rayns, IBM, ITSO Poughkeepsie

Rod Turner, IBM, Senior Software Engineer, Perth, Australia

Adrian Simcock, IBM, System Programmer, Perth, Australia

Corrine Stith, Rocket Software

Johann Pramberger, IBM, Software Configuration Management Team Lead and Architect, Germany

Rachel Trout, IBM, Software Engineer, Perth, Australia

Douglas Nadel, IBM, SCLM Developer Toolkit Development, Raleigh

Archived

Yvonne Lyon, Editor, International Technical Support Organization, San Jose, CA

Debbie DeCarlo, Event Planner, Social, and Culinary Support, San Jose, CA

Joe DeCarlo, Manager, Special Projects, International Technical Support Organization, Raleigh, NC via San Jose, CA

Become a published author

Join us for a two- to six-week residency program! Help write a Redbooks publication dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks publications to be as helpful as possible. Send us your comments about this or other Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks publication form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Archived

Part 1

Basic setup and migration

In this part of the book, we discuss the basic setup of your SCLM project, creating language definitions based upon the samples and configuring your project for DB2. We also cover the steps you need to take to migrate to SCLM once your setup is complete, as well as an overview of SCLM architecture definitions. For further information about any of these topics, you can refer to the SCLM manuals.

SCLM manuals

Throughout this book, we reference the SCLM manuals.

Previously, these manuals were known as:

- *ISPF SCLM Project Manager's and Developer's Guide*, SC34-4817
- *ISPF SCLM Reference*, SC34-4818

For z/OS 1.8, the two SCLM manuals were merged into a single manual. From z/OS 1.8 onwards, it is now called:

- *ISPF SCLM Guide and Reference*, SC34-4817

Introduction to SCLM and the terminology used in this book

Software Configuration Management (SCM) is the term used to describe the tracking and control of software development. To manage your software assets, you need the right process and the right tools. Within SCM, these processes can include:

- **Library management:**

  The management of the source and versions of the source. Versioning is the making of copies of data at some meaningful point in order to return to that point at a later date, if necessary.

© Copyright IBM Corp. 2007. All rights reserved.

- **Configuration and delivery:**

  Configuration control implies a higher level of abstraction than version control. The SCM tool must have some knowledge of which versions from a set of components comprise a specific build.

- **Process management:**

  Process management deals with the grouping and manipulation of versions of software assets as they progress through the software development lifecycle. This typically involves change management, approval levels, and production control.

- **Problem tracking:**

  Problem tracking entails recording enhancement/change requests or defect reports and correlating these with the resolution of the request. These reports may include a listing of the sources involved in the change. These change sets can then create released products containing only the feature and fixes desired.

How do SCLM and SCLM Advanced Edition meet the requirements of an SCM tool?

Library management

As a library manager, SCLM manages changes to your application data, performs auditing and versioning, and controls the movement of your application from one set of staging libraries to the next (called *Promote* in SCLM).

Configuration and delivery

As a configuration manager, SCLM knows how all of the pieces of your application fit together. In SCLM, this is not limited to just standard source, object, and load modules. You can specify additional relationships in SCLM to associate your other data, such as test cases, publications, and JCL, with the application. The SCLM Build function translates your inputs into outputs. In SCLM, this is not limited to just compiling source code and linking object modules. You can use Build to drive most any procedure that translates an input into an output. On a Promote, SCLM ensures that all of your inputs and outputs are in sync. You no longer have to worry about one of your production load modules not matching the source in your production source libraries.

Process management

SCLM provides Architecture definitions that are used to group parts of an application together. These architecture definition members are then used to drive the promotion process through the development life cycle. Additionally, the SCLM Advanced Edition package approval product, Breeze, provides approval control.

Problem tracking

SCLM provides the ability to assign change codes to members as they are edited and saved. Build and promote driven by these change codes can also be performed. SCLM in itself is not a problem tracking system. However, SCLM provides numerous user exit points where you can interface with other problem tracking systems.

Archived

1

SCLM project setup

In this chapter we present the setup of a simple SCLM project with one project database and a single high-level qualifier. We give you the steps to set up the project definition manually. You can automate SCLM project setup with the help of the SCLM Administration Toolkit, which is covered in Chapter 23, “IBM SCLM Administrator Toolkit” on page 577; Chapter 24, “Introduction to the IBM SCLM Administrator Toolkit” on page 587; and Chapter 25, “Beyond the basics: A case study” on page 661.

Once you have set up the simple SCLM project and tested it out, you can enhance it to meet your configuration management (CM) needs. You must carefully analyze your needs. If you are migrating from another CM tool to SCLM, you need to decide if you want to implement an SCLM environment that matches your current environment as closely as possible or an environment that takes the best advantage of all the functions SCLM provides.

If you clone your existing environment in SCLM, you can minimize the number of changes in your existing organization and procedures. If you decide to take advantage of all of the functions SCLM provides, you may need to change your existing organization and procedures but you can greatly improve your software development and configuration management processes. If you are setting up a brand-new CM environment at your installation using SCLM, you should choose an environment that takes the best advantage of all the functions SCLM provides, and maximizes the benefits of your software development and configuration management processes.

1.1 Roles addressed throughout the book

The following roles are addressed throughout this book:

- **Project Administrator:**

  This role is responsible to create and manage the SCLM controlled project environment. This person will create the SCLM project definitions and maintain the project datasets. An overview of this role is discussed in the next section: 1.2, “Overview of the SCLM project environment” on page 4.

© Copyright IBM Corp. 2007. All rights reserved.

- **SCLM User:**

  This role is the programmer, team leader, or manager who will use SCLM to analyze, edit, build, and test their code and initiate and approve the movement of the code up the hierarchy to production. Using SCLM is covered beginning in Chapter 6, “Using SCLM edit” on page 153.

1.2 Overview of the SCLM project environment

The SCLM project environment consists of the following types of data and is maintained by the SCLM project administrator:

- User application data
- SCLM control data
- Project definition data

User application data consists of the application data (programs, copybooks, JCL, and so on) being developed for a single project. SCLM stores all user data associated with a single project as members within a hierarchical set of MVS partitioned data sets (ISPF libraries). These data sets are called the project partitioned data sets. Users refer to SCLM-controlled ISPF libraries with an SCLM naming convention containing three levels of qualification, specifically:

```
project.group.type
```

The first qualifier, project, is the unique project identifier associated with the hierarchy. SCLM organizes project data sets into groups, the second identifier within the naming convention. Each group represents a different stage or state of the user data within the life cycle of a project. For example, assume a project has three groups named DEV1, SYST, and PROD. The DEV1 group represents data being modified. The SYST group represents data being tested. The PROD group represents production data for customer use. The groups of a project are organized into hierarchical order to form a tree-like hierarchy.

A group is made up of several data sets that can contain different types of data. Types, the third qualifier of the naming convention, are used to differentiate the kinds of data maintained in the groups of a project. For example, source code would be stored in one type and listings in another type. It is better not to mix different data types in SCLM. (Although SCLM allows you to do this, it is not recommended; data with different formats should be stored in different types.)

SCLM control data is stored in the control data sets and is used to track and control application programs and members within the hierarchy. SCLM stores accounting and audit information in VSAM data sets whose names are defined in the project definition.

A project definition specifies the desired development environment to SCLM for an individual project. Using the project definition, the project administrator can define:

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

You can generate multiple project definitions for a single project. If you do, you must designate one of them as the primary project definition. All other project definitions for the same project are alternate project definitions. Use alternate project definitions for specific tasks that cannot or should not be performed with the primary project definitions. Keep alternate project definitions to a minimum.

For example, here are a couple of typical situations that are good candidates for an alternate project definition:

- You need two distinct promotion paths, one for normal development and another for emergency fixes. Define a primary project definition for the normal development with a hierarchy of DEV1, TEST and PROD groups. Define an alternate project definition for emergency fixes with a hierarchy of EMER and PROD groups. An alternate is needed in this situation, so that common members in DEV1 and EMER do not lock each other out. Use this alternate with care as it can cause additional work.

  For example, if a developer is working on a member in DEV1 for some new development and another developer needs to make a production fix to it then the other developer can use the alternate to edit the member in EMER, make the fix and then promote it back to PROD. This creates work for the DEV1 developer. They must be notified of the fix so they they can merge the fixes into their version of the member in development in DEV1. In addition the DEV1 developer must fix the predecessor error that will occur if they try to promote the member. Fixing this error is covered on the last page of Chapter 6, “Using SCLM edit” on page 153.
- You are migrating from another CM tool to SCLM, and need to populate your SCLM hierarchy in reverse order, like PROD followed by TEST followed by DEV1. You need 3 project definitions to accomplish this, because members can only be added to a group that is the lowest or only group in a hierarchy. Define the first project definition with a hierarchy of only the PROD group. Define the second project definition with a hierarchy of only TEST and PROD groups. Define the third project definition with a hierarchy of groups DEV1, TEST and PROD . Use the first definition to migrate all production modules, the second to migrate all test modules, and the last one to migrate all development modules. Designate the last one as the primary project definition and others as alternate project definitions. See Chapter 4, “Migrating to SCLM” on page 119 for a detail discussion of this approach.

1.3 Overview of sample project

In this section, we present the steps to set up a simple SCLM project. By completing the steps outlined here, you can create a project that is under SCLM control. The name of the sample SCLM project is SCLM01 and it contains a CM environment for three simple MVS batch applications, one each developed in Enterprise COBOL, z/OS high level assembler, and Enterprise PL/I. It contains all the required components of SCLM projects in general and can serve as a model for future projects.

The SCLM01 project uses a 3-level hierarchy of development, test and production. Since the team has multiple developers who may need to work in parallel at times, the project uses two development groups, DEV1 and DEV2. They coordinate their individual work, and promote to a common CM environment consisting of a single TEST group and a single PROD group.

The project uses a single type, SOURCE, for source code developed in Enterprise COBOL, Enterprise PL/I and high-level assembler. It uses separate types, COPYLIB, PLINCL and MACROS, for copy or include code for each of these languages. The project captures object modules into the OBJ type, compile listings into the LIST type, and executable code into the LOAD type. It also uses the ARCHDEF type to maintain SCLM architecture definitions used to build and promote the application.

ARCHDEF, SOURCE, COPYLIB, PLINCL and MACROS are types that contain editable members. Developers maintain modules in those types for specific requirements. The rest of the types store generated outputs. The project provides for versioning for all types that store editable members at the TEST and PROD groups. Also, the project uses authorization codes to help prevent overlaying of changes by developers.

1.3.1 Steps to set up the sample project

We use the following steps to set up the example project. You should follow the steps for your environment in the order listed and exactly as they are described. The example uses the name of the sample project as SCLM01. If you wish to use a different name for your project, choose a 1 to 8 character name that satisfies the standard MVS data set name qualifier requirements and change all occurrences of SCLM01 to the name of your choosing. When you have completed all of the steps, you will have an SCLM project definition with which you can experiment to better understand how SCLM works. If you encounter any errors during the following steps, use the JES, assembler and linkage editor messages to correct the problem.

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

   `SCLM01.PROJDEFS.SOURCE`

   This partitioned data set contains the source code for the library structure (project definition), customized language definitions as appropriate and project specific JCLs.
3. Using the ISPF Data Set Utility, allocate the following partitioned data set with space in cylinders (1,5), with 10 or more directory blocks, and with record format U, LRECL 0, BLKSIZE 6144:

   `SCLM01.PROJDEFS.LOAD`

   This partitioned data set contains the load module (machine readable code) for the library structure as defined in the project definition.

   Note: Depending on the ISPF configuration for your site, you might receive warning or error messages when attempting to edit an SCLM project using the ISPF editor.

4. Using the ISPF Move/Copy Utility, copy the following members from the ISPF systems macro library, ISP.SISPMACS, into SCLM01.PROJDEFS.SOURCE:
   - FLM@COBE — SCLM language definition for Enterprise COBOL
   - FLM@HLAS — SCLM language definition for z/OS High Level Assembler
   - FLM@L370 — SCLM language definition for z/OS Binder (Linkage Editor)
   - FLM@PLIE — SCLM language definition for Enterprise PL/I
5. Create and execute JCL member SCLM01.PROJDEFS.SOURCE(ALLOCDS) to allocate project partition data sets for SCLM01. You must customize this JCL to execute in your environment.

   Note: Alternatively, you can allocate project partition data sets via the ISPF data set utility. Use data set attributes as specified in the ALLOCDS member.

   See Appendix A-1, “SCLM01.PROJDEFS.SOURCE(ALLOCDS)” on page 691 for the listing of the ALLOCDS member.
6. Create and execute JCL member SCLM01.PROJDEFS.SOURCE(ALLOCVER) to allocate versioning partition data sets (PDSs) for SCLM01. The JCL allocates versioning PDSs for all groups and types being versioned, namely TEST and PROD groups and all editable types, ARCHDEF, SOURCE, COPYLIB, PLINCL, and MACROS. You must customize this JCL to execute in your environment.

   Note: Alternatively, you can allocate versioning partition data sets via the ISPF data set utility. Use data set attributes as specified in the ALLOCVER member.

See Appendix A-2, “SCLM01.PROJDEFS.SOURCE(ALLOCVER)” on page 693 for the listing of the ALLOCVER member.

7. Change SCLM01.PROJDEFS.SOURCE members FLM@COBE, FLM@HLAS, FLM@L370 and FLM@PLIE, so that macro libraries, assembler/compiler libraries, and assembler/compiler options match the libraries and products in use at your location. The changes are specified in the initial comments of each macro member.

   Note: If you make changes to these members after Step 10 while installing this example project, then reassemble and relink the project definition SCLM01.PROJDEFS.SOURCE(SCLM01).

   See Appendix A-3, “SCLM01.PROJDEFS.SOURCE(FLM@ARCD)” on page 694 through Appendix A-8, “SCLM01.PROJDEFS.SOURCE(FLM@TEXT)” on page 703 for the listings of the FLM* members with customizations for the example project highlighted in bold. We include Appendix A-3 FLM@ARCD and A-8 FLM@TEXT only for reference purposes. They require no changes for the example project.

8. Create and execute JCL member SCLM01.PROJDEFS.SOURCE(SCLMACCT) to define the VSAM cluster for SCLM accounting information of the SCLM01 project. You must customize this JCL to execute in your environment.

   This job deletes, defines and initializes the VSAM cluster. Because the cluster does not exist the first time you submit the job, you receive a return code of 8 from the delete operation.

   Notes:

   - Alternatively, you can define the VSAM cluster via the ISPF VSAM utilities interface. Use VSAM cluster attributes as specified in the SCLMACCT member.
   - We recommend that you use the SCLMACCT JCL job to define the VSAM cluster, as it is considerably simpler than the ISPF VSAM utilities option.

   See Appendix A-9, “SCLM01.PROJDEFS.SOURCE(SCLMACCT)” on page 703 for the listing of the SCLMACCT member.

9. Create and execute JCL member SCLM01.PROJDEFS.SOURCE(SCLMVERS) to define the VSAM cluster for SCLM audit and versioning information of the SCLM01 project. You must customize this JCL to execute in your environment.

   This job deletes, defines, and initializes the VSAM cluster. Because the cluster does not exist the first time you submit the job, you receive a return code of 8 from the delete operation.

   Notes:

   - Alternatively, you can define the audit and versioning VSAM cluster via the ISPF VSAM utilities interface. Use VSAM cluster attributes as specified in the SCLMVERS member.
   - We recommend that you use the SCLMVERS JCL job to define the VSAM cluster, as it is considerably simpler than the ISPF VSAM utilities interface.

   See Appendix A-10, “SCLM01.PROJDEFS.SOURCE(SCLMVERS)” on page 704 for the listing of the SCLMVERS member.

10. Create the SCLM01 project definition from the example in A-11. You can start with the sample project FLM01PRJ in the ISPF sample library SISPSAMP, if you do not want to create it from scratch. Save the project definition in SCLM01.PROJDEFS.SOURCE(SCLM01)
11. Create the $ASMPROJ assemble and link edit JCL member from the example in A-12. You can start with the sample JCL member FLM02PRJ in the ISPF sample library SISPSAMP, if you do not want to create it from scratch. Update the assembler SYSLIB DD concatenation sequence of the JCL for the ISPF SISPMACS library at your installation. Save the JCL in SCLM01.PROJDEFS.SOURCE($ASMPROJ).
12. Assemble and link edit project definition SCLM01.PROJDEFS.SOURCE(SCLM01) by submitting JCL member SCLM01.PROJDEFS.SOURCE($ASMPROJ).

    This constructs the load module SCLM01.PROJDEFS.LOAD(SCLM01) that is executed by SCLM to control the project.

    Look at both assembler and linkage editor listings and confirm that no statements were flagged and that both steps completed without errors.

    Note: Alternatively, you can assemble and link edit SCLM01.PROJDEFS.SOURCE(SCLM01) using ISPF Foreground Assembler and ISPF Foreground Linkage Editor. If you choose this option, using the ISPF Data Set Utility, allocate the following partitioned data set with space in cylinders (1,5), with 10 directory blocks, and with record format FB, LRECL 80:

    `SCLM01.PROJDEFS.OBJ`

    This partitioned data set would contain the object code for the library structure as defined in the project definition. SCLM01.PROJDEFS.OBJ(SCLM01) is output of the foreground assembler and input to the foreground linkage editor.

    See Appendix A-11, “SCLM01.PROJDEFS.SOURCE(SCLM01)” on page 705 and Appendix A-12, “SCLM01.PROJDEFS.SOURCE($ASMPROJ)” on page 706 for the listings of the SCLM01 project definition and the $ASMPROJ JCL members, respectively.

SCLM sample project utility

You can also define a similar sample project using the SCLM sample project utility (ISPF Option 10.7). Refer to the z/OS V1R8.0 ISPF SCLM Guide and Reference for more information.

1.3.2 Understanding the sample project definition

Let us now try to make sense of our sample project definition. Since the project administrator is responsible to maintain the CM environment of the SCLM project, it is important to acquire an in-depth understanding of macros and other components of the project definition.

Project definition macros

The sample project definition uses the following SCLM macros:

- FLMABEG: Initializes the project definition by defining the project name as SCLM01.
- FLMTYPE: Defines each type. We use the following type values in this sample project:
  - ARCHDEF: architecture definitions
  - SOURCE: source code
  - COPYLIB: COBOL copy code
  - PLINCL: PL/I includes

  - MACROS: Assembler macros
  - LIST: listings from compilers and assemblers
  - OBJ: object code
  - LMAP: load module maps
  - LOAD: executable load modules
  - PROCLIB: JCL procedures
  - JOBLIB: JCL jobs

  We have selected type names that are most suitable for the sample project. You should choose SCLM type names that best describe libraries of your project. See topic 1.5, “Choosing your SCLM types” on page 15 for a detail discussion of SCLM project types.

- FLMGROUP: Defines each group. The PROMOTE keyword defines the library structure. In the sample project, DEV1 and DEV2 are promoted to TEST and TEST is promoted to prod. See topic 1.4, “Defining your hierarchy ” on page 10 for an in-depth discussion of SCLM project groups.
- FLMCNTRL: Identifies the default VSAM data sets for the project. The VSAM data sets store library control information about the members in the project hierarchy.
- COPY: Identifies members to be copied into the project definition. We use it to include language definition macros into the sample project definition. You can also maintain other SCLM project components, such as types, groups, project controls, and so on, in separate PDS members, and include them into the project definition.
- FLMAEND: Ends the project definition.

Language definitions

The sample project uses the following SCLM language definitions:

- FLM@ARCD: ARCHDEF (Architecture) language definition
- FLM@COBE: COBE (Enterprise COBOL) language definition
- FLM@HLAS: HLAS (high-level assembler) language definition
- FLM@L370: LE370 (z/OS binder/linkage editor) language definition
- FLM@PLIE: PLIE (Enterprise PL/I) language definition
- FLM@TEXT: TEXT (Untranslated Text) language definition

These language definitions use the following SCLM macros:

- FLMSYSLB: This macro can be used to define a set of external libraries that contain project and/or system macros or includes.
- FLMLANGL: This macro specifies the language identifier.
- FLMTRNSL: This macro is used once for each translator to be invoked for a language.
- Translators are programs, CLIST or REXX™ called during user actions such as SAVE, EDIT, BUILD, PROMOTE, or DELETE.

The parse translator is invoked on an SCLM SAVE action when the keyword FUNCTN specifies PARSE. The parse translator can store statistics (for example, lines-of-code counts) and dependency information (for example, includes and copy statements). The programs listed in the sample do come with SCLM.

The build translator is invoked on an SCLM BUILD action when the keyword FUNCTN specifies BUILD. For example, in FLM@L370, the linkage editor IEWL is invoked. The build fails unless the return code is equal to, or less than, the value specified by the keyword GOODRC (0 in this example). The programs called are either load module, CLIST, or REXX, depending on the build scope. Some build translator programs come with SCLM, most of them are part of z/OS or extra products needed during a build.

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

The FLMTRNSL, FLMALLOC, and FLMCPYLB macros can be compared to the JCL elements, EXEC, and DD statements. More information can be found in chapter 5 of the *SCLM Guide and Reference*.

To summarize, the project definition specifies the names of the partitioned data sets used by the project (for example, SCLM01.DEV1.SOURCE), the library structure for the groups (for example, DEV1 members are promoted to TEST), and the languages to be used (for example, architecture definition, ASM, PL/I, COBOL, and link-edit).

1.4 Defining your hierarchy

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

1.4.1 Rules governing SCLM project hierarchies

The following rules govern the creation of hierarchies:

- Each group can have no more than one parent.
- Each parent group can have multiple child groups promoting into it.
- There is no restriction on the total number of groups a hierarchy can have besides the total allocated PDS in a build may not exceed 123.
- Each hierarchy has one root group, the topmost group.
- It is possible to have more than one hierarchy defined for one project.
- Defining no more than four layers makes it easier to use ISPF tools on the SCLM-controlled members.

1.4.2 Project hierarchy sample

It is usually easier to draw a diagram of your hierarchy, to help you visualize what the hierarchy looks like. Use the preceding rules and the requirements of your project to draw your hierarchy and label each group.

The following section shows a sample hierarchy. It is set up based on the development phases potential projects might use. You can create hierarchies other than one presented here. As a project evolves, the requirements that the project has on the hierarchy will change. With SCLM, you can change the hierarchy to meet the needs of the project.

Figure 1-1 shows the hierarchy set up for the SCLM01 sample project. It uses three layers; development, test, and production (prod), as follows:

- The development layer promotes to the test layer.
- The test layer promotes to the production layer.
- The development layer is composed of the groups DEV1 and DEV2.
- You can assign these groups, DEV1 and DEV2, to two separate developers, two different teams, or for parallel development within a single team.

Archived

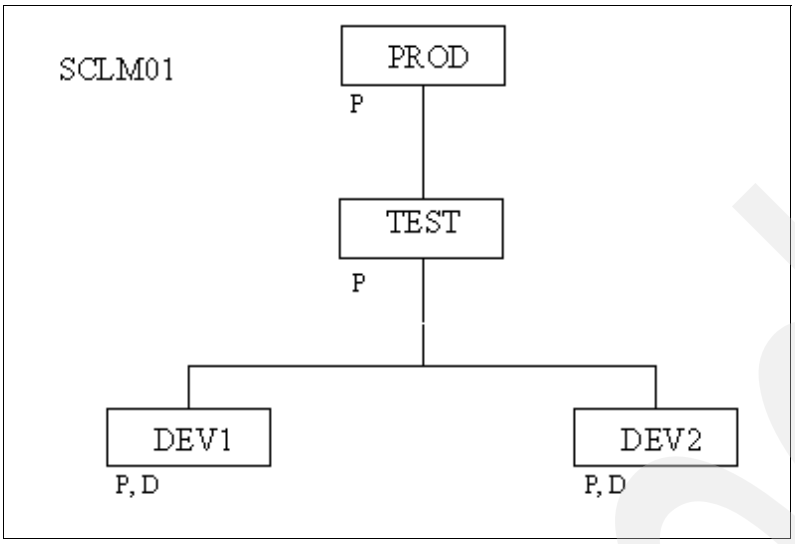


Figure 1-1 SCLM01 Hierarchy

The SCLM hierarchy is defined with the following SCLM macros:

```
DEV1     FLMGROUP AC=(P,D),KEY=Y,PROMOTE=TEST
DEV2     FLMGROUP AC=(P,D),KEY=Y,PROMOTE=TEST
TEST     FLMGROUP AC=(P),KEY=Y,PROMOTE=PROD
PROD     FLMGROUP AC=(P),KEY=Y
```

AC=(P,D) defines the authorization codes for the development group. An authorization code is an identifier used by SCLM to control authority to update and promote members within a hierarchy. Defining two authorization codes for DEV1 and DEV2 allows for parallel development. That is, it allows a same named member to be edited into both DEV1 and DEV2, one with an authorization code of P and one with an authorization code of D.

Each SCLM group must have at least one authorization code. To promote from one group to the next higher group, each must have one common authorization code. All members in the main hierarchy for our sample project have an authorization code of P, which allows them to be drawn down to DEV1 or DEV2 and re-promoted back to PROD. If a member has been drawn down to DEV1 with the default authorization code of P and you want to edit a same named member in DEV2, you must edit it using an authorization code of D. This is because only one member at a time can be edited in a set of parallel development groups using the default or promotable authorization code.

By parallel we mean multiple development groups promoting into a common group. In our example the common group is TEST. If edited using the D authorization code, the DEV2 version of the member will not be promotable. You can later merge its changes into the promotable version or take steps to make it promotable. The procedure to make it promotable will be covered in Chapter 6, “Using SCLM edit” on page 153.

1.4.3 Establishing authorization codes

Let us now examine in detail how you as the project administrator can support parallel development and emergency maintenance within your SCLM project. SCLM supports configuration management of parallel development and emergency maintenance through authorization codes.

Authorization codes control movement of data within the project hierarchy. They restrict the draw down and promotion of members (application components) to certain groups within the hierarchy.

In regard to authorization codes, here are some rules that you should be aware of:

- Authorization codes can be up to 8 characters and cannot contain commas.
- Authorization codes work only on editable types such as source, not on build outputs.
- Groups can have any number of authorization codes assigned to them.
- You must assign an authorization code to each editable member, when you register it with SCLM.
- In order to put (draw-down) a member into a group, the authorization code of that member must match one of the authorization codes that have been assigned to the group.
- If no authorization codes exist for a group, SCLM does not permit draw-down or promotion of members to this group.
- When you promote a member from one group to the next, the member retains its authorization code. Thus, the group being promoted into and the group being promoted from must have a matching authorization code.
- If, as a result of a promote, you replace an older version of the module, SCLM does not retain the authorization code assigned to that older version.

Define at least one authorization code for your project. Assign authorization code(s) to each SCLM group. A single code is generally sufficient for simple situations. It does allow each member under SCLM control to be drawn down to any development group and be promoted to the top of the hierarchy.

If you require tighter restrictions on movement of members for your project, you must identify those situations and define additional authorization codes.

Using the diagram you drew earlier in this section, examine the flow of members and determine if you require any restrictions on movement of members. Label each group with at least one authorization code.

An example of multiple authorization codes

One example of when you can use multiple authorization codes is when your project has multiple subsystems being developed in different legs of the hierarchy and you need to ensure that members of these subsystems do not get mixed in the development groups in the hierarchy legs. You set up authorization codes to prevent the members of one subsystem from being drawn down into development groups of another subsystem.

Figure 1-2 shows a simple hierarchy with four groups: PROD, TEST, DEV1 and DEV2. The group PROD has been assigned only one authorization code: DEV. Group TEST has two authorization codes: DEV and TESTONLY. Three authorization codes (DEV, PROTO, and TESTONLY) have been assigned to DEV1. Group DEV2 has DEV and L0 as its authorization codes.

This hierarchy is defined with the following SCLM macros:

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

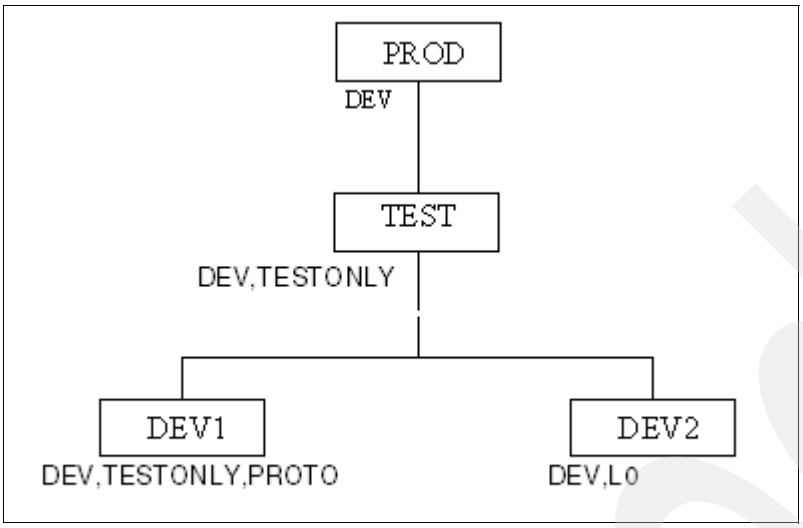


Figure 1-2 Hierarchy with multiple authorization codes

In Figure 1-2 the following relationships exist:

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

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

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

SCLM allows you to have two members of the same name and type residing in two different development groups (such as DEV1 and DEV2 in Figure 1-2) under certain conditions. Each of those members has an authorization code assigned to it. Those codes, along with the authorization codes assigned to the higher groups in the hierarchy, determine how far up the hierarchy each of those members can be promoted. If the two promotion paths do not intersect, SCLM lets you edit those members in those groups. However, if there is at least one group through which both members can be promoted, changes made to one member would be lost when the other member is promoted. In that case, SCLM does not let you edit the members in those groups.

Archived

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

Supporting parallel development and emergency maintenance

You can use the information in the previous section to set up a project in which you can make enhancements to what you have in production (development), while being able to make quick fixes to production modules (emergency maintenance). A simple hierarchy to illustrate this is shown in Figure 1-3 and is defined with the following SCLM macros:

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

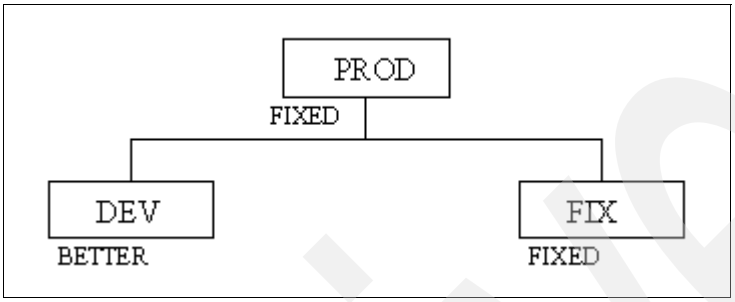


Figure 1-3 Hierarchy for normal and emergency development

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

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

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

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

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

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

Archived

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

Keep in mind that although authorization codes can be used to restrict promotion paths, they do not provide security against modifications to SCLM-controlled data made outside of the SCLM environment. You should use a security product such as Resource Access Control Facility (RACF®), which is part of the z/OS Security Server or its functional equivalent for that purpose.

1.5 Choosing your SCLM types

This step identifies the types of data required by the applications under development for your project. Some examples of the types of data used are source code, object modules, load modules, and source listings.

Determine the number of types you need based on the data you want to maintain for the project. For example, if you want to maintain compiler listings, a listing type is necessary. At a minimum, you should use four types to produce executable code:

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

Here are a few important facts about SCLM types:

- SCLM supports the same kind of data supported by MVS partitioned data sets (PDSs) and partitioned data sets extended (PDSEs).
- If your application source code resides in PDSs or PDSEs and if it follows the three-level naming convention (such as project.group.type), you can reuse the third-level data set qualifiers of your application for SCLM types.
- Data with different formats (such as objects and listings) must reside in their own separate SCLM types.
- Editable (such as source) and non-editable (such as object) members must reside in separate SCLM types, regardless of whether they have the same format or not.
- SCLM allows you to store members of different languages in the same type, so long as they have the same format, for example, RECFM=FB, LRECL=80.
- SCLM requires several additional types which are generally not present in the existing environment, for example, ARCHDEF, DB2CLIST, DB2OUT.
- SCLM does not require allocation of data sets for all SCLM types (and SCLM groups), thus providing flexibility to manage the storage requirement for your project. The next section discusses this feature in detail.

SCLM provides code integrity between all kinds of related data. Hence, we recommend that you take advantage of this capability. For example, define compile and linkage editor listings as derived or non-editable types for their counterpart source type, and keep compile and link edit listings synchronized with your source code and load modules. They come in handy for speedy problem resolution and for software quality assurance.

You can use either a single source type for all languages in the project, or a separate type for each language. You can also use a single type for the application include or copy code or a separate type for each language specific include or copy library.

Archived

We have observed that most customers like to use a single source type for all languages, because it minimizes project management effort by reducing the number of application libraries under SCLM control. On the other hand, we have not observed such a near unanimity in the choice of single versus multiple language specific types for the application include or copy code.

SCLM offers two ways to refer to include and copy code:

- The EXTEND=extended_type parameter of the FLMTYPE project definition macro is an 8-character name that can be used as an alternate type when resolving include dependencies.
- The FLMINCLS language definition macro associates include sets with types in the project hierarchy. This association is then used to determine the location of include members within the project. Each language definition can use this macro to define multiple include sets and each include set can be used throughout the language definition as needed.

If you use a single type for include or copy code, you can use the Extend parameter of the FLMTYPE project definition macro or the FLMINCLS macro. The Extend parameter can refer to only one include or copy type. If you use a separate type for each include or copy library and you store your source code in a single source type, you must use the SCLM include set specification coded with the FLMINCLS macro.

We recommend that you choose the FLMINCLS include set specification over the FLMTYPE Extend parameter, because FLMINCLS provides better flexibility than EXTEND.

Use a single type for each unique format or layout of build output or derived members to simplify management of your environment. For example, we have used a single LOAD type for all load modules and a single LIST type for all compiler/assembler listings in the SCLM01 project.

The SCLM01 sample project uses a single source type for all languages used in the project, and separate types for assembler macro, COBOL copy and PL/I include members, MACROS, COPYLIB and PLINCL, respectively.

1.5.1 Deferred Data Set Allocation

SCLM offers a feature called Deferred Data Set Allocation. This feature helps you to manage storage requirements of your SCLM project.

According to this feature, you need to allocate a data set for a particular SCLM type and group combination, only if you plan to use that data set. In other words, you need not allocate data sets that will not be used for a particular type and group.

Here are a couple of simple examples of how you can take advantage of this feature:

- If a developer is responsible for source code but not panels, you can leave the data set for the type containing panels unallocated for that developer's group.
- If the project hierarchy uses an include or copy type using the EXTEND or FLMINCLS functionality and the hierarchy design specifies that the source code for the include or copy type is always at a higher group, you can leave the data set for the include or copy type unallocated for the lower group.

Archived

To illustrate the above point, consider a project definition with the FLMTYPE macro written as follows:

Example with EXTEND Parameter:

```
MACROS     FLMTYPE
SOURCE     FLMTYPE  EXTEND=MACROS
```

Example with FLMINCLS Macro:

```
MACROS     FLMTYPE
SOURCE     FLMTYPE
.
.
.
           FLMINCLS  TYPES=(SOURCE,MACROS)
```

In this situation, the type MACROS can contain members referenced by members in the SOURCE type. However, if the source code in type MACROS will always be at a higher layer (group) in the hierarchy (for example, PROD in the SCLM01 project), you do not need to allocate data sets for type MACROS below the PROD layer in the main hierarchy.

1.6 Data set naming conventions

The naming convention of project partitioned data sets is very closely related with your choice of the project hierarchy (groups) and SCLM types.

A typical SCLM project definition uses the default naming convention of project.group.type for project partitioned data sets. However, if your project cannot use the default naming convention, you can specify an alternate naming convention either for all project partitioned data sets or for project partitioned data sets associated with individual groups in the hierarchy.

If you inherit an existing project with a naming convention other than project.group.type, the existing data sets can be used in conjunction with SCLM's flexible data set naming capability. The only prerequisite for using this capability is that the project must use the standard MVS data set naming conventions. For example, it is possible to use four or five qualifiers in the data set names instead of the three qualifiers that are used by the standard SCLM naming convention as long as one of the qualifiers is the SCLM type. The only exception is the SCLM project dataset, which must use the project.PROJDEFS.LOAD naming convention.

The next section provides additional information on using the flexible naming capability.

1.6.1 Flexible naming of project partitioned data sets

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

Archived

Note: This mapping is only maintained while users are executing SCLM functions. If ISPF utilities are used on data controlled by SCLM, the users should know the mapping between the PROJECT.GROUP.TYPE name and the fully qualified data set name. Use the SCLM accounting information to find the fully qualified names of project data sets for members stored in a particular group and type.

The data set names are defined in the project definition with the FLMCNTRL and FLMALTC macros. Each macro has a DSNAME parameter that allows the project manager to specify the data set names for the entire project or for individual groups. The FLMCNTRL macro defines the data set names for the entire project. The FLMALTC macro defines the data set names on a group-by-group basis. The project definition may be easier to define and understand if you use FLMALTC to define the data set names for each group in the hierarchy and use the FLMCNTRL macro to only define project elements that all global to the entire project.

The sample project SCLM01 described in section 1.1 uses the standard SCLM naming convention for both project and versioning partition data sets. A modified version of the SCLM01 sample project is shown in Appendix A-13, “Flexible Naming of Project Partitioned Data Sets” on page 707 that uses the flexible naming convention. The modified SCLM01 project uses the flexible naming convention of the pattern:

```
component_name_1.component_name_2.@@FLMPRJ.@@FLMGRP.@@FLMTYP
```

The DEVCNTL FLMALTC macro defines an alternate accounting database and data set names to be used by the DEV1 and DEV2 groups that references this macro. The partitioned data sets associated with the DEVx groups have the naming convention:

- 'ISPFSCLM.DEV.SCLM01.DEVx.type'.

The TESTCNTL FLMALTC macro defines an accounting database and data set names to be used by the TEST group that references this macro. The naming convention used for the partitioned data sets associated with the TEST group is:

- 'ISPFSCLM.TEST. SCLM01.TEST.type'.

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

The SCLM variable @@FLMDSN is created from the value of the DSNAME parameter. Therefore, if the data set name pattern is:

```
component_name_1.component_name_2.@@FLMPRJ.@@FLMGRP.@@FLMTYP
```

Then the value of @@FLMDSN will be:

```
component_name_1.component_name_2.@@FLMPRJ.@@FLMGRP.@@FLMTYP.
```

The versioning partitioned data sets can also use a naming convention other than SCLM's default naming convention. The VERPDS parameter on the FLMCNTRL and FLMALTC macros is used to specify the name of the versioning partitioned data sets. SCLM uses a

default of @@FLMDSN.VERSION for the names of the versioning data sets. If a pattern other than the default is used, the variables @@FLMGRP and @@FLMTYP must be part of the data set name pattern. Using two variables minimizes the risk that the versioning data set names associated with different groups are the same, and prevents data from being overwritten. The modified sample SCLM01 project uses the default @@FLMDSN.VERSION names for versioning data sets.

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

There is also an alternative method of implementing a flexible naming convention and it does not depend upon SCLM. It is the IDCAMS DEFINE ALIAS facility, also known as data set alias. We have observed that some customers prefer data set alias instead of SCLM's standard flexible naming convention method.

Here is an example of how you can implement flexible naming convention using data set alias:

```
DEFINE ALIAS (NAME('<project>.<group>.HCSS')                    -
RELATE ('<project>.<group>.HELP.CSS'))
```

Further discussion of data set alias is outside the scope of this redbook. Refer to the DFSMS/MVS® documentation for additional information on this topic.

1.7 Choosing your SCLM languages

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

IBM provides examples of SCLM language definitions for many commonly used languages such as COBOL, PL/I, assembler, C and Java. See Appendix A-14, "Language Definitions Supplied with z/OS SCLM" on page 708 for the list of all example language definitions included with z/OS SCLM.

To define the language definitions:

- Determine what languages are used in your project.
- Copy the appropriate example language definitions to the project.PROJDEFS.SOURCE data set. It is recommended that you name the member containing the language definition the same as that specified in the LANG= option of the FLMLANG macro.
- Modify the language definitions according to your specific requirements. See the next section for a detail discussion of modifying language definitions.

1.7.1 Modifying example language definitions

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

FLMSYSLB Use this macro to define data sets that contain system, project, or language dependencies that are referenced by SCLM members but are not in the SCLM hierarchy themselves. Examples are system macros for Assembler programs and compiler-supplied include files for C programs.

FLMLANGL Use this macro to define the language name to SCLM.

FLMINCLS Use this macro to associate sets of includes found during the parse of a member with the types in the project definition that contain those includes. FLMALLOC macros then reference this macro to allocate the include libraries for build translators. The FLMINCLS macro can be used multiple times for each language, but each FLMINCLS macro must have a unique name within the language and be associated with at least one FLMALLOC macro. This helps ensure that the includes that are found by build are the same ones found by the translators.

FLMLRBLD Use this macro to tell SCLM to automatically rebuild members with this language after they are promoted into the listed groups.

FLMTRNSL Use this macro to define a translator for a language. It can be used multiple times for a language.

FLMTOPTS Use this macro to vary the options passed to a build translator based on the group where the build is taking place. Options can be appended to the existing options or replace the options completely. FLMTOPTS macros must follow an FLMTRNSL macro with FUNCTN=BUILD.

FLMTCOND Use this macro to specify conditional execution of BUILD translator. Part of the specification can include examination of return codes from previous BUILD translators in the language definition.

FLMALLOC Use this macro for each data set allocation required by a translator. If you are using a ddname substitution list, specify an FLMALLOC macro for each ddname in the correct order. If not, determine the ddnames that are needed by the translator and specify an FLMALLOC macro for each ddname.

FLMCPYLB Use this macro to identify data sets to be concatenated to a ddname. The data sets must be preallocated. The FLMCPYLB data sets are used as input to the Parse and other translators.

For each language, take the following actions as necessary:

- Specify data sets containing dependencies that are not to be tracked, such as assembler system macros (macro FLMSYSLB).
- Specify the maximum number of includes, change codes, user data records, compilation units, and external dependencies expected in a source member (macro FLMLANGL; keyword BUFSIZE).
- Specify the version for the language (macro FLMLANGL: keyword: VERSION). This change will also trigger rebuilds of members with this language even if nothing else has changed. So each time you make essential changes to the language change the version. A proposed name is Dyymmdd, where yymmdd is the year month day format, e g. D070326 which says it was changed on March 26, 2007.
- Determine if ddname substitution is needed for the translator. This information can be found in the translator documentation. Adjust the PORDER parameter on the FLMTRNSL macro as needed.
- Verify translator load module names and load data sets for accuracy (macro FLMTRNSL; keywords COMPILE, DSNAME, and TASKLIB).
- Adjust translator return codes to project requirements if nonzero return codes are acceptable (macro FLMTRNSL; keyword GOODRC).
- Update default translator options, as required (macro FLMTRNSL; keyword OPTIONS).
- Verify translator version information (macro FLMTRNSL; keyword VERSION).
- Specify output listings (macro FLMALLOC; keyword PRINT).

- Specify output default types (macro FLMALLOC; keyword DFLTTYP) to match the FLMTYPE type specified in the project definition.
- Verify that system libraries are being allocated for build translators. Either specify ALCSYSLB=Y on the FLMLANGL macro or ensure that the data sets from FLMSYSLB macros are specified on FLMCPYLB macros following IOTYPE=I allocations.
- Specify the include sets for the language to use. You must specify all the include-sets returned by the parser for the language. If you add a new FLMINCLS macro, ensure that it is referenced by at least one FLMALLOC of a build translator. If you remove an FLMINCLS macro, update any FLMALLOC macros that reference it, ensuring that no member's accounting data contains references to that include set.

Example A-3 on page 694 through Example A-8 on page 703, referenced in 1.3.1, “Steps to set up the sample project” on page 6, show language definitions supplied by SCLM and tailored for the SCLM01 sample project. To help understand language definition macros more thoroughly, let us examine the FLM@COBE language definition in Example A-4 on page 695 in detail.

In FLM@COBE, the FLMLANGL macro defines the COBE (Enterprise COBOL) language to SCLM. The FLMTRNSL parameters specify particular information about the compiler:

- The name of the compiler: Enterprise COBOL.
- The name of the compiler load module:IGYCRCTL.
- The version of the compiler: 3.4.0.
- The compiler options: XREF,LIB,APOST,NODYNAM,LIST,NONUMBER,NOSEQ

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

You can include language definitions into the project definition, either by placing them directly into the project definition or having them copied into the project definition when the project definition is assembled. It is easier to maintain the project definition if you keep each language definition in a separate member and copy it into the project definition when the project definition is assembled. The SCLM01 sample project definition uses this method of including the language definitions.

1.8 Project controls

Project controls define how SCLM operates for the project. The project control options dictate SCLM processing for the project. You have the following project control options at your disposal.

Primary accounting data set

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

The ACCT control option specifies the name of the primary accounting data set. The data set must be a VSAM KSDS cluster. The default accounting cluster name is project.ACCOUNT.FILE, where project is the 8-character name for the project.

The SCLM01 sample project definition listed in Figure 1-11 uses the primary accounting data set, SCLM01.ACCOUNT.FILE.

Secondary accounting data set

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

The ACCT2 control option specifies the name of the secondary accounting data set for the project. The data set must be a VSAM KSDS cluster. If a severe problem occurs with the primary accounting data set, you could use this backup data set to restore the primary accounting information. If you use this option, additional VSAM updates to the secondary accounting data set take place and can affect SCLM's performance.

The SCLM01 sample project definition listed in Appendix A-11 does not use a secondary accounting data set.

Audit control data set

The audit control data set contains audit information about changes to the software components in the project for groups that have auditing enabled. The audit control data sets also support member versioning when that capability has been enabled. Enabling versioning is discussed later in this chapter. Specify the audit control data sets, if (and only if) you plan to use SCLM's auditing and versioning capability. The secondary audit control data set is a backup of the information in the audit control data set.

The VERS control option specifies the primary audit control data set. The VERS2 control option specifies the secondary audit control data set that is a backup for the primary audit control data set. Both must be VSAM KSDS clusters.

When using the auditing capability, the secondary audit control data set is optional. Use the FLMALTC macro to specify different audit control data sets on specific groups.

The SCLM01 sample project definition listed in Appendix A-11 uses the primary audit control data set, SCLM01.VERSION.FILE.

Export accounting data set

The export accounting data set contains accounting information that has been exported from the accounting data set. The EXPACCT control option specifies the name of the export accounting data set. The data set must be a VSAM KSDS cluster. The following variables can be used in specifying the name of the export accounting data set name:

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

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

The SCLM01 sample project definition listed in Example A-11 on page 705 does not specify an export accounting data set. If you need to define an export data sets for your project, use the model code segment provided in Example 1-1.

Example 1-1 Model Project Definition Code Segment for Export Control Data Sets

```
        FLMCNTRL ACCT=SCLM01.ACCOUNT.FILE,                          C
             EXPACCT=SCLM01.EXPORT.ACCOUNT.FILE
SAMPLE  FLMALTC  ACCT=SCLM01.ACCOUNT.SAMPLE,                        C
             EXPACCT=SCLM01.EXPORT.ACCOUNT.SAMPLE
```

1.8.1 Data set naming conventions

Use the DSNAME control option on the FLMCNTRL macro to specify a naming convention other than the SCLM default for the project partitioned data sets. You can specify the naming convention for all data sets in the hierarchy with the DSNAME option. Use the FLMALTC macro, if the naming convention of the project partitioned data sets is different for specific groups. The FLMALTC macro changes naming convention for the data sets associated with specific groups.

Use the VERPDS control option on the FLMCNTRL macro to specify the names of partitioned data sets that will contain the audit and versioning data for a project. If the names of the versioning partitioned data sets will be different for specific groups, the FLMALTC macro must be used to associate the names of the versioning partitioned data sets with the specific groups.

The following variables can be used in specifying both project and versioning partitioned data set names:

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

For more information about modifying the naming convention for project and versioning partitioned data sets, see 1.6.1, “Flexible naming of project partitioned data sets” on page 17".

1.8.2 SCLM temporary data set allocation

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

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

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

Some of the temporary data sets used by versioning will use the VIO unit as long as the size of the temporary data set to be allocated is less than or equal to the MAXVIO value. The default value for MAXVIO is 5000, and the maximum allowable value is 2147483647. Specify a relatively large value, such as 999999, in order to ensure that SCLM temporary data sets are allocated using the VIO unit. If SCLM functions fail for lack of memory (S80A ABEND or S878 ABENDs), try reducing this value. The SCLM01 sample project definition listed in Appendix A-11 uses the MAXVIO value of 999999.

Temporary data set allocations that fail to meet any of the preceding conditions will be allocated using the unit specified via the DASDUNIT parameter on the FLMCNTRL macro. The size of the temporary data sets allocated for translators is determined by the attributes specified on the FLMALLOC macros in the language definition. The size of the temporary data sets used by versioning is based on the attributes of the source data set being versioned.

Archive

1.8.3 Member level locking

Member level locking is a new feature introduced in SCLM for z/OS V1R8. It allows SCLM administrators to stop users from modifying members that belong to other users. To implement member level locking, add MEMLOCK=Y, CONTROL, and ADMINID parameters to the FLMCNTRL macro in the project definition, as showed in Appendix A-11.

The SCLM Admin option is used to add and delete SCLM administrators for a project. SCLM administrators are allowed to maintain any locked members and to transfer ownership of a member to another user ID. The SCLM Admin option is only available to users who are already SCLM administrators. When member level locking is first enabled, the only user who can access this option is the user whose ID is specified in the ADMINID parameter on the FLMCNTRL macro in the project definition.

When member level locking is in effect, you can edit a member if any of the following conditions is true:

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

If another user needs to modify the member, either the SCLM administrator or the user who last updated the member (Change User ID on the accounting record) can use the Transfer option in option 3.1. See the "Transfer ownership" topic in the *z/OS ISPF SCLM Guide and Reference*. Member Level Locking is covered in more detail in Chapter 10.

1.8.4 Miscellaneous control options

SCLM also offers you several other control options to further customize your project setup. Use them as required for your project.

VSAM record level sharing

The VSAMRLS control option indicates whether VSAM Record Level Sharing should be used. VSAMRLS=YES supports sharing the VSAM data sets across systems in a sysplex environment. It requires the Coupling Facility Hardware and a VSAM cluster allocated with proper characteristics. The default value is NO. The SCLM01 sample project does not use VSAM Record Level Sharing.

Refer to the DFSMS/MVS® documentation for additional information about the hardware and software requirements to support VSAM RLS.

Maximum lines per page

Use the MAXLINE control option to specify the maximum lines per page for all SCLM-generated reports. The default is 60. The minimum number of lines per page is 35. The SCLM01 sample project definition uses the default MAXLINE of 60.

Translator option override

The OPTOVER control option allows you to keep developers from overriding project-defined translator options. If you specify Y, developers can override the translator options for any of the languages by using the PARM statement in the architecture members. If you specify N, SCLM uses only translator options you specify in the language definition for the translators. Specifying N also overrides the OPTFLAG parameter, which allows option override by the translator. The default for the OPTOVER control option is Y. The SCLM01 sample project definition uses the default OPTOVER option Y.

1.8.5 Common project control macros

To summarize, we show here several project control macros in the order that they are usually used in the project definition:

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

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

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

1.9 Setting up your accounting and project data sets

SCLM uses partitioned data sets (PDSs) or partitioned data sets Extended (PDSEs), to store members; both source components and generated outputs. SCLM uses VSAM KSDS clusters to store accounting information to track and control the members within the project hierarchy.

The project partitioned data sets store the user application data. They are organized into a hierarchy and controlled by the project definition. Allocate the project partitioned data sets using either the ISPF Data Set Utility (option 3.2) or a JCL process (such as the IEFBR14 utility). Use the information in this step to determine the names, number, and physical characteristics of the project partitioned data sets.

The project VSAM data sets store the SCLM control data for source and derived objects. The accounting data set contains information about the software components in the project. The secondary accounting data set is a backup of the information in the accounting data set. The export accounting data set contains accounting information that has been exported from the accounting data set. At least one accounting data set is required for a project.

1.9.1 Creating the accounting data sets

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

You can also create an optional secondary accounting data set. The secondary accounting data set is a backup for the primary accounting data set. It allows for the restoration of accounting information if the primary data set becomes corrupted, for example due to a disk failure. The secondary accounting data set must have a unique data set name, and should be stored on a different volume than the primary accounting data set.

Be aware that, if you use a secondary accounting data set, the performance of SCLM will be affected, because updates are made to both the primary and secondary data sets. Compare information in both data sets periodically to ensure the integrity of the accounting information.

Create both the primary and secondary accounting data sets the same way, as a VSAM cluster, using the IDCAMS define cluster utility. Create additional accounting data sets, if you need to maintain accounting information for different groups in separate accounting data sets.

Example A-9 on page 703, discussed in 1.3.1, “Steps to set up the sample project” on page 6, shows the JCL, called SCLM01.PROJDEFS.SOURCE(SCLMACCT), which is used to define the accounting data set supplied by SCLM and tailored for the SCLM01 sample project.

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

1.9.2 Creating the export data sets

The export control data sets are optional unless the export and import functions are used. Before using the EXPORT service, you must allocate and define an export accounting data set.

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

1.9.3 Creating the project partitioned data sets

SCLM supports standard z/OS PDSs and PDSEs, and does not restrict the format of project data sets. Allocate data sets of the same type with same attributes. Table 1-1 lists recommended data set attributes for some typical types.

Table 1-1 Recommended data set attributes for typical SCLM types

Type	DSORG	RECFM	LRECL	BLKSIZE
Source	PO	FB	80	0
Object	PO	FB	80	0
Load	PO	U	0	>=6144
Listings	PO	VB	137	0
Linkedit Maps	PO	FBM	121	0
Architecture definitions	PO	FB	80	0
Other Text	PO	FB	80	0

Note: Blocksize=0 ensures best performance by using the system-determined block size.

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

Archived

1.9.4 PDS versus PDSE considerations

SCLM supports both PDSs and PDSEs as project data sets. Hence, make your choice that best suits your project requirements.

Here are some points about PDSs and PDSEs that you might want to consider in making your choice:

- Both PDSs and PDSEs provide efficient access to libraries of related members, such as program source code, load modules, JCL, and many other types of software content.
- The size of a PDS directory is set at allocation time. If you need to store more entries than there is room in the directory, you have to allocate a new PDS with more directory blocks and copy all members from the old PDS into the new one. This means that when you allocate a PDS, you must calculate the amount of directory space you need.
- The directory of a PDSE grows dynamically as the data set expands and more members are stored into it. This saves time and effort for storage administrators and application developers.
- There is no mechanism in a PDS to reuse DASD space of deleted and replaced members. This wasted space is often called gas and must be removed periodically by compressing the PDS.
- There is no mechanism to stop the PDS directory from being overwritten if a program mistakenly opens it for sequential output. If this happens, the directory is destroyed, and all the members are lost. A PDSE protects the directory from accidental corruption.
- If you inadvertently add a member to a PDS whose DCB characteristics differ from those of the other members, you change the DCB attributes of the entire PDS, and all the old members become unusable. PDSEs have built-in protection to guard against such a situation.
- To compress a PDS, increase the amount of directory space or to change a member of a PDS, you need exclusive access to the entire PDS. This means that no other jobs can have access to the data set during an update. If the data set is always accessed for input by a system task that runs 24 hours a day, there is a problem. If you want to update the data set, you must either shut down the system task or turn off the exclusive locking control for this data set. Neither option is acceptable to many installations.
- A PDSE does not require compression, and offers a facility to control sharing of partitioned data sets at the member level, so that if you access one member to update it, you do not lock all the other members in the data set.
- A load library PDS (RECFM=U,LRECL=0) can store an executable as a load module, but a PDSE cannot. A PDSE stores an executable as a program object. You need to use the IEBCOPY utility to convert a load module to a program object. SCLM handles this conversion for you for SCLM controlled load libraries.
- In a multi-LPAR environment with shared DASD and with SCLM datasets defined as PDSEs, where developers and/or another tools, such as OPCA, are accessing these datasets concurrently from another LPAR, the PDSESHARING option must be set to EXTENDED. For PDSE use in this situation, PDSESHARING=EXTENDED is required to avoid 213-070 abends. The PDSESHARING setting can usually be found in SYS1.PARMLIB member IGDSMSxx. Global Resource Serialization (GRS) or its functionally equivalent product must also be available on the systems. See your Storage Management contact or systems programmer to obtain additional information.

1.9.5 Some recommendations

Start with one control data set, the primary accounting data set. Add more control data sets as the project evolves and you need more advanced SCLM capabilities.

While it is not required that the first qualifier of VSAM data sets match the project name, it makes project maintenance easier.

SCLM supports sharing of VSAM data sets across a sysplex environment through VSAM RLS. Refer to the DFSMS/MVS® documentation for hardware and software requirements to support VSAM RLS. Do not share VSAM data sets under any other condition. Accessing any of the VSAM data sets from multiple systems when VSAM RLS is not available can result in the corruption of data, system errors, or other integrity problems. To avoid these problems, the project manager must allocate VSAM data sets so that they cannot be accessed from multiple systems.

Back up all VSAM data sets periodically using the IDCAMS reproduction (REPRO) utility. This will reduce fragmentation and optimize the performance of your VSAM data sets.

Choose PDSEs over PDSs for project data sets, wherever feasible. At a minimum, always use PDSEs for load libraries allocated to online applications, such as CICS regions, IMS MPP and BMP regions, ADABAS/NATURAL, IDMS/ATSO, TSO/ISPF, and long running background address spaces, such as DB2, MQ Series, and so on.

1.10 Setting up your audit and versioning capability

This topic describes how you can set up and implement audit and versioning capability for your project.

The SCLM audit and versioning capability enables you to audit all application components controlled by SCLM and to create versions of editable components. The project administrator controls the audit and versioning capability through the FLMCNTRL, FLMALTC, and FLMATVER project definition macros. SCLM uses VSAM KSDS clusters to store audit control information of source and output components, and PDSs or PDSEs to store versions of source components within the project hierarchy.

1.10.1 Audit and versioning capability: Overview

The audit and version utility enables you to audit SCLM operations on SCLM-controlled members and create versions of editable members. Using the audit and version utility, you can view the audit information for a member, retrieve a version to a sequential or a partitioned data set outside SCLM, or to an SCLM-controlled development group, as well as to delete audit and version information from the database. See Chapter 9, “SCLM utilities” on page 185 for details of the audit and version utility.

Use the FLMCNTRL and FLMALTC project definition macros to specify audit control VSAM data sets and versioning partitioned data sets. Audit control data sets contain audit information about changes to the software components in the project for groups that have auditing enabled. Versioning partitioned data sets store prior versions of the user application data.The secondary audit control data set is a backup of the information in the audit control data set.

Use the FLMATVER project definition macro to enable the audit and version utility. Using the group and type defined in this macro, SCLM records information in the audit control data set each time a member's accounting information is created, updated, or deleted within that SCLM group. This information is a record that contains:

- The member's accounting information
- The type of operation
- The user ID of the user who performed the operation
- The date and time the operation occurred

Versioning partitioned data sets are organized into a hierarchy which matches that of project data sets, and controlled by the project definition. The version contains the information to recreate the member as it previously existed.

Note: Version information is captured each time an editable member or an output that is not record format U is created or updated, but not when it is deleted. Since versioning of output or derived members or components does not generally add much value to the software configuration management (SCM) process, most customers implement versioning to create versions of only editable components.

1.10.2 Setting up the project definition for audit and versioning capability

You can use several parameters of the FLMCNTRL and FLMALTC macros and all parameters of the FLMATVER macro of the project definition to set up the audit and versioning capability for your project. We introduce these parameters in detail below.

FLMCNTRL and FLMALTC macro parameters

Next we discuss the FLMCNTRL and FLMALTC macro prarmeters.

Primary audit control data set:

- Use the VERS parameter on the FLMCNTRL macro to specify the name of the primary audit control data set for your project. If the name of the primary audit control data set is different for a specific group, use the FLMALTC macro to associate the name of the primary audit control data set with the specific group. The default is no audit and versioning for the project.

Secondary audit control data set:

- Use the VERS2 parameter on the FLMCNTRL macro to specify the name of the secondary audit control data set for your project. If the name of the secondary audit control data set is different for a specific group, use the FLMALTC macro to associate the name of the secondary audit control data set with the specific group. Do not specify VERS2 without specifying VERS. If you do, an error occurs during assembly of the project definition.

Versioning partitioned data sets:

- Use the VERPDS parameter on the FLMCNTRL macro to specify the names of versioning partitioned data sets for your project. If the names of versioning partitioned data sets are different for specific groups, use the FLMALTC macro to associate the names of versioning partitioned data sets with the specific groups.
- Use the following variables in specifying versioning partitioned data set names: @@FLMPRJ, @@FLMGRP and @@FLMTYP or @@FLMDSN.
- The default for VERPDS on the FLMCNTRL macro is @@FLMDSN.VERSION. If you intend to use another VERPDS naming convention, either use the FLMALTC macro to

specify different VERPDS data sets for each group or use the @@FLMGRP variable in the VERPDS name on the FLMCNTRL macro. Either way, do ensure that VERPDS data sets are unique for each group and type in the project hierarchy. Failure to specify and allocate unique VERPDS data sets can result in difficulty retrieving versions.

- You can have only one VERPDS data set per group and type at a time. However, you can respecify the VERPDS data set name to control the size of the versioning partitioned data sets. If the VERS=primary audit control data set name remains the same, a pointer to the VERPDS that holds a particular version allows you to retrieve and delete versions of members, even if you have changed the name of the VERPDS data set.

Number of versions to keep:

- Use the VERCOUNT parameter on the FLMCNTRL macro to specify how many versions of a member to keep. The default value of zero, used in the SCLM01 example project indicates that all versions are kept. The number of versions specified using this parameter applies to all types that are versioned. You can override this value for specific groups and types using the VERCOUNT parameter on the FLMATVER (not FLMALTC) macro.
- Valid values are 0 and any integer value greater than or equal to 2. 1 is not a valid value, because that is already in the hierarchy. If you do specify a value of 2 or more, allocate a separate VERPDS for each group and type that has versioning enabled.

FLMATVER macro parameter

Next we discuss the FLMATVER macro parameter.

Group name:

- Use the GROUP parameter to specify the name of the group for which audit and/or version data is to be maintained. The group must be defined in the project using the FLMGROUP parameter. Use an asterisk (*) to indicate all groups.

Type name:

- Use the TYPE parameter to specify the name of the type for which audit and/or version data is to be maintained. The type must be defined in the project using the FLMTYPE macro. Use an asterisk (*) to indicate all types.
- We do not recommend using TYPE=* with VERSION=YES, because, even though audit information can be captured for all types, editable as well as noneditable, version information cannot be captured for non-editable types that are record format U, e.g. load modules. If you attempt to version data that is record format U, an error message is issued during SCLM processing.

Activate versioning:

- Use the VERSION parameter to activate or deactivate versioning. If you specify YES, both the versioning and auditing processes are active. If you specify NO which is also the default, versioning is not active; however, the audit process is active. Version data can be captured for any editable or non-editable (output) members that are not record format U. You cannot have versioning without auditing.

Number of versions to keep:

- Use the VERCOUNT parameter to specify how many versions of a member to keep for a specific group and type. The default value of zero, used in the SCLM01 example project indicates that all versions are kept. This value overrides the VERCOUNT parameter on the FLMCNTRL macro. See the previous section on FLMCNTRL and FLMALTC macro parameters for details.

Ignore sequence number differences:

- Use the SEQNUM parameter to indicate whether to ignore sequence number differences. If you specify STANDARD, STD, or COBOL, SCLM ignores sequence number differences when creating a version of a member. STANDARD or STD means ignore differences in the last eight columns of the data for fixed formats, and the first eight columns of the data for variable formats. In both cases the ignored columns are presumed to be standard sequence numbers. COBOL means ignore differences in the first six columns of the data, which are presumed to be COBOL sequence numbers. Omitting this parameter, or specifying NONE, indicates that all columns are to be treated as data.

Enable checksum verification on version retrieval

- Use the CHECKSUM parameter to indicate whether to enable checksum verification. The default value of YES enables checksum verification of versions on retrieval. The value of NO disables checksum verification.
- The checksum verification is a new feature of SCLM for z/OS V1R8. When you migrate from an earlier SCLM release in which SEQNUM support was not available, you might encounter message FLM39220 Return Code 34. If this happens, disable checksum verification (CHECKSUM=NO) and retry retrieval of the version to override the checksum verification failure. However, the validity of the retrieved version is not a sure thing. Hence, we recommend this procedure only for migration or for emergency use.

1.10.3 Creating the audit control data sets

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

You can create an optional secondary audit control data set. The secondary audit control data set is a backup for the primary audit control data set. It allows you to restore audit control information if the primary audit control data set is corrupted. Choose a unique name for this data set and put it on a different volume than the primary audit control data set.

Be aware that, if a secondary audit control data set is used, SCLM's performance will be affected, because updates are made to both the primary and secondary audit control data sets. Compare information in both data sets periodically to ensure the integrity of the audit control information.

Create both the primary and secondary audit control data sets the same way, as a VSAM cluster, using the IDCAMS define cluster utility . Create additional audit control data sets, if you need to maintain audit control information for different groups in separate audit control data sets.

Example A-10 on page 704 , referenced in 1.3.1, “Steps to set up the sample project” on page 6, shows the JCL used to define the audit control data set supplied by SCLM and tailored for the SCLM01 sample project. This JCL is called SCLM01.PROJDEFS.SOURCE(SCLMVERS).

Each audit control data set requires approximately one cylinder of 3390 DASD for every 100 partitioned data set members that SCLM controls. The space required varies depending on how much information SCLM will control. If you require additional space in the data set, modify the space parameter (shown as CYLINDERS in the example JCL).

1.10.4 Creating the versioning partitioned data sets

Allocate at least one versioning partitioned data set, If the versioning capability is going to be used. If you intend to use the VERCOUNT parameter on the FLMCNTRL macro to specify that two or more versions be maintained, you must specify at least one versioning partitioned data set for each group to be versioned. Otherwise, errors can occur during version retrieval. You can also choose to have a versioning partitioned data set associated with each “group.type” to be versioned. Table 1-2 lists recommended data set attributes for versioning partitioned data sets.

Table 1-2 Recommended Data Set Attributes for Versioning Partitioned Data Sets

TYPE	RECFM	LRECL	BLKSIZE
*.VERSION	VB	The larger of 259 and the source data set's LRECL + 4	At least LRECL + 4 Bytes. Use the optimal block size for your system (BLKSIZE=0).

Notes:

- The LRECL value must be at least 259 and must be 4 bytes more than the LRECL of the largest source data set to be versioned.
- The 4 bytes in the block size calculation are for MVS control information, specifically for the blocklength field. For example, with a blocking factor of 10 the block size would be calculated as (259 x 10) + 4 = 2594.

1.10.5 PDS versus PDSE considerations

SCLM has no special considerations that require the allocation of PDSs or PDSEs as version data sets. Hence, make a choice that best suits your project requirements. See section 1.9.4, “PDS versus PDSE considerations” on page 27 for some points about PDSs and PDSEs that you might want to consider in making your choice.

1.11 Setting up the project for package backout

This topic describes how you can set up and implement package backout. For further information, see the section “Implementing package backout” in Chapter 3 of the SCLM Guide and Reference.

Package backout enables users to quickly restore an executable environment, in an event of a critical production problem caused by a bad software release (package). Once the immediate problem has been resolved in the executable environment, users can apply corrections to the software release using the normal development process. They can either retrieve the version of the source corresponding to the backed out member into a development group for editing, or make changes in the existing copy of the source member in the hierarchy.

1.11.1 Package Backout utility: Overview

The SCLM Package Backout utility enables you to back up and recover non-editable types, such as object and load modules, using a backup group controlled within SCLM. The backout process restores an executable environment by promoting the previously backed up modules from the backup group. It does not recover editable types, such as Source code. Developers

must recover source code through versioning which is outside the scope of the Package Backout process.

The term "package" refers to an SCLM architecture member that is used during build and promote processes. This architecture member defines the modules and architecture members that are promoted using include or change code parameters.

The libraries that contain packages are determined by using the ISAPACK=Y flag on the FLMTYPE macro within the project definition. If an architecture member is promoted from a library which does not have an ISAPACK=Y flag then the package backout process will not be invoked and no modules will be backed up.

During package backout, the equivalent of normal promote processing is performed from the backup group, with both the Promote Copy and Purge phases. The Copy phase copies modules from the backup group to the target group. The Purge phase deletes the backed up modules. The promote process also allows DB2 BINDs to be performed against any recovered DBRMs. Promote copy and purge exit processing is also invoked during the package backout process. This ensures that the integrity of backed out load modules and ensures that any other exit processing that is in place during a normal promote copy or purge process is maintained.

Package Backout involves two phases, Backup and Restore. The backup phase occurs during a Promote process. For each member of a package marked for backout, it copies the old members to the existing backup data set, and saves the package details into a separate file, Before allowing the promote to continue. The restore phase occurs when users explicitly request to back out a package or an individual member. Restore promotes the old members back to the original group.

The Package Details file PDS holds details of both the backed up members and the editable members in the package. They can be used as input to determine the appropriate versions to be recovered. You as the SCLM Administrator define This PDS, using the PACKFILE=Y parameter on the FLMTYPE macro.

The Package Details PDS members hold the package backout information, such as:

- Package status
- Group
- Type
- Member
- Old member timestamp
- New member timestamp
- Timestamp when backed out
- Member status
- Member-level selection flag

A package has the status of "BACKEDUP" when it is initially backed up, and "RESTORED" after a package-level restore is performed. A similar status is retained against the backed-up member, showing either "BACKEDUP", or "RESTORED" if it is restored using a member-level restore.

To be able to recover source code of the Backed out Package, you must implement versioning for editable types that are promoted to a level at which Package Backout has been implemented. Package Backout by itself cannot control backout of editable types. Edit compare can be used to merge any desired changes from intermediate levels, and the member can be fixed and then built, tested, and promoted through the normal development process.

1.11.2 Package backout: Step-by-step instructions

Do the following steps to enable package backout for your project:

1. Determine the TYPE (for example, ARCHPACK) to hold the package high-level architecture members. If required allocate the appropriate data sets.
2. Update the project definition for this type to have the parameter ISAPACK=Y on the FLMTYPE macro. When an architecture member using this type is promoted, the package backout is invoked.
3. Determine the types of files (such as Object, load libraries) that are to be backed up during the promotion of a package high-level architecture member.
4. The Project definition for these file TYPES should be updated to specify the BACKUP=Y on the FLMTYPE macro.
5. Determine at which level (for example, production) the package backout is to be implemented, and the group that the members will be backed up to.
6. In the Project definition for this level, use the BKGRP=group_name parameter on the FLMGROUP macro to specify the group to which the members will be backed up.
7. This new backup group needs to be added to the project definition, so add an FLMGROUP macro for it.
8. Make sure the new backup group is a key group.
9. Use the group that is being backed up as the PROMOTE= group.

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

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

10. Determine if member-level restore is to be implemented to allow individual members to be restored instead of an entire package.
11. If it is required, update the FLMGROUP macro to have BKMBRLVL=Y.
12. Create the backup libraries for the TYPES you have specified with BACKUP=Y at the levels package backout has been specified.
13. The data sets will have the format <project_name>.<group_name>.<ds_type>, where group_name is the value specified on the BKGRP= parameter for each level.
14. Allocate the backup libraries with the same attributes as the libraries that are being backed up.
15. Determine the File type to contain the package backout details. Add the parameter PACKFILE=Y to the Project definition for this type.
16. The PACKFILE flag must only be specified on one FLMTYPE in the project definition, for example BACKUP FLMTYPE PACKFILE=Y
17. Allocate a library of this type at the level where BKGRP= is specified in the FLMGROUP macro.
18. Use the format <project_name>.<group_name>.<ds_type>, where ds_type is the type on the FLMTYPE macro with PACKFILE=Y.
19. Allocate this data set with LRECL=130 and RECFM=FB.
20. Determine if package reuse is to be used. If so set 'REUSEDAY=nnnn' on the FLMTYPE macro that has the PACKFILE=Y specified.
21. Reassemble and link the project definitions.

Archived

1.11.3 Package backout: Example project definition

Example A-15 on page 710 shows how you can customized the SCLM01 sample project definition to enable package backout.

1.12 Basic security considerations

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

The following sections describe the security requirements for the different types of data in the SCLM environment. Use this information to set up the security for the project environment.

1.12.1 PROJDEFS data sets

Restrict the project definition LOAD data set such that only you as the project manager have UPDATE authority to it. All other developers need only READ access to this data set. Developers have no need to update PROJDEFS SOURCE/OBJ data sets and should not have UPDATE access to those data sets. READ access can be granted to these data sets if this is reasonable for the project.

1.12.2 Project partitioned data sets

Each developer needs READ authority to all the project partitioned data sets.

Each developer needs UPDATE authority to the development groups that the individual uses to change SCLM-controlled members. UPDATE authority is also required for any groups the developer is allowed to promote into.

If the SCLM versioning capability is used, each developer needs UPDATE authority to the versioning partitioned data sets for the groups in which they work.

If the import/export capability is enabled, each developer needs UPDATE authority to the export data sets.

We recommend that the project manager have ALTER authority to all the project partitioned data sets.

1.12.3 Control data sets

Each developer in the project needs UPDATE authority to the control data sets that are updated by the developers.

Each developer needs READ access to the primary and secondary (if used) accounting data sets for all groups in the hierarchy. This authorization is required for SCLM to perform its verification.

If the auditing capability is used, each developer needs UPDATE authority to the audit control data sets.

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

Archived

1.13 Logon proc and FLMLIBS considerations

Let us now discuss some unique features of SCLM that help you take advantage of the z/OS Virtual Input Output (VIO) functionality, and also help customize the SCLM batch submission facility.

1.13.1 SCLM batch considerations

Before using the SCLM batch facility, modify the FLMLIBS skeleton to allow for batch submissions. The FLMLIBS skeleton is found in your ISPF skeleton target data set ISP.SISPSLIB. FLMLIBS is the common imbed for the other SCLM skeletons used for batch submission. Data set names in member FLMLIBS need to be modified to match your installation's naming conventions.

Notes:

- In Appendix A-16, Sample FLMLIBS Skeleton, the 'ISP' data set high-level qualifier represents your ISPF data sets.
- xxx corresponds to a national language as follows:
  - Language xxx
  - US English = ENU (the default)
  - Uppercase English = ENP
  - Swiss German = DES
  - Japanese = JPN
  - German = DEU

See Appendix A-16, "Sample FLMLIBS Skeleton" on page 711 for a sample FLMLIBS skeleton.

1.13.2 Preallocating ISPF temporary data sets to VIO

ISPF uses temporary data sets to generate JCL or utility control statements or to generate listings. To preallocate these data sets to VIO, include the DD statements in Appendix A-17, DD Statements to Preallocate Data Sets, in the TSO LOGON procedure.

Preallocation of these data sets to VIO is not mandatory; ISPF automatically allocates them to real data sets if required. However, we recommend preallocation, because it reduces overhead and eliminates potential problems from insufficient space.

See Appendix A-17, "DD Statements to Preallocate Data Sets" on page 712 for the DD statements to preallocate the data sets.

Notes:

- When allocating to VIO, make sure that enough auxiliary storage is dedicated to VIO so that system availability is not affected.
- Use of the BUFNO parameter on allocation of ISPF libraries is not supported.
- The ISPF temporary data set default names associated with the ISPCTLx are SPFTEMPx.CNTL, respectively, where x= value 0-9, A-W.
- The ISPF temporary data set default names associated with the ISPWRKx are SPFTEMPx.WORK, respectively, where x= value 1-9, A-W.
- The ddnames ISPWRKx are used by ISPF for file tailoring services with ISPFILE allocated to a PDS. The ddnames ISPLSTx are used for generated listings.
- The ddname ISPCTL0 is used with the edit SUBMIT command. The ddnames ISPCTLx are used with the PDF compress (both interactive and the LMCOMP service), PDF batch (option 5) and by ISPF for processing file tailoring service FTOPEN TEMP.
- ISPCTL1 is a required ddname when using SCLM. The data set should be allocated as an ISPF temporary data set for SCLM foreground processing and should be added to the FLMLIBS skeleton for batch processing.
- ISPF does not support multivolume temporary data sets. If DFSMS® is installed, temporary data sets dynamically allocated by ISPF must be assigned a data class with a volume count of one. This can be controlled in the ACS routines by testing the execution mode (&XMODE) for a value of TSO.
- If you have dialogs that need to edit or browse temporary data sets, use the LMINIT service to associate a DATAID with ddname ZTEMPN and invoke edit or browse using the DATAID parameter.
- For more information, refer to the BROWSE and EDIT services in the z/OS ISPF Services Guide.

Archive

Archived

2

Defining languages translators for traditional compliers: Assembler, COBOL, and PL/I

Language definitions provide SCLM with language-specific control information such as the language name and the definition of language parse, build, copy, and purge translators. By creating a language definition as part of the project definition, you specify to SCLM the languages that will be used for the project. You can customize the examples provided with the product to suit specific requirements of your project. You can also tailor SCLM to support languages other than those listed in those examples.

Archived

© Copyright IBM Corp. 2007. All rights reserved.

2.1 Overview

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

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

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

SCLM offers a language definition macro for each of the following definitions:

- FLMSYSLB — System library definitions (Optional)
- FLMLANGL — Language identifier definition (Required)
- FLMINCLS — Include set definitions (Optional)
- FLMTRNSL — Translator definitions (Optional)
- FLMALLOC — Allocation definitions (Optional)
- FLMCPYLB — Copy library definitions (Optional)

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

To determine what modifications you can make to the language definition, become familiar with the parameters of the language definition macros as documented in the z/OS ISPF Software Configuration and Library Manager Reference.

Typically, if you want to write a new language definition or customize an existing one, you should copy a sample language definition and then modify it to meet your specific needs. However it is recommended to use the member name as the same name as specified on the option LANG= of FLMLANGL macro. This will make it easier to find the language translator for a particular language.

2.2 Language definitions based upon samples

As indicated in Chapter 1, “SCLM project setup” on page 3, sample language definitions supplied by SCLM reside in the SISPMACS library of your installation. In the following sections, we examine several traditional z/OS language definitions more closely in order to describe some of the implementations of language definitions.

We describe the control structures used to manage SCLM processes and illustrate how to customize an SCLM sample language. We show the statements needed to define the control structures and SCLM macros. We also show how to gather the information you need and how to specify that information to SCLM in the form of language definition macros.

Archived

We discuss the following language definitions in this chapter:

FLM@COBE Enterprise COBOL language definition, LANG=COBE (Appendix A-4)

FLM@CCBE Enterprise COBOL with integrated CICS Precompile language definition, LANG=CICSCBE (Appendix B-1)

COBCICS Enterprise COBOL with separate CICS Precompile language definition, LANG=CCOBCICS (Appendix B-1)

FLM@PLIE Enterprise PL/I language definition, LANG=PLIE (Appendix A-7)

FLM@HLAS High-level assembler language definition, LANG=HLAS (Appendix A-5)

FLM@L370 z/OS binder/linkage editor language definition, LANG=LE370 (Appendix A-6)

See Appendix B-1, “SCLM01.PROJDEFS.SOURCE(FLM@CCBE)” on page 713 for the listing of the FLM@CCBE language definition with customizations for the example project, SCLM01, highlighted in bold. See Chapter 1, “SCLM project setup” on page 3 for the remaining language definitions.

In the next sections, we illustrate the process of defining each language to SCLM. As we progress through each language definition, we discuss SCLM macros along with the information SCLM needs to control translator modules. Reference language definitions in the code for a project definition by means of the COPY statement.

2.2.1 Using ddnames and ddname substitution lists

Many translators support a ddname substitution list; this contains ddnames, which are passed as a parameter to the translator. The position number indicates the position of the ddname in a ddname substitution list.

You tell SCLM that you will be using a ddname substitution list by use of the PORDER keyword on the FLMTRNSL macro. By using PORDER=3, or PORDER=2 if you do not want to pass the options list, you tell SCLM that for this translate step you will be using a list of DDs that correspond to the list, with order of the DDs expected by the compiler, or utility program. When you use a ddname substitution list, you must define the ddnames in the order in which they are expected to appear in the ddname substitution list by the translator. The first ddname defined is placed by SCLM into position 1 in the ddname substitution list. The second ddname specified is placed into position 2 in the ddname substitution list, and so on.

Notice that some position numbers do not have a ddname associated with them but they still need a nullfile FLMALLOC in that position.

You do not have to specify any ddnames in the macros of language definitions using a ddname substitution list. SCLM will create temporary unique ddnames and place them into the ddname substitution list positions. Because of the way ddname substitution lists work, the compiler uses those temporary ddnames instead of the standard documented ddnames (like SYSIN).

SCLM allows a maximum of 512 characters for the ddname substitution list. Because every FLMALLOC for a given translator causes an 8 character ddname to be put into the ddname substitution list, when the PORDER > 1, a given translator may have a maximum of 64 FLMALLOCs.

The ddname substitution lists are usually documented in the programming guide for specific compilers and linkage editors. Many of the ddname positions are common across some compilers, such as SYSLIB, but there are also a number of ddname positions that are different between compilers.

Compilers are not required to support a ddname substitution list in order to be defined to SCLM. However, ddname substitution list support makes it easy to link or string two different compilers or preprocessors together.

Where to find the ddname substitution lists

Here we give you some tips to help you find the ddname substitution list when you need it. Appendix E, “DDname substitution lists” on page 747 contains two tables listing the relative DD number and the DD name substituted in that position. Additionally, for reference, in this book we list the documented locations of this information. Here are the most recent versions of these guides:

- *High Level Assembler for MVS & VM & VSE Programmer's Guide Release 5*, SC26-4941:

  See the chapter referring to “Invoking the assembler dynamically.”

- *Enterprise COBOL for z/OS Programming Guide Version 3 Release 4*, SC27-1412

  See the chapter referring to “Starting the compiler from an assembler program.”

  Note: At the time of writing, we did not get the DB2 coprocessor working with PORDER=3. It is unclear at the moment if the Enterprise COBOL compiler supports ddname lists with the DB2 coprocessor.

- *COBOL for OS/390® & VM Programming Guide Version 2 Release 2*, SC26-9049

  See the chapter referring to “Starting the compiler from an assembler program.”

- *PL/I for MVS & VM Programming Guide*, SC26-3113

  See the chapter referring to “DDNAME list.”

- *z/OS V1R8.0 XL C/C++ User's Guide*, SC09-4767

  See the appendix referring to “Calling the z/OS XL C/C++ compiler from assembler.”

- *z/OS V1R8.0 MVS Program Management: Advanced Facilities*, SA22-7644

  See the chapter referring to “Invoking the binder from a program.”

- *z/OS V1R8.0 DFSMSdfp™ Utilities*, SC26-7414

  See the appendix referring to “Invoking Utility Programs from an Application Program.”

- *DB2 UDB for z/OS V8 Application Programming and SQL Guide*, SC18-7415

  See the chapter referring to “Starting the precompiler dynamically.”

- *CICS Application Programming Guide*, SC34-6433

  See the chapter referring to “Dynamic invocation of the separate translator.”

- *Enterprise PL/I for z/OS Programming Guide Version 3 Release 6*, SC27-1457

  Alternate ddname lists are supported through the DD compile option. The DD option allows you to specify alternate DD names for the various datasets used by the compiler. Up to eight DD names can be specified. In order, they specify alternate DD names for:

  - SYSPRINT
  - SYSIN
  - SYSLIB
  - SYSPUNCH
  - SYSLIN
  - SYSADATA
  - SYSXMLSD
  - SYSDEBUG

Archived

If you wanted to use ALTIN as the DD name for the primary compiler source file, you would have to specify DD(SYSPRINT,ALTIN). If you specified DD(ALTIN), SYSIN would be used as the ddname for the primary compiler source file and ALTIN would be used as the DD name for the compiler listing.

You can also use * to indicate that the default DD name should be used. Thus DD(*,ALTIN) is equivalent to DD(SYSPRINT,ALTIN).

For more information, see the chapter relating to “Compile-time option descriptions.”

Note: What this means from an SCLM perspective is that PORDER=3 is not supported for the Enterprise PL/I compiler, so PORDER=1 should be used. However the use of the DD compiler option with PORDER=1 will provide the equivalent functionality as previous versions of the compiler using PORDER=3.

2.2.2 Enterprise COBOL

In the following sections we describe the steps to set up a language definition for Enterprise COBOL using the supplied sample FLM@COBE as a template.

Step 1: Define the language

First, we tell SCLM that this is a new language definition. The following FLMLANGL macro does this as shown in Figure 2-1.

```
*
         FLMLANGL    LANG=COBE,VERSION=D071115,ALCSYSLB=Y,              C
               LANGDESC='ENTERPRISE COBOL'
*
```

Figure 2-1 FLMLANGL macro specification for Enterprise COBOL translator

In this example, values are specified for four parameters. Defaults are used for the other parameters:

LANG This specifies the language name a user must enter on the SPROF panel or on the Migrate Utility panel to request that this language definition be used to drive parse and build operations of the Enterprise COBOL modules.

VERSION Use up to 8 characters to identify the specific language version with a certain release of the current Enterprise COBOL compiler. If you install a new release or version of the Enterprise COBOL compiler, you can set this parameter to a different value so that SCLM can mark all Enterprise COBOL modules needing to be rebuilt. You must then re-assemble and re-link your project definition. A suggested format is *Dyymmdd* to indicate the date the translator was changed.

ALCSYSLB Verifies that external system libraries defined in FLMSYSLB are being allocated for build translators

LANGDESC This is a description of the language, maximum 40 characters.

Notice that the continuation character in column 72, in this case a C. If the parameters for a macro will spread beyond one (1) line, then a continuation character must be used.

Step 2: Define include sets to identify the location of included members

After the language is defined, you can specify where SCLM finds included members for the Enterprise COBOL language. As shown in Figure 2-2, the FLMINCLS macro identifies the SCLM controlled types that are searched for includes:

```
*
          FLMINCLS   TYPES=(COPYLIB)
*
```

Figure 2-2 FLMINCLS macro specification

The FLMINCLS macro specifies the name of the include set that uses this definition. If no name is specified (as in this example), the definition is associated with the default include set. An include set defines a search path for all includes associated with that include set. Multiple include sets can be specified in a language definition if the parser and compiler support distinguishing one kind of include from another. For the parser, this means that the syntax of the language must support determining which include set an include belongs to. For the compiler, this means that a separate ddname must be used for each different include set (kind of include). For example, Figure 2-3 shows how FLMINCLS could be specified when DB2 DCLGENs are read from a separate library:

```
*
         FLMINCLS TYPES=(COPYBOOK,DCLGEN)
DB2DCLS  FLMINCLS TYPES=(DCLGEN)
*
```

Figure 2-3 FLMINCLS specifying multiple include sets

In the above example, the COPYBOOK and DCLGEN could both have the same name, because they are included from different data sets.

The following parameter is used:

TYPES This specifies the name(s) of the types which are searched to find includes. In this case, SCLM searches the COPYLIB type first, and next the default @@FLMTYP type. The @@FLMTYP SCLM variable indicates that SCLM also needs to search the type of the member that is processed by the Enterprise COBOL compiler. If SCLM01.DEV1.SOURCE(IGYTSALE), for example, is going to be compiled, SCLM looks for includes first in the data sets associated with the COPYLIB type, and next the SOURCE type.

Step 3: Specify the programs that process the modules

Next, identify the programs that are used to parse and build the Enterprise COBOL modules. There are usually two such programs: a parser and a compiler. For each of these programs, code an FLMTRNSL macro and the appropriate FLMALLOC macros and FLMCPYLB macros.

Translator 1: Enterprise COBOL parser

The Enterprise COBOL parser is FLMLPCBL, and it resides in the ISPF load library ISP.SISPLPA. The parser requires an option string @@FLMSIZ,@@FLMSTP,@@FLMLIS, and reads the source from ddname SOURCE. A parser is a program that is called by SCLM at the time when a member is saved. The parsers generally parse the source and extract information such as the names of included modules and various bits of statistical information.

SCLM uses the included module information to maintain its dependency tables so that it knows which programs to recompile if an include changes.

There are many default parsers with SCLM, but you can also create your own, or modify some of the samples. For more information on modifying parsers, see Chapter 7. “Understanding and using the customizable parsers” in the *SCLM Guide and Reference*, SC34-4817-05. The parse translator is shown in Figure 2-4.

```
*
          FLMTRNSL   CALLNAM='SCLM COBOL PARSE',                        C
               FUNCTN=PARSE,                                            C
               COMPILE=FLMLPCBL,                                        C
               PORDER=1,                                                C
               CALLMETH=LINK,                                           C
               OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,)
```

Figure 2-4 FLMTRNSL macro specification for COBOL parser

The following parameters are used in this example:

CALLNAM This is a character string that appears in messages during the specified FUNCTN (in this case PARSE). This value will assist in recognizing which translator was executing during the specified FUNCTN.

FUNCTN The value PARSE tells SCLM that this program is to be invoked whenever you save a module with LANG=COBE.

COMPILE This is the member name of the load module for the Enterprise COBOL parser. Note that the keyword COMPILE actually identifies the load module name of a translator (which may or may not be a compiler).

PORDER The value 1 tells SCLM that this program expects an options string but not a ddname substitution list.

CALLMETH The value LINK specifies the invocation method of the module specified in the COMPILE parameter. For CALLMETH of ATTACH, LINK, or TSOLNK, this is the name of a REXX exec or CLIST, or the entry point to a load module. For a CALLMETH of ISPLNK, this must have a value of SELECT.

OPTIONS This specifies the options string to be passed to the parser. Strings that start with @@FLM are SCLM variables, and they are replaced by their current values before the string is passed to the parser.

DSNAME DSNAME is not used in this example, but it identifies the partitioned data set that contains the Enterprise COBOL parser load module. DSNAME is required only when the data set containing the desired module is not in the system’s ISPLLIB, STEPLIB, or LINKLIST concatenations. FLMLPCBL is in ISP.SISPLPA, which is defined in the LINKLIST concatenation, so DSNAME is not required in this case. When more than one data set is to be searched, you can use the TASKLIB parameter in conjunction with, or as a replacement for, the DSNAME parameter.

Since the parser reads its source from a ddname, you must tell SCLM how to allocate that ddname. To do this, we use an FLMALLOC macro and an FLMCPYLB macro as shown in Figure 2-5.

Archived

```
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
```

Figure 2-5 FLMALLOC and FLMCPYLB macro specifications for parser

The following parameters are used in this example:

IOTYPE=A The value A tells SCLM to allocate a ddname to one or more specific data set(s). Each of those data sets are subsequently identified by using an FLMCPYLB macro.

DDNAME This identifies the ddname to be allocated.

@@FLMDSN(@@FLMMBR) This identifies the member to be parsed. When the two SCLM variables are resolved, the parser is passed the data set and member that is being saved.

Translator 2: Enterprise COBOL compiler

Now you can tell SCLM how to invoke the Enterprise COBOL compiler. To do so, use an FLMTRNSL macro followed by one or more FLMALLOC and FLMCPYLB macros. The Enterprise COBOL compiler is IGYCRCTL, and on the system we used, it resides in the Enterprise COBOL load library ECOBOL.V340.SIGYCOMP as shown in Figure 2-6.

```
*
          FLMTRNSL   CALLNAM='ENTERPRISE COBOL COMPILER',             C
               FUNCTN=BUILD,                                          C
               COMPILE=IGYCRCTL,                                      C
               DSNAME=ECOBOL.V340.SIGYCOMP,                           C
               VERSION=3.4.0,                                         C
               GOODRC=0,                                              C
               PORDER=1,                                              C
               OPTIONS=(XREF,LIB,APOST,NODYNAM,LIST,NONUMBER,NOSEQ)
*
```

Figure 2-6 FLMTRNSL macro specification for COBOL compiler

You can specify as many parameters as required, and let SCLM supply default values for the others:

CALLNAM This is a character string that appears in the build messages. It names the Enterprise COBOL compiler.

FUNCTN This tells SCLM that this program gets invoked whenever you want to build a member with LANG=COBE.

COMPILE This identifies the load module name for the Enterprise COBOL compiler.

DSNAME Names the partitioned data set that contains the Enterprise COBOL compiler load module. Since ECOBOL.V340.SIGYCOMP is not defined in the LINKLIST concatenation, DSNAME is used here. There are *two ways* you can control where the compiler is loaded from:

Do specify the DSNAME so that you as the administrator are assured where the compiler is loaded from. The disadvantage is that, if you specify the version in the name, you will have to modify your translators when you install a new release of the compiler. The advantage is that you are assured of knowing where the compiler is loaded from.'

Do not specify DSNAME and load the compiler up from the system concatenation for ISPLLIB, STEPLIB and LINKLIST. The advantage is that you will not have to modify your translators. The disadvantage is that you might load a version of the compiler you are not expecting.

PORDER The value 1 tells SCLM that this program expects an options string but not a ddname substitution list.

GOODRC The value 4 indicates that SCLM is to consider this build unsuccessful if the compiler completes with a return code greater than 4, meaning only informational and warning diagnostics are acceptable.

OPTIONS This specifies the options string to be passed to the compiler.

Let us now examine the ddnames used by the Enterprise COBOL compiler in detail as specified in the sample as provided as shown in Figure 2-7. As we have used PORDER=1, we have only allocated the ddnames that are required:.

Allocate SYSLIN DD to the object module

```
*
           FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,                C
               RECNUM=5000,DFLTTYP=OBJ
```

Figure 2-7 FLMALLOC macro specification for SYSLIN DD

The following parameters are used in this macro:

IOTYPE=O The value O tells SCLM to allocate a temporary sequential data set to ddname SYSLIN. After the language definition completes successfully, the temporary sequential data set is copied as a member in the SCLM hierarchy.

DDNAME This identifies the ddname to be allocated. In this case SYSLIN.

KEYREF The value OBJ tells SCLM to save what is written to this ddname and keep it under SCLM control, SCLM must be able to determine the member name and the SCLM-controlled data set name in which it is to save this output module. If SCLM is building an architecture definition, it determines the project, group, type and member as follows:

The high-level qualifier is the project identifier that was previously specified.

The group is the level at which the build is taking place. The group name is the second qualifier.

SCLM looks at the architecture definition being built and retrieves the member and type from the architecture statement associated with the keyword OBJ. The type name is the third qualifier.

The construction of the PDS name, however, still follows the defined rules in FLMCNTRL or FLMALTC based on the SCLM identifiers @@FLMPRJ, @@FLMGRP and @@FLMTYP.

DFLTTYP The value OBJ tells SCLM to save what is written to this ddname to the SCLM type OBJ and keep it under SCLM control. If SCLM is building a source member, it determines the project, group, type and member as follows:

The high-level qualifier is the project identifier that was previously specified.

The group is the level at which the build is taking place.

The SCLM type is the value of the DFLTTYP= keyword.

The member name defaults to the name of the member being built.

If SCLM is building an architecture definition (and not a source member directly) then the DFLTTYP= value is ignored. Instead, SCLM uses the type associated with the KEYREF= value.

RECNUM The value 5000 tells SCLM to allocate enough space in this data set to hold 5000 records.

Allocate SYSLIB DD to one or more partitioned data sets

Enterprise COBOL uses this ddname to find included members. The FLMINCLS macro described earlier needs to be referenced here to ensure that the compiler is picking up includes from the correct data sets. Since IOTYPE=I allocations default to the default include set shown earlier, this is automatically done. If another name was used on the FLMINCLS macro, that name needs to be referenced here using the INCLS parameter.

IOTYPE=I allocates a ddname with a concatenation of all the PDSs in the hierarchy starting with the group specified for the BUILD and ending with the top, or production level, group. First the hierarchy for the INCLUDE type is allocated, followed by the type of the first SINCed member from the architecture definition, or, if no architecture definition is used, the type of the member being built. This is shown in Figure 2-8.

```
*
            FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
```

Figure 2-8 FLMALLOC macro specification for SYSLIB DD

The following parameters are used with this macro:

IOTYPE=I The value I tells SCLM to allocate this ddname to a concatenation of SCLM-controlled data sets. The types used in the concatenation are determined by the FLMINCLS macro referenced by the INCLS= parameter on the FLMALLOC macro. In this case, there is no INCLS= parameter so the default FLMINCLS (or include set) is used.

A hierarchy of data sets is concatenated for each type specified for the referenced FLMINCLS macro. The hierarchy begins at the group where the build is taking place and extends to the top of the project's hierarchy.

In this case, the concatenation first contains all of the data sets for the INCLUDES type followed by the data sets for the value substituted into the @@FLMTYP variable. See the KEYREF= parameter to determine the value which is substituted into the @@FLMTYP and @@FLMETP variables.

DDNAME This identifies the ddname to be allocated. In this case SYSLIB.

KEYREF The value SINC tells SCLM that, if you are building an architecture definition, refer to the first SINC statement in that architecture definition for the type that is substituted into the @@FLMTYP macro. The value for @@FLMETP comes from the EXTEND= parameter of the FLMTYPE macro for that type. If you are not building an architecture definition, the type is the type of the member being built.

Archive

Allocate ddname SYSIN to the source to be compiled

Figure 2-9 shows the FLMALLOC macro specification for SYSIN DD.

```
*
           FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000
```

Figure 2-9 FLMALLOC macro specification for SYSIN DD

The following parameters are used in this example:

IOTYPE=S The value S tells SCLM to allocate a temporary sequential data set and create the input stream for the translator by concatenating the contents of all the members that are SINCed as well as any text specified via CMD cards.

DDNAME This identifies the ddname to be allocated, in this case, SYSIN.

KEYREF The value SINC tells SCLM that, if you are building a source module directly, SCLM copies that member to this temporary data set. If you are building a CC architecture definition, SCLM copies the members listed on the SINC statement to this data set.

Allocate SYSUT1 through SYSUT7 DDs to temporary work data sets

The Enterprise COBOL compiler requires a number of temporary work data sets to be allocated as shown in Figure 2-10.

```
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT2,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT3,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT4,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT5,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT6,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT7,RECNUM=5000
```

Figure 2-10 FLMALLOC macro specifications for SYSUT work DDs

The following parameters are used in this example:

IOTYPE=W The value W tells SCLM to allocate a temporary sequential data set for translator use.

DDNAME This identifies the ddname to be allocated, in this case, SYSUT1 through SYSUT7.

Allocate SYSTERM and SYSPUNCH DDs to dummy data sets

These data sets are not needed for any output, so they can be allocated to a dummy data set as shown in Figure 2-11.

Archived

```
*
          FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
           FLMCPYLB NULLFILE
*
          FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
           FLMCPYLB NULLFILE
```

Figure 2-11 FLMALLOC macro specifications for SYSTERM and SYSPUNCH DDs

The following parameters are used in this example:

IOTYPE=A The value A tells SCLM to allocate a ddname to the data set identified by the next FLMCPYLB macro.

DDNAME This identifies the ddname to be allocated, in this case, SYSTERM and SYSPUNCH.

NULLFILE This allocates the specified DD to a dummy data set.

Allocate SYSPRINT DD to the compiler listing

Figure 2-12 shows the FLMALLOC macro specification for SYSPRINT DD.

```
          FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,KEYREF=LIST,            C
             RECFM=FBA,LRECL=133,                                    C
             RECNUM=50000,PRINT=Y,DFLTTYP=LIST
```

Figure 2-12 FLMALLOC macro specification for SYSPRINT DD

The following parameters are used in this example:

IOTYPE=O The value O tells SCLM to allocate a temporary sequential data set to ddname SYSPRINT. After the language definition completes successfully the temporary sequential data set is copied as a member in the SCLM hierarchy

DDNAME This identifies the ddname to be allocated, in this case, SYSPRINT.

KEYREF The value LIST refers SCLM to the LIST record in the architecture definition being built. That record contains the member name and type into which the listing is saved after a successful build. (SCLM copies the data from the temporary data sets into members of the PDSs controlled by SCLM after a successful build.)

DFLTTYP The value LIST specifies the data set type into which this listing is written whenever a module is built directly or when using INCLD in an architecture definition.

PRINT The value Y specifies that this is a listing that should be copied to the Build List data set after the build process completes.

RECFM The value FBA specifies the record format of the temporary data set that SCLM creates. In this example, the record format is fixed block with an ASA control character.

LRECL The value 133 specifies the record length, in characters, of the temporary data set that SCLM creates.

RECNUM The value 50000 tells SCLM to allocate enough space in this data set to hold 50000 records.

Archived

Language translator summary

These macros together constitute a language definition similar to one in Appendix A, “Chapter 1 listings” on page 691, SCLM01.PROJDEFS.SOURCE(FLM@COBE). It contains comments to explain the flow of operations. When you are ready to reassemble your project definition, add a COPY statement in your main project definition file to include the this language definition.

2.2.3 Enterprise COBOL with integrated CICS translator

In the following discussion, we describe the steps to set up a language definition for Enterprise COBOL with CICS using the supplied sample FLM@CCBE as a template. This is a single step translator that performs the CICS precompile and COBOL compile in a single translator step.

Step 1: Define static copy libraries

First, we tell SCLM which external copy libraries to use. For CICS, the system-wide CICS include libraries are required to be available to the translator. The CICS precompiles use the system include libraries CICS.TS31.CICS.SDFHCOB and CICS.TS31.CICS.SDFHMAC, which are common for the entire installation, and hence, must remain outside the control of individual SCLM projects. It is a good idea to code the required FLMSYSLB macro in a separate member that can be COPYed into other language translators.

The FLMSYSLB macro for the system CICS libraries is shown in Figure 2-13.

```
*
CICSCBE  FLMSYSLB  CICS.TS31.CICS.SDFHCOB
         FLMSYSLB  CICS.TS31.CICS.SDFHMAC
```

Figure 2-13 FLMSYSLB macro specification for Enterprise COBOL translator with CICS

The format of the macro is:

language FLMSYSLB *data set*

The language is an 8-character language name that must be the same name as the language specified in the LANG field on the FLMLANGL macro. To specify multiple data sets for a language, omit the language on all but the first data set. The data set parameter lists each “static” copy data set under the FLMSYSLB macro for this language definition.

Step 2: Define the language

Next, we tell SCLM that this is a new language definition. Figure 2-14, the FLMLANGL macro, does this.

```
*
        FLMLANGL LANG=CICSCBE,VERSION=DO71122,ALCSYSLB=Y,            C
              CHKSYSLB=BUILD,LANGDESC='ENTERPRISE COBOL WITH CICS'
*
```

Figure 2-14 FLMLANGL macro specification for Enterprise COBOL translator with CICS

In this example, values are specified for 5 parameters. Defaults are used for the other parameters.

LANG This specifies the language name a user must enter on the SPROF panel or on the Migrate Utility panel to request that this language definition be used to drive parse and build operations of the Enterprise COBOL with CICS modules.

VERSION Use up to 8 characters to identify the specific language version with a certain release of the current Enterprise COBOL compiler. If you install a new release or version of the Enterprise COBOL compiler, you can set this parameter to a different value so that SCLM can mark all Enterprise COBOL modules needing to be rebuilt. You must then re-assemble and re-link your project definition. A suggested format is *Dyymmdd* to indicate the date the translator was changed.

ALCSYSLB A value of Y verifies that system data sets are being allocated for build translators, and automatically adds these data sets to the concatenation of IOTYPE=I and KEYREF=SINC allocations.

CHKSYSLB The value BUILD defers checking of FLMSYSLB data sets until build time.

LANGDESC This is a description of the language, maximum 40 characters.

Notice the continuation character in column 72, in this case a C. If the parameters for a macro will spread beyond 1 line, then a continuation character must be used.

Step 3: Define include sets to identify the location of included members

After the language is defined, you can specify where SCLM finds included members for the Enterprise COBOL with CICS language. In Figure 2-15, the FLMINCLS macro identifies the types that are searched for.

```
*
          FLMINCLS   TYPES=(COPYLIB)
*
```

Figure 2-15 FLMINCLS macro specification

In this example, the TYPES parameter of the FLMINCLS macro is used to tell SCLM where to look for includes. Because no name label is specified, this definition applies to the default include set.

The FLMINCLS macro specifies the name of the include set that uses this definition. If no name is specified (as in this example), the definition is associated with the default include set. An include set defines a search path for all includes associated with that include set. Multiple include set s can be specified in a language definition if the parser and compiler support distinguishing one kind of include from another. For the parser, this means that the syntax of the language must support determining which include set an include belongs to. For the compiler, this means that a separate ddname must be used for each different include set (kind of include).

The following parameter is used:

TYPES This specifies the name(s) of the types which are searched to find includes. In this case, SCLM searches the COPYLIB type first, and next the default @@FLMTYP type. The @@FLMTYP SCLM variable indicates that SCLM also needs to search the type of the member that is processed by the Enterprise COBOL compiler.

Step 4: Specify the programs that process the modules

Next, identify the programs that are used to parse and build the Enterprise COBOL with CICS modules. In this example there are two such programs: a parser and a compiler, as the CICS precompile is handled within the Enterprise COBOL compile. For each of these programs, code an FLMTRNSL macro and the appropriate FLMALLOC macros and FLMCPYLB macros.

Translator 1: Enterprise COBOL parser

The Enterprise COBOL parser is FLMLPCBL and the definition is exactly the same as previously described above in , “Translator 1: Enterprise COBOL parser” on page 44.

Translator 2: Enterprise COBOL compiler

Now you can tell SCLM how to invoke the Enterprise COBOL compiler. To do so, use an FLMTRNSL macro followed by one or more FLMALLOC and FLMCPYLB macros. The Enterprise COBOL compiler is IGYCRCTL, and on the system we used it resides in the Enterprise COBOL load library ECOBOL.V340.SIGYCOMP. This is shown in Figure 2-16.

```
*
          FLMTRNSL   CALLNAM='COBOL COMPILER WITH CICS PREPROCESS',    C
               FUNCTN=BUILD,                                           C
               COMPILE=IGYCRCTL,                                       C
               VERSION=3.4.0,                                          C
               TASKLIB=TASKLIB,                                        C
               GOODRC=0,                                               C
               PORDER=1,                                               C
               OPTIONS=(RENT,NODYNAM,LIB,CICS(''COBOL3''))
*
```

Figure 2-16 FLMTRNSL macro specification for COBOL compiler with CICS coprocessor

Specify as many parameters as required, and let SCLM supply default values for the others:

CALLNAM This is a character string that appears in the build messages. It names the Enterprise COBOL compiler.

FUNCTN This tells SCLM that this program gets invoked whenever you want to build a member with LANG=COBE.

COMPILE This identifies the load module name for the Enterprise COBOL compiler.

VERSION Version of the compiler used for information purposes. This value is stored in the account record for any generated outputs.

TASKLIB This specifies the ddname associated with one or more data sets that contain the translator load module or modules. The data sets will be specified using an FLMALLOC macro with in our example, a ddname of TASKLIB. When specified for a translator using a ddname substitution list, the TASKLIB allocation does not appear in the list passed to the translator. TASKLIB is only valid for CALLMETH=ATTACH. The operating system searches for executable members in, if specified, the DSNAME parameter, then in the TASKLIB concatenation, and then in the system concatenation.

GOODRC The value 4 indicates that SCLM is to consider this build unsuccessful if the compiler completes with a return code greater than 4, meaning only informational and warning diagnostics are acceptable.

PORDER The value 1 tells SCLM that this program expects an options string but not a ddname substitution list.

OPTIONS This specifies the options string to be passed to the compiler. As this translator needs to perform a CICS precompile we must pass the parameter `CICS(''COBOL3'')` to the Enterprise COBOL compiler to tell it to translate the CICS statements.

Let us now examine the ddnames used by the Enterprise COBOL compiler in detail as specified in the sample as provided. As we have used PORDER=1 we have only allocated the ddnames that are required. We can see that as we are using the Enterprise COBOL translator and utilizing the built-in CICS translator, that the DDnams used are exactly the same as were used in the previous example, 2.2.2, “Enterprise COBOL” on page 43. So refer to that example for an explanation of the ddnames used in the COBOL compile step.

The only difference is the use of the TASKLIB option on the FLMTRNSL macro and therefore the provision of a TASKLIB FLMALLOC macro.

Allocate TASKLIB DD

In a single step translator that requires multiple load modules from different locations, as in this case with the CICS precompiler and COBOL compiler, we need to tell the translator where to find all the load modules. If the compilers are not allocated to the system linklist then we must define the load module locations in the translator. The DSNAME parameter on the FLMTRNSL macro only lets us specify a single data set. In our case we need both the CICS precompile and COBOL compile. So we must use a TASKLIB statement on the FLMTRNSL macro that specifies a ddname that lists all of the required compiler data sets as shown in Figure 2-17.

```
*          (* TASKLIB*)
           FLMALLOC  IOTYPE=A,DDNAME=TASKLIB
            FLMCPYLB  ECOBOL.V340.SIGYCOMP
            FLMCPYLB  CICS.TS31.CICS.SDFHLOAD
```

Figure 2-17 FLMALLOC macro specification for TASKLIB DD

A description of the parameters follows:

IOTYPE=A The value A tells SCLM to allocate a ddname to one or more specific data set(s). Each of those data sets are subsequently identified by using an FLMCPYLB macro.

DDNAME This identifies the ddname to be allocated.In this case TASKLIB. This matches the ddname specified by the TASKLIB parameter on the previous FLMTRNSL macro.

The 2 FLMCPYLB macros specify the data sets required by the compiler to perform the CICS translation as well as the Enterprise COBOL compilation.

Language translator summary

These macros together constitute a language definition similar to one in Appendix B, “Chapter 2 listings” on page 713, SCLM01.PROJDEFS.SOURCE(FLM@CCBE). It contains comments to explain the flow of operations. When you are ready to reassemble your project definition, add a COPY statement in your main project definition file to include the this language definition.

2.2.4 Enterprise COBOL with separate CICS precompile

With the advent of Enterprise COBOL with integrated CICS precompile and DB2 Coprocessor, it might not be necessary to run separate precompile and compile steps in your translators any more. The previous example showed the Enterprise COBOL with integrated CICS preprocessor. However, many customers might want to leave their translators as 2-step processors. Also, there will undoubtedly be occasions when a multistep build translator is required and as such it will be helpful to see how PORDER=3 can be used for ddname substitution to ensure that the correct DD names are passed into the second step. To this end we describe in detail the setup of the Enterprise COBOL with separate CICS precompiler. The full sample, COBCICS, is shown in Appendix B, “Chapter 2 listings” on page 713.

Step 1: Define static copy libraries

First, we tell SCLM which external copy libraries to use. For CICS, the system-wide CICS include libraries are required to be available to the translator. The CICS precompiles uses the system include libraries CICS.TS31.CICS.SDFHCOB and CICS.TS31.CICS.SDFHMAC, which are common for the entire installation, and hence, must remain outside the control of individual SCLM projects. A good idea is to code the required FLMSYSLB macro in a separate member that can be COPYed into other language translators.

The FLMSYSLB macro for the system CICS libraries is shown in Figure 2-18.

```
*
COBCICS  FLMSYSLB  CICS.TS31.CICS.SDFHCOB
         FLMSYSLB  CICS.TS31.CICS.SDFHMAC
```

Figure 2-18 FLMSYSLB macro specification for Enterprise COBOL translator with CICS

The format of the macro is:

language FLMSYSLB *data set*

The language is an 8-character language name that must be the same name as the language specified in the LANG field on the FLMLANGL macro. To specify multiple data sets for a language, omit the language on all but the first data set. The data set parameter lists each “static” copy data set under the FLMSYSLB macro for this language definition.

Step 2: Define the language

Next, we tell SCLM that this is a new language definition. The following FLMLANGL macro does this action in Figure 2-19.

```
*
        FLMLANGL  LANG=COBCICS,VERSION=DO71122,ALCSYSLB=Y,            C
              CHKSYSLB=BUILD,LANGDESC='ENTERPRISE COBOL WITH CICS'
*
```

Figure 2-19 FLMLANGL macro specification for Enterprise COBOL translator with CICS

In this example, values are specified for 5 parameters. Defaults are used for the other parameters.

LANG This specifies the language name a user must enter on the SPROF panel or on the Migrate Utility panel to request that this language definition be used to drive parse and build operations of the Enterprise COBOL with CICS modules.

VERSION Use up to 8 characters to identify the specific language version with a certain release of the current Enterprise COBOL compiler. If you install a new release or version of the Enterprise COBOL compiler, you can set this parameter to a different value so that SCLM can mark all Enterprise COBOL modules needing to be rebuilt. You must then re-assemble and re-link your project definition. A suggested format is *Dyymmdd* to indicate the date the translator was changed.

ALCSYSLB A value of Y verifies that system data sets are being allocated for build translators, and automatically adds these data sets to the concatenation of IOTYPE=I and KEYREF=SINC allocations.

CHKSYSLB The value BUILD defers checking of FLMSYSLB data sets until build time.

LANGDESC This is a description of the language, maximum 40 characters.

Notice the continuation character in column 72, in this case a C. If the parameters for a macro will spread beyond 1 line, then a continuation character must be used.

Step 3: Define include sets to identify the location of included members

After the language is defined, you can specify where SCLM finds included members for the Enterprise COBOL with CICS language. In Figure 2-20, the FLMINCLS macro identifies the types that are searched for.

```
*
          FLMINCLS   TYPES=(COPYLIB)
*
```

Figure 2-20 FLMINCLS macro specification

In this example, the TYPES parameter of the FLMINCLS macro is used to tell SCLM where to look for includes. Because no name label is specified, this definition applies to the default include set.

The FLMINCLS macro specifies the name of the include set that uses this definition. If no name is specified (as in this example), the definition is associated with the default include set. An include set defines a search path for all includes associated with that include set. Multiple include set s can be specified in a language definition if the parser and compiler support distinguishing one kind of include from another. For the parser, this means that the syntax of the language must support determining which include set an include belongs to. For the compiler, this means that a separate ddname must be used for each different include set (kind of include).

The following parameter is used:

TYPES This specifies the name(s) of the types which are searched to find includes. In this case, SCLM searches the COPYLIB type first, and next the default @@FLMTYP type. The @@FLMTYP SCLM variable indicates that SCLM also needs to search the type of the member that is processed by the Enterprise COBOL compiler.

Step 4: Specify the programs that process the modules

Next, identify the programs that are used to parse and build the Enterprise COBOL with CICS modules. There are three such programs: a parser, a precompiler and a compiler. For each of these programs, code an FLMTRNSL macro and the appropriate FLMALLOC macros and FLMCPYLB macros.

Translator 1: Enterprise COBOL parser

The Enterprise COBOL parser is FLMLPCBL and the definition is exactly the same as previously described above in , “Translator 1: Enterprise COBOL parser” on page 44.

Translator 2: CICS precompile

You are now ready to tell SCLM how to invoke the CICS precompiler. To do so, use an FLMTRNSL macro followed by one or more FLMALLOC and FLMCPYLB macros. The CICS precompiler is DFHECP1$, and it resides in the CICS Transaction Server (TS) load library CICS.TS31.CICS.SDFHLOAD as shown in Figure 2-21.

```
*         - CICS PRECOMPILE -
          FLMTRNSL   CALLNAM='CICS PRE-COMPILE',                    C
               FUNCTN=BUILD,                                        C
               COMPILE=DFHECP1$,                                    C
               DSNAME=CICS.TS31.CICS.SDFHLOAD,                      C
               VERSION=2.1,                                         C
               GOODRC=4,                                            C
               PORDER=3,                                            C
               OPTIONS=(SOURCE,NOSEQ)
```

Figure 2-21 FLMTRNSL macro specification for CICS precompiler

Specify as many parameters as required, and let SCLM supply default values for the others:

CALLNAM Names the CICS precompiler. This name appears in build messages.

FUNCTN This tells SCLM that this program gets invoked whenever you want to build a member with language Enterprise COBOL with CICS.

COMPILE This identifies the load module name, DFHECP1$, for the CICS precompiler.

DSNAME Names the partitioned data set that contains the CICS precompiler load module. Since CICS.TS31.CICS.SDFHLOAD is not defined in the LINKLIST concatenation, DSNAME is used here.

VERSION Version of the compiler used for information purposes. This value is stored in the account record for any generated outputs.

GOODRC The value 4 indicates that SCLM is to consider this build unsuccessful if the precompiler completes with a return code greater than 4, meaning Only informational and warning diagnostics are acceptable.

PORDER The value 3 tells SCLM that this program expects an options string and a ddname substitution list to the CICS precompiler. See 2.2.1, “Using ddnames and ddname substitution lists” on page 41 for more information on ddname substitution lists.

OPTIONS This specifies the options string to be passed to the precompiler.

By using PORDER=3, a ddname substitution list is passed to the CICS precompiler. This allows us to pass the output from the precompiler to the Enterprise COBOL compiler via the CICSTRAN DD allocation. The output from the CICS precompiler normally goes to SYSPUNCH and the input to the Enterprise COBOL compiler comes from SYSIN.

Archived

As these are two different DD names, using PORDER=1, SCLM is not going to be able to associate the output from the CICS precompile to the input to the Enterprise COBOL compile. PORDER=3 allows us to substitute a different ddname than is expected by the compiler for both the CICS precompile and COBOL compile, thus allowing us to associate the output from the CICS precompile to the input of the COBOL compile.

As discussed earlier in 2.2.1, “Using ddnames and ddname substitution lists” on page 41, when you use a ddname substitution list, you must define the ddnames in the order in which they are expected to appear in the ddname substitution list by the translator. The compiler uses temporary ddnames created by SCLM, or the ddname you tell SCLM to use, instead of the standard documented ddnames, such as SYSIN.

Let us now examine the ddnames used by the CICS precompiler in detail.

Skip over positions 1 through 4

The first four positions in the ddname substitution list are not used. Create an FLMALLOC macro with IOTYPE=N for each of them to tell SCLM to fill these name fields with hex zeros and to continue to the next ddname as shown in Figure 2-22.

```
*  1       -- N/A --
           FLMALLOC  IOTYPE=N
*  2       -- N/A --
           FLMALLOC  IOTYPE=N
*  3       -- N/A --
           FLMALLOC  IOTYPE=N
*  4       -- N/A --
           FLMALLOC  IOTYPE=N
```

Figure 2-22 FLMALLOC macro specifications to skip over positions 1 through 4

Allocate ddname SYSIN (Position 5) to the source to be precompiled

Figure 2-23 shows the FLMALLOC macro specification for SYSIN DD.

```
*  5       (* SYSIN *)
           FLMALLOC  IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,          C
               DDNAME=SYSIN
```

Figure 2-23 FLMALLOC macro specification for SYSIN DD

The following parameters are used in this example:

IOTYPE=S The value S tells SCLM to allocate a temporary sequential data set and create the input stream for the translator by concatenating the contents of all the members that are SINCed as well as any text specified via CMD cards.

KEYREF The value SINC tells SCLM that, if you are building a source module directly, SCLM copies that member to this temporary data set. If you are building a CC architecture definition, SCLM copies the members listed on the SINC statement to this data set.

RECFM The value FB specifies the record format of the temporary data set that SCLM creates. In this example, the record format is fixed block.

LRECL The value 80 specifies the record length, in characters, of the temporary data set that SCLM creates.

DDNAME This identifies the ddname to be allocated. In this case SYSIN.

Allocate SYSPRINT DD (Position 6) to the compiler listing

Figure 2-24 shows the FLMALLOC macro specification for SYSPRINT DD.

```
*  6       (* SYSPRINT *)
          FLMALLOC  IOTYPE=O,RECFM=FBA,LRECL=121,                    C
             RECNUM=35000,PRINT=Y
```

Figure 2-24 FLMALLOC macro specification for SYSPRINT DD

This definition contains the following parameters:

IOTYPE=O The value O tells SCLM that the precompiler writes to this ddname using a sequential data set. SCLM creates a temporary sequential data set and allocates it to a temporary ddname (since this is part of a ddname substitution list).

RECFM The value FBA specifies the record format of the temporary data set that SCLM creates. In this example, the record format is fixed block with an ASA control character.

LRECL The value 121 specifies the record length, in characters, of the temporary data set that SCLM creates.

RECNUM The value 35000 tells SCLM to allocate enough space in this data set to hold 35000 records.

PRINT The value Y specifies that this is a listing that should be copied to the Build List data set after the build process completes.

Allocate SYSPUNCH DD (Position 7) to substituted ddname of CICSTRAN

Next, define the SYSPUNCH ddname to SCLM for the precompiler modified source to pass to the Enterprise COBOL compiler as shown in Figure 2-25.

```
*  7       (* SYSPUNCH *)
          FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,                      C
             RECNUM=5000,DDNAME=CICSTRAN
```

Figure 2-25 FLMALLOC macro specifications for SYSTERM and SYSPUNCH DDs

The following parameters are used in this example:

IOTYPE=W The value W tells SCLM that the precompiler writes to this ddname using a sequential data set. SCLM creates a temporary sequential data set and allocates it to a ddname specified on the ddname parameter (since this is part of a ddname substitution list). See also the DDNAME parameter.

RECFM The value FB specifies the record format of the temporary data set that SCLM creates. In this example, the record format is fixed block.

LRECL The value 80 specifies the record length, in characters, of the temporary data set that SCLM creates.

RECNUM The value 5000 tells SCLM to allocate enough space in this data set to hold 5000 records.

DDNAME This tells SCLM to assign the ddname CICSTRAN to this allocation, so that the Enterprise COBOL compiler can refer back to it for the modified source.

Archived

Translator 3: Enterprise COBOL compiler

Now you can tell SCLM how to invoke the Enterprise COBOL compiler, and how to receive the modified source from the CICS precompiler. To do so, use an FLMTRNSL macro followed by one or more FLMALLOC and FLMCPYLB macros. The Enterprise COBOL compiler is IGYCRCTL, and it resides in the Enterprise COBOL load library ECOBOL.V340.SIGYCOMP. as shown in Figure 2-26.

```
*         --COBOL INTERFACE--
          FLMTRNSL   CALLNAM='COBOL',                                    C
               FUNCTN=BUILD,                                             C
               COMPILE=IGYCRCTL,                                         C
               DSNAME=ECOBOL.V340.SIGYCOMP,                              C
               VERSION=3.4.0,                                            C
               GOODRC=4,                                                 C
               PORDER=3,                                                 C
               OPTIONS=(DMA,PRI,SIZE=512K,APOS,CNT=77,BUF=30K,OPT,       C
               XREF,LIB)
```

Figure 2-26 FLMTRNSL macro specification for COBOL compiler with separate CICS precompile

You can specify as many parameters as required, and let SCLM supply default values for the others:

CALLNAM This is a character string that appears in the build messages. It names the Enterprise COBOL compiler.

FUNCTN This tells SCLM that this program gets invoked whenever you want to build a member with LANG=COBE.

COMPILE This identifies the load module name for the Enterprise COBOL compiler.

DSNAME Names the partitioned data set that contains the Enterprise COBOL compiler load module. Since ECOBOL.V340.SIGYCOMP is not defined in the LINKLIST concatenation, DSNAME is used here.

VERSION Version of the compiler used for information purposes. This value is stored in the account record for any generated outputs.

GOODRC The value 4 indicates that SCLM is to consider this build unsuccessful if the compiler completes with a return code greater than 4, meaning only informational and warning diagnostics are acceptable.

PORDER The value 3 tells SCLM that this program expects an options string and a ddname substitution list to the CICS precompiler. See 2.2.1, “Using ddnames and ddname substitution lists” on page 41 for more information on ddname substitution lists.

OPTIONS This specifies the options string to be passed to the compiler.

By using PORDER=3, a ddname substitution list is passed to the COBOL compiler. This allows the Enterprise COBOL compiler to receive input from the CICSTRAN DD allocation previously allocated in the CICS precompile step rather than the SYSIN DD as normally expected by the COBOL compiler.

Archived

As discussed earlier in 2.2.1, “Using ddnames and ddname substitution lists” on page 41, when you use a ddname substitution list, you must define the ddnames in the order in which they are expected to appear in the ddname substitution list by the translator. The compiler uses temporary ddnames created by SCLM, or the ddname you tell SCLM to use, instead of the standard documented ddnames, such as SYSIN.

Let us examine the ddnames used by the Enterprise COBOL compiler in detail:

Allocate SYSLIN DD (position 1) to the object module

Figure 2-27 shows the FLMALLOC macro specification for SYSLIN DD.

```
*  1       (* SYSLIN *)
           FLMALLOC  IOTYPE=O,KEYREF=OBJ,RECFM=FB,LRECL=80,            C
               RECNUM=5000,DFLTTYP=OBJ
```

Figure 2-27 FLMALLOC macro specification for SYSLIN DD

The following parameters are specified in this macro:

IOTYPE=O The value O tells SCLM to allocate a temporary sequential data set to ddname SYSLIN. After the language definition completes successfully the temporary sequential data set is copied as a member in the SCLM hierarchy

DDNAME This identifies the ddname to be allocated. In this case SYSLIN.

KEYREF The value OBJ tells SCLM to save what is written to this ddname and keep it under SCLM control, SCLM must be able to determine the member name and the SCLM-controlled data set name in which it is to save this output module. If SCLM is building an architecture definition, it determines the project, group, type and member as follows:

The high-level qualifier is the project identifier that was previously specified.

The group is the level at which the build is taking place. The group name is the second qualifier.

SCLM looks at the architecture definition being built and retrieves the member and type from the architecture statement associated with the keyword OBJ. The type name is the third qualifier.

The construction of the PDS name however still follows the defined rules in FLMCNTRL or FLMALTC based on the SCLM identifiers @@FLMPRJ, @@FLMGRP and @@FLMTYP.

DFLTTYP The value OBJ tells SCLM to save what is written to this ddname to the SCLM type OBJ and keep it under SCLM control. If SCLM is building a source member, it determines the project, group, type and member as follows:

The high-level qualifier is the project identifier that was previously specified.

The group is the level at which the build is taking place.

The SCLM type is the value of the DFLTTYP= keyword.

The member name defaults to the name of the member being built.

If SCLM is building an architecture definition (and not a source member directly) then the DFLTTYP= value is ignored. Instead, SCLM uses the type associated with the KEYREF= value.

RECNUM The value 5000 tells SCLM to allocate enough space in this data set to hold 5000 records.

Skipping positions 2 and 3

The first two positions in the ddname substitution list are not used. Create an FLMALLOC macro with IOTYPE=N for each of them to tell SCLM to fill these name fields with hex zeros and to continue to the next ddname as shown in Figure 2-28.

```
*  2       (* N/A *)
           FLMALLOC  IOTYPE=N
*  3       (* N/A *)
           FLMALLOC  IOTYPE=N
```

Figure 2-28 FLMALLOC macro specifications skipping positions 2 and3

Allocate SYSLIB DD (position 4) to one or more partitioned data sets

Allocate the ddname in position 4 of the ddname substitution list to one or more partitioned data sets. Enterprise COBOL uses this ddname to find included members. The FLMINCLS macro, if included in this language definition, ensures that the compiler is picking up includes from the correct data sets. IOTYPE=I allocates a ddname with a concatenation of all the PDSs in the hierarchy starting with the group specified for the BUILD and ending with the top, or production level, group. First the hierarchy for the INCLUDE type is allocated, followed by the type of the first SINCed member from the architecture definition, or, if no architecture definition is used, the type of the member being built, as shown in Figure 2-29.

```
*  4       (* SYSLIB *)
           FLMALLOC  IOTYPE=I,KEYREF=SINC,DDNAME=SYSLIB
```

Figure 2-29 FLMALLOC macro specification for SYSLIB DD

The following parameters are used with this macro:

IOTYPE=I The value I tells SCLM to allocate this ddname to a concatenation of SCLM-controlled data sets. The types used in the concatenation are determined by the FLMINCLS macro referenced by the INCLS= parameter on the FLMALLOC macro. In this case, there is no INCLS= parameter so the default FLMINCLS (or include set) is used.

A hierarchy of data sets is concatenated for each type specified for the referenced FLMINCLS macro. The hierarchy begins at the group where the build is taking place and extends to the top of the project's hierarchy.

In this case, the concatenation first contains all of the data sets for the INCLUDES type followed by the data sets for the value substituted into the @@FLMTYP variable. See the KEYREF= parameter to determine the value which is substituted into the @@FLMTYP and @@FLMETP variables.

KEYREF The value SINC tells SCLM that, if you are building an architecture definition, refer to the first SINC statement in that architecture definition for the type that is substituted into the @@FLMTYP macro. The value for @@FLMETP comes from the EXTEND= parameter of the FLMTYPE macro for that type. If you are not building an architecture definition, the type is the type of the member being built.

DDNAME This identifies the ddname to be allocated. In this case SYSLIB.

Allocate ddname SYSIN (position 5) to the source to be compiled

Figure 2-30 shows the FLMALLOC macro specification for SYSINDD.

```
*  5       (* SYSIN *)
           FLMALLOC  IOTYPE=U,KEYREF=SINC,DDNAME=CICSTRAN
```

Figure 2-30 FLMALLOC macro specification for SYSIN DD

The following parameters are used in this example:

IOTYPE=U The value U tells SCLM to allocate a temporary sequential data set using ddname CICSTRAN. CICSTRAN refers back to the same ddname preallocated by the CICS precompiler for the modified source.

KEYREF The value SINC is ignored for IOTYPE=U

DDNAME This tells SCLM to assign the ddname CICSTRAN to this allocation for the Enterprise COBOL compiler to refer back to it for the modified source.

Allocate SYSPRINT DD (position 6) to the compiler listing

Figure 2-31 shows the FLMALLOC macro specification for SYSPRINT DD.

```
 *  6       (* SYSPRINT *)
           FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=FBA,LRECL=133,         C
              RECNUM=25000,PRINT=Y,DFLTTYP=LIST
```

Figure 2-31 FLMALLOC macro specification for SYSPRINT DD

This definition contains the following parameters:

IOTYPE=O The value O tells SCLM to allocate a temporary sequential data set to ddname SYSPRINT. After the language definition completes successfully the temporary sequential data set is copied as a member in the SCLM hierarchy

KEYREF The value LIST refers SCLM to the LIST record in the architecture definition being built. That record contains the member name and type into which the listing is saved after a successful build. (SCLM copies the data from the temporary data sets into members of the PDSs controlled by SCLM after a successful build.)

RECFM The value FBA specifies the record format of the temporary data set that SCLM creates. In this example, the record format is fixed block with an ASA control character.

LRECL The value 133 specifies the record length, in characters, of the temporary data set that SCLM creates.

RECNUM The value 25000 tells SCLM to allocate enough space in this data set to hold 25000 records.

PRINT The value Y specifies that this is a listing that should be copied to the Build List data set after the build process completes.

DFLTTYP The value LIST specifies the data set type into which this listing is written whenever a module is built directly or when using INCLD in an architecture definition.

Allocate SYSPUNCH DD (position 7) to dummy data sets

These data sets are not needed for any output, so they can be allocated to a dummy data set as shown in Figure 2-32.

```
*  7       (* SYSPUNCH *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
```

Figure 2-32 FLMALLOC macro specifications for SYSPUNCH DD

The following parameters are used in this example:

IOTYPE=A The value A tells SCLM to allocate a ddname to the data set identified by the next FLMCPYLB macro.

NULLFILE This allocates the specified DD to a dummy data set.

Allocate SYSUT1 through SYSUT4 DDs to temporary work data sets

DD positions 8 through 11 in the ddname substitution list are allocated to temporary work data sets for the compiler as shown in Figure 2-33.

```
*  8       (* SYSUT1 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
*  9       (* SYSUT2 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 10       (* SYSUT3 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 11       (* SYSUT4 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
```

Figure 2-33 FLMALLOC macro specifications for SYSUT work DDs

The following parameters are used in this example:

IOTYPE=W The value W tells SCLM to allocate a temporary sequential data set for translator use.

DDNAME This identifies the ddname to be allocated. In this case SYSUT1 through SYSUT7.

Allocate SYSTERM, SYSUT5 and SYSUT6 DDs to dummy data sets

These data sets are not needed for any output, so they can be allocated to a dummy data set as shown in Figure 2-34.

```
* 12       (* SYSTERM *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
* 13       (* SYSUT5 *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
* 14       (* SYSUT6 *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
```

Figure 2-34 FLMALLOC macro specifications for SYSPUNCH DD

The following parameters are used in this example:

IOTYPE=A The value A tells SCLM to allocate a ddname to the data set identified by the next FLMCPYLB macro.

NULLFILE This allocates the specified DD to a dummy data set.

Additional data sets expected by compiler using ddname substitution

DD positions 15 through 19 in the ddname substitution list are not specified because they are not used by this invocation of the compiler. However, if you were using the debug side file, which is allocated to position 18 (SYSDEBUG), then you would need to specify NULLFILE allocations for positions 15, 16 and 17 prior to specifying an FLMALLOC for SYSDEBUG.

Note: Use of PORDER=3 and the ddname substitution list is mandatory for the Enterprise COBOL compiler step, but it is optional for the CICS precompiler step. It is mandatory, not because Enterprise COBOL requires it, but because the refer-back is to a ddname other than SYSIN. If you choose to use PORDER=1 and explicit ddnames on FLMALLOC macros on the CICS precompiler step, you can pass the precompiler output via the SYSPUNCH DD allocation to Enterprise COBOL. See Chapter 13, “Creating language definitions from JCL” on page 289, for an example of this approach.

Language translator summary

These macros together constitute a language definition similar to one in Appendix B, “Chapter 2 listings” on page 713, SCLM01.PROJDEFS.SOURCE(COBCICS). It contains comments to explain the flow of operations. When you are ready to reassemble your project definition, add a COPY statement in your main project definition file to include the this language definition.

2.2.5 Enterprise PL/I

The following describes the steps to setup a language definition for Enterprise PL/I using the supplied sample, FLM@PLIE, as a template.

Step 1: Define the language

First we tell SCLM that this is a new language definition. Figure 2-35 shows the FLMLANGL macro that does this.

```
*
         FLMLANGL    LANG=PLIE,VERSION=D071113,                          C
               LANGDESC='ENTERPRISE PL/I'
```

Figure 2-35 FLMLANGL macro specification for Enterprise PL/I

In this example, values are specified for 3 parameters. Defaults are used for the other parameters.

LANG This specifies the language name a user must enter on the SPROF panel or on the Migrate Utility panel to request that this language definition be used to drive parse and build operations of the Enterprise PL/I modules.

VERSION Use up to 8 characters to identify the specific language version with a certain release of the current Enterprise PL/I compiler. If you install a new release or version of the Enterprise PL/I compiler, you can set this parameter to a different value so that SCLM can mark all Enterprise PL/I modules needing to be rebuilt. You must then re-assemble and re-link your project definition. A suggested format is *Dyymmdd* to indicate the date the translator was changed.

LANGDESC This is a description of the language, maximum 40 characters.

Notice the continuation character in column 72, in this case a C. If the parameters for a macro will spread beyond 1 line, then a continuation character must be used.

Step 2: Define include sets to identify the location of included members

After the language is defined, you can specify where SCLM finds included members for the Enterprise PL/I language. In Figure 2-36, the FLMINCLS macro identifies the types that are searched for includes:

```
*
          FLMINCLS   TYPES=(PLINCL)
*
```

Figure 2-36 FLMINCLS macro specification for Enterprise PL/I

In this example, the TYPES parameter of the FLMINCLS macro is used to tell SCLM where to look for includes. Because no name label is specified, this definition applies to the default include set.

The FLMINCLS macro specifies the name of the include set that uses this definition. If no name is specified (as in this example), the definition is associated with the default include set. An include set defines a search path for all includes associated with that include set. Multiple include set s can be specified in a language definition if the parser and compiler support distinguishing one kind of include from another. For the parser, this means that the syntax of the language must support determining which include set an include belongs to. For the compiler, this means that a separate ddname must be used for each different include set (kind of include).

The following parameter is used:

TYPES This specifies the name(s) of the types which are searched to find includes. In this case, SCLM searches the PLINCL type first, and next the default @@FLMTYP type. The @@FLMTYP SCLM variable indicates that SCLM also needs to search the type of the member that is processed by the Enterprise PL/I compiler. For example, if 'SCLM01.DEV2.SOURCE(FLM01MD2)' is being compiled, SCLM looks for includes first in the data sets associated with the PLINCL type, and next the SOURCE type.

Step 3: Specify the programs that process the modules

Next, identify the programs that are used to parse and build the Enterprise PL/I modules. There are two such programs: a parser and a compiler. For each of these programs, code an FLMTRNSL macro and the appropriate FLMALLOC macros and FLMCPYLB macros.

The Enterprise PL/I parser is FLMLPGEN, and it resides in the ISPF load library ISP.SISPLPA. The parser requires five option parameters, SOURCEDD=SOURCE, STATINFO=@@STP, LISTINFO=@@LIS, LISTSIZE=@@SIZ and LANG=I, and reads the source from ddname SOURCE.

The parse translator is shown in Figure 2-37.

Archived

```
*
          FLMTRNSL   CALLNAM='SCLM PL/I PARSE',                          C
               FUNCTN=PARSE,                                             C
               COMPILE=FLMLPGEN,                                         C
               PORDER=1,                                                 C
               OPTIONS=(SOURCEDD=SOURCE,                                 C
               STATINFO=@@FLMSTP,                                        C
               LISTINFO=@@FLMLIS,                                        C
               LISTSIZE=@@FLMSIZ,                                        C
               LANG=I)
```

Figure 2-37 FLMTRNSL macro specification for Enterprise PL/I parser

The following parameters are used in this example:

CALLNAM This is a character string that appears in messages during the specified FUNCTN (in this case PARSE). This value will assist in recognizing which translator was executing during the specified FUNCTN.

FUNCTN The value PARSE tells SCLM that this program is to be invoked whenever you save a module with LANG=PLIE.

COMPILE Member name of the load module for the Enterprise PL/I parser. Note that the keyword COMPILE actually identifies the load module name of a translator (which may or may not be a compiler).

PORDER The value 1 tells SCLM that this program expects an options string but not a ddname substitution list.

OPTIONS This specifies the options string to be passed to the parser. Strings that start with @@FLM are SCLM variables, and they are replaced by their current values before the string is passed to the parser.

The LANG=I options parameter tells the parser that this is a PL/I member. FLMLPGEN is a general purpose parser, and hence, needs to know whether this is a assembler, PL/I, REXX or text member.

CALLMETH The CALLMETH parameter is not specified so the default value of ATTACH specifies the invocation method of the module specified in the COMPILE parameter.

DSNAME DSNAME is not used in this example, but it identifies the partitioned data set that contains the Enterprise PL/I parser load module. DSNAME is required only when the data set containing the desired module is not in the system's ISPLLIB, STEPLIB or LINKLIST concatenations. FLMLPGEN is in ISP.SISPLPA which is defined in the LINKLIST concatenation, so DSNAME is not required in this case. When more than one data set is to be searched, you can use the TASKLIB parameter in conjunction with, or as a replacement for, the DSNAME parameter.

Since the parser reads its source from a ddname, you must tell SCLM how to allocate that ddname. To do this, we use an FLMALLOC macro and an FLMCPYLB macro as shown in Figure 2-38.

```
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
```

Figure 2-38 FLMALLOC and FLMCPYLB macro specifications for parser

The following parameters are used in this example:

IOTYPE=A The value A tells SCLM to allocate a ddname to one or more specific data set(s). Each of those data sets are subsequently identified by using an FLMCPYLB macro.

DDNAME This identifies the ddname to be allocated.

@@FLMDSN(@@FLMMBR) This identifies the member to be parsed. When the two SCLM variables are resolved, the parser is passed the data set and member that is being saved.

Now you can tell SCLM how to invoke the Enterprise PL/I compiler. To do so, use an FLMTRNSL macro, as shown in Figure 2-39, followed by one or more FLMALLOC and FLMCPYLB macros. The Enterprise PL/I compiler is IBMZPLI, and on the system we used it resides in the Enterprise PL/I load library EPLI.V350.SIBMZCMP.

```
*
          FLMTRNSL   CALLNAM='ENTERPRISE PL/I COMPILER',              C
               FUNCTN=BUILD,                                          C
               COMPILE=IBMZPLI,                                       C
               DSNAME=EPLI.V350.SIBMZCMP,                             C
               VERSION=3.5.0,                                         C
               GOODRC=0,                                              C
               PORDER=1,                                              C
               OPTIONS=(MACRO,OBJECT,SOURCE,XREF)
*
```

Figure 2-39 FLMTRNSL macro specification for Enterprise PL/I

You can specify as many parameters as required, and let SCLM supply default values for the others:

CALLNAM This is a character string that appears in the build messages. It names the Enterprise PL/I compiler.

FUNCTN This tells SCLM that this program gets invoked whenever you want to build a member with LANG=PLIE.

COMPILE This identifies the load module name for the Enterprise PL/I compiler.

DSNAME Names the partitioned data set that contains the Enterprise PL/I compiler load module. Since EPLI.V350.SIBMZCMP is not defined in the LINKLIST concatenation, DSNAME is used here. There are *two ways* you can control where the compiler is loaded from:

Do specify the DSNAME so that you as the administrator are assured where the compiler is loaded from. The disadvantage is that, if you specify the version in the name, you will have to modify your translators when you install a new release of the compiler. The advantage is you are assured of where the compiler is loaded from.'

Do not specify DSNAME and load the compiler up from the system concatenation for ISPLLIB, STEPLIB and LINKLIST. The advantage that is you will not have to modify your translators. The disadvantage is you may load a version of the compiler you are not expecting.

VERSION This is the version of the compiler used for information purposes. This value is stored in the account record for any generated outputs.

Archived

GOODRC The value 0 indicates that SCLM is to consider this build unsuccessful if the compiler completes with a return code greater than 0, meaning only informational are acceptable.

PORDER The value 1 tells SCLM that this program expects an options string but not a ddname substitution list.

OPTIONS This specifies the options string to be passed to the compiler.

Let us now examine the ddnames used by the Enterprise PL/I compiler in detail as specified in the sample as provided. As we have used PORDER=1 we have allocated the minimum ddnames that are required.

Allocate SYSLIN DD to the object module

Figure 2-40 shows the FLMALLOC macro specification for SYSLIN DD.

```
*
          FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,DFLTTYP=OBJ,    C
             RECNUM=5000
```

Figure 2-40 FLMALLOC macro specification for SYSLIN DD

The following parameters are specified in this macro:

IOTYPE=O The value O tells SCLM to allocate a temporary sequential data set to ddname SYSLIN. After the language definition completes successfully the temporary sequential data set is copied as a member in the SCLM hierarchy

DDNAME This identifies the ddname to be allocated. In this case SYSLIN.

KEYREF The value OBJ tells SCLM to save what is written to this ddname and keep it under SCLM control, SCLM must be able to determine the member name and the SCLM-controlled data set name in which it is to save this output module. If SCLM is building an architecture definition, it determines the project, group, type and member as follows:

The high-level qualifier is the project identifier that was previously specified.

The group is the level at which the build is taking place. The group name is the second qualifier.

SCLM looks at the architecture definition being built and retrieves the member and type from the architecture statement associated with the keyword OBJ. The type name is the third qualifier.

The construction of the PDS name however still follows the defined rules in FLMCNTRL or FLMALTC based on the SCLM identifier @@FLMPRJ, @@FLMGRP and @@FLMTYP.

DFLTTYP The value OBJ tells SCLM to save what is written to this ddname to the SCLM type OBJ and keep it under SCLM control. If SCLM is building a source member, it determines the project, group, type and member as follows:

The high-level qualifier is the project identifier that was previously specified.

The group is the level at which the build is taking place.

The SCLM type is the value of the DFLTTYP= keyword.

The member name defaults to the name of the member being built.

Archived

If SCLM is building an architecture definition (and not a source member directly) then the DFLTTYP= value is ignored. Instead, SCLM uses the type associated with the KEYREF= value.

RECNUM The value 5000 tells SCLM to allocate enough space in this data set to hold 5000 records.

Allocate SYSLIB DD to one or more partitioned data sets

Enterprise PL/I uses this ddname to find included members. The FLMINCLS macro described earlier needs to be referenced here to ensure that the compiler is picking up includes from the correct data sets. Since IOTYPE=I allocations default to the default include set shown earlier, this is automatically done. If another name was used on the FLMINCLS macro, that name needs to be referenced here using the INCLS parameter. IOTYPE=I allocates a ddname with a concatenation of all the PDSs in the hierarchy starting with the group specified for the BUILD and ending with the top, or production level, group. First the hierarchy for the INCLUDE type is allocated, followed by the type of the first SINCed member from the architecture definition, or, if no architecture definition is used, the type of the member being built.This macro is shown in Figure 2-41.

```
*
            FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
```

Figure 2-41 FLMALLOC macro specification for SYSLIB DD

The following parameters are used in this example:

IOTYPE=I The value I tells SCLM to allocate this ddname to a concatenation of SCLM-controlled data sets. The types used in the concatenation are determined by the FLMINCLS macro referenced by the INCLS= parameter on the FLMALLOC macro. In this case, there is no INCLS= parameter so the default FLMINCLS (or include set) is used.

A hierarchy of data sets is concatenated for each type specified for the referenced FLMINCLS macro. The hierarchy begins at the group where the build is taking place and extends to the top of the project's hierarchy.

In this case, the concatenation first contains all of the data sets for the INCLUDES type followed by the data sets for the value substituted into the @@FLMTYP variable. See the KEYREF= parameter to determine the value which is substituted into the @@FLMTYP and @@FLMETP variables.

DDNAME This identifies the ddname to be allocated. In this case SYSLIB.

KEYREF The value SINC tells SCLM that, if you are building an architecture definition, refer to the first SINC statement in that architecture definition for the type that is substituted into the @@FLMTYP macro. The value for @@FLMETP comes from the EXTEND= parameter of the FLMTYPE macro for that type. If you are not building an architecture definition, the type is the type of the member being built.

If you have any %INCLUDE ddname(member) statements in your Enterprise PL/I code, you need an FLMALLOC macro for each unique ddname in your language definition. You also need multiple PL/I include types, one for each ddname, if you want to keep the includes separate in SCLM. You can also specify multiple FLMINCLS macros to define a search order for each of the different include sets. An example of this can be seen in , “Step 2: Define include sets to identify the location of included members” on page 44. in the Enterprise COBOL example. A common use of this would be for normal includes used in conjunction with DB2 DCLGENs that share the same name.

Allocate SYSUT1 DD to temporary work data sets

The Enterprise PL/I compiler requires a temporary work data set to be allocated as shown in Figure 2-42.

```
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,RECNUM=5000
```

Figure 2-42 FLMALLOC macro specification for SYSUT1 work DD

The following parameters are used in this example:

IOTYPE=W The value W tells SCLM to allocate a temporary sequential data set for translator use.

DDNAME This identifies the ddname to be allocated. In this case SYSUT1.

Allocate SYSIN DD to the source to be compiled

Figure 2-43 shows the FLMALLOC macro specification for SYSIN DD.

```
*
           FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000
```

Figure 2-43 FLMALLOC macro specification for SYSIN DD

The following parameters are used in this example:

IOTYPE=S The value S tells SCLM to allocate a temporary sequential data set and create the input stream for the translator by concatenating the contents of all the members that are SINCed as well as any text specified via CMD cards.

DDNAME This identifies the ddname to be allocated. In this case SYSIN.

KEYREF The value SINC tells SCLM that, if you are building a source module directly, SCLM copies that member to this temporary data set. If you are building a CC architecture definition, SCLM copies the members listed on the SINC statement to this data set.

Allocate SYSPUNCH DD to a dummy data set

This data set is not needed for any output, so they can be allocated to a dummy data set as shown in Figure 2-44.

```
*
          FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
           FLMCPYLB NULLFILE
```

Figure 2-44 FLMALLOC macro specification for SYSPUNCH DD

The following parameters are used in this example:

IOTYPE=A The value A tells SCLM to allocate a ddname to the data set identified by the next FLMCPYLB macro.

DDNAME This identifies the ddname to be allocated. In this case SYSPUNCH.

NULLFILE This allocates the specified DD to a dummy data set.

Allocate SYSPRINT DD to the compiler listing

Figure 2-45 shows the FLMALLOC macro specification for SYSPRINT DD.

Archived

```
*
          FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,                          C
              KEYREF=LIST,DFLTTYP=LIST,                               C
              RECFM=VBA,LRECL=137,                                    C
              PRINT=Y,RECNUM=5000
```

Figure 2-45 FLMALLOC macro specification for SYSPRINT DD

This definition contains the following parameters:

IOTYPE=O The value O tells SCLM to allocate a temporary sequential data set to ddname SYSPRINT. After the language definition completes successfully the temporary sequential data set is copied as a member in the SCLM hierarchy

DDNAME This identifies the ddname to be allocated. In this case SYSPRINT.

KEYREF The value LIST refers SCLM to the LIST record in the architecture definition being built. That record contains the member name and type into which the listing is saved after a successful build. (SCLM copies the data from the temporary data sets into members of the PDSs controlled by SCLM after a successful build.)

DFLTTYP The value LIST specifies the data set type into which this listing is written whenever a module is built directly or when using INCLD in an architecture definition.

RECFM The value VBA specifies the record format of the temporary data set that SCLM creates. In this example, the record format is variable blocked with an ASA control character.

LRECL The value 137 specifies the record length, in characters, of the temporary data set that SCLM creates.

PRINT The value Y specifies that this is a listing that should be copied to the Build List data set after the build process completes.

RECNUM The value 5000 tells SCLM to allocate enough space in this data set to hold 5000 records.

Language translator summary

These macros together constitute a language definition similar to one in Appendix A, “Chapter 1 listings” on page 691, SCLM01.PROJDEFS.SOURCE(FLM@PLIE). It contains comments to explain the flow of operations. When you are ready to reassemble your project definition, add a COPY statement in your main project definition file to include the this language definition.

2.2.6 High Level Assembler

The following describes the steps to setup a language definition for High Level Assembler using the supplied sample FLM@HLAS as a template.

Step 1: Define static copy libraries

First, we tell SCLM which external copy libraries to use. The assembler language uses the system macro library SYS1.MACLIB which is common for the entire installation, and hence, must remain outside the control of individual SCLM projects. A good idea is to code the required FLMSYSLB macro in a separate member that can be COPYed into other language translators.

The FLMSYSLB macro for the system CICS libraries is shown in Figure 2-46.

```
*
HLAS     FLMSYSLB SYS1.MACLIB
```

Figure 2-46 FLMSYSLB macro specification for High Level Assembler

The format of the macro is:

language FLMSYSLB *data set*

The language is an 8-character language name that must be the same name as the language specified in the LANG field on the FLMLANGL macro. To specify multiple data sets for a language, omit the language on all but the first data set. The data set parameter lists each “static” copy data set under the FLMSYSLB macro for this language definition.

Step 2: Define the language

Next, we tell SCLM that this is a new language definition. The following FLMLANGL macro does this as shown in Figure 2-47.

```
*
         FLMLANGL    LANG=HLAS,VERSION=D071123,ALCSYSLB=Y,              C
               LANGDESC='HIGH LEVEL ASSEMBLER'
```

Figure 2-47 FLMLANGL macro specification for High Level Assembler

In this example, values are specified for four parameters. Defaults are used for the other parameters:

LANG This specifies the language name a user must enter on the SPROF panel or on the Migrate Utility panel to request that this language definition be used to drive parse and build operations of the High Level Assembler modules.

VERSION Use up to 8 characters to identify the specific language version with a certain release of the current High Level Assembler. If you install a new release or version of the High Level Assembler, you can set this parameter to a different value so that SCLM can mark all High Level Assembler modules needing to be rebuilt. You must then re-assemble and re-link your project definition. A suggested format is *Dyymmdd* to indicate the date the translator was changed.

ALCSYSLB A value of Y verifies that system data sets are being allocated for build translators, and automatically adds these data sets to the concatenation of IOTYPE=I and KEYREF=SINC allocations.

LANGDESC This is a description of the language, maximum 40 characters.

Notice the continuation character in column 72, in this case a C. If the parameters for a macro will spread beyond 1 line, then a continuation character must be used.

Step 3: Define include sets to identify the location of included members

After the language is defined, you can specify where SCLM finds included members for the High Level Assembler language. Figure 2-48 shows the FLMINCLS macro that identifies the types that are searched for includes.

Archived

```
*
          FLMINCLS   TYPES=(MACROS)
*
```

Figure 2-48 FLMINCLS macro specification for High Level Assembler

In this example, the TYPES parameter of the FLMINCLS macro is used to tell SCLM where to look for included macros. Because no name label is specified, this definition applies to the default include set.

The FLMINCLS macro specifies the name of the include set that uses this definition. If no name is specified (as in this example), the definition is associated with the default include set. An include set defines a search path for all includes associated with that include set. Multiple include set s can be specified in a language definition if the parser and compiler support distinguishing one kind of include from another. For the parser, this means that the syntax of the language must support determining which include set an include belongs to. For the compiler, this means that a separate ddname must be used for each different include set (kind of include).

The following parameter is used:

TYPES This specifies the name(s) of the types which are searched to find includes. In this case, SCLM searches the MACROS type first, and next the default @@FLMTYP type. The @@FLMTYP SCLM variable indicates that SCLM also needs to search the type of the member that is processed by the High Level Assembler. For example, if 'SCLM01.DEV2.SOURCE(FLM01MD1)' is being compiled, SCLM looks for macros first in the data sets associated with the MACROS type, and next the SOURCE type.

Step 4: Specify the programs that process the modules

Next, identify the programs that are used to parse and build the High Level Assembler modules. There are usually two such programs: a parser and an assembler. For each of these programs, code an FLMTRNSL macro and the appropriate FLMALLOC macros and FLMCPYLB macros.

The High Level Assembler parser is FLMLPGEN, and it resides in the ISPF load library ISP.SISPLPA. The parser requires five option parameters, SOURCEDD=SOURCE, STATINFO=@@STP, LISTINFO=@@LIS, LISTSIZE=@@SIZ and LANG=A, and reads the source from ddname SOURCE.

The parse translator is shown Figure 2-49.

```
*
          FLMTRNSL   CALLNAM='SCLM HLAS PARSE',                        C
               FUNCTN=PARSE,                                           C
               COMPILE=FLMLPGEN,                                       C
               PORDER=1,                                               C
               OPTIONS=(SOURCEDD=SOURCE,                               C
               STATINFO=@@FLMSTP,                                      C
               LISTINFO=@@FLMLIS,                                      C
               LISTSIZE=@@FLMSIZ,                                      C
               LANG=A)              *** THIS IS ASSEMBLER ONLY ***
```

Figure 2-49 FLMTRNSL macro specification for Enterprise PL/I parser

The following parameters are used in this example:

CALLNAM This is a character string that appears in messages during the specified FUNCTN (in this case PARSE). This value will assist in recognizing which translator was executing during the specified FUNCTN.

FUNCTN The value PARSE tells SCLM that this program is to be invoked whenever you save a module with LANG=PLIE.

COMPILE Member name of the load module for the High Level Assembler parser. Note that the keyword COMPILE actually identifies the load module name of a translator (which may or may not be a compiler).

PORDER The value 1 tells SCLM that this program expects an options string but not a ddname substitution list.

OPTIONS This specifies the options string to be passed to the parser. Strings that start with @@FLM are SCLM variables, and they are replaced by their current values before the string is passed to the parser.

The LANG=A options parameter tells the parser that this is a PL/I member. FLMLPGEN is a general purpose parser, and hence, needs to know whether this is an assembler, PL/I, REXX or text member.

CALLMETH The CALLMETH parameter is not specified so the default value of ATTACH specifies the invocation method of the module specified in the COMPILE parameter.

DSNAME DSNAME is not used in this example, but should be mentioned. DSNAME identifies the partitioned data set that contains the High Level Assembler parser load module. DSNAME is required only when the data set containing the desired module is not in the system's ISPLLIB, STEPLIB or LINKLIST concatenations. FLMLPGEN is in ISP.SISPLPA which is defined in the LINKLIST concatenation, so DSNAME is not required in this case. When more than one data set is to be searched, you can use the TASKLIB parameter in conjunction with, or as a replacement for, the DSNAME parameter.

Since the parser reads its source from a ddname, you must tell SCLM how to allocate that ddname; we use an FLMALLOC macro and an FLMCPYLB macro as shown in Figure 2-50.

```
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
```

Figure 2-50 FLMALLOC and FLMCPYLB macro specifications for parser

A description of the parameters follows:

IOTYPE=A The value A tells SCLM to allocate a ddname to one or more specific data set(s). Each of those data sets are subsequently identified by using an FLMCPYLB macro.

DDNAME This identifies the ddname to be allocated.

@@FLMDSN(@@FLMMBR) This identifies the member to be parsed. When the two SCLM variables are resolved, the parser is passed the data set and member that is being saved.

Now you can tell SCLM how to invoke the High Level Assembler. To do so, use an FLMTRNSL macro, as shown in Figure 2-51, followed by one or more FLMALLOC and FLMCPYLB macros. The High Level Assembler is ASMA90, and on the system we used it resides in the assembler load library ASM.SASMMOD1.

```
*
          FLMTRNSL   CALLNAM='HL ASM',                                  C
               FUNCTN=BUILD,                                            C
               COMPILE=ASMA90,                                          C
               VERSION=1.5,                                             C
               GOODRC=0,                                                C
               PORDER=1,                                                C
               OPTIONS=(XREF(SHORT),LINECOUNT(75),OBJECT,RENT)
```

Figure 2-51 FLMTRNSL macro specification for High Level Assembler

You can specify as many parameters as required, and let SCLM supply default values for the others:

CALLNAM This is a character string that appears in the build messages. It names the High Level Assembler.

FUNCTN This tells SCLM that this program gets invoked whenever you want to build a member with LANG=HLAS.

COMPILE This identifies the load module name for the High Level Assembler.

VERSION Version of the compiler used for information purposes. This value is stored in the account record for any generated outputs.

GOODRC The value 0 indicates that SCLM is to consider this build unsuccessful if the compiler completes with a return code greater than 0, meaning only informational are acceptable.

PORDER The value 1 tells SCLM that this program expects an options string but not a ddname substitution list.

OPTIONS This specifies the options string to be passed to the compiler.

DSNAME Like the parser translator the DSNAME is not used in this example, so the High Level Assembler load module, ASMA90, will be located in the system's ISPLLIB, STEPLIB or LINKLIST concatenations. ASMA90 is in ASM.SASMMOD1 which is defined in the LINKLIST concatenation, so DSNAME is not required in this case. When more than one data set is to be searched, you can use the TASKLIB parameter in conjunction with, or as a replacement for, the DSNAME parameter.

Let us now examine the ddnames used by the High Level Assembler in detail as specified in the sample as provided. As we have used PORDER=1 we have allocated the minimum ddnames that are required.

Allocate SYSIN DD to the source to be compiled

Figure 2-52 shows the FLMMALLOC macro specification for SYSIN DD.

```
*
           FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=9000
```

Figure 2-52 FLMALLOC macro specification for SYSIN DD

The following parameters are used in this example:

IOTYPE=S The value S tells SCLM to allocate a temporary sequential data set and create the input stream for the translator by concatenating the contents of all the members that are SINCed as well as any text specified via CMD cards.

DDNAME This identifies the ddname to be allocated. In this case SYSIN.

KEYREF The value SINC tells SCLM that, if you are building a source module directly, SCLM copies that member to this temporary data set. If you are building a CC architecture definition, SCLM copies the members listed on the SINC statement to this data set.

Allocate SYSUT1 DD to temporary work data sets

The High Level Assembler requires a temporary work data set to be allocated as shown in Figure 2-53.

```
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,RECNUM=15000
```

Figure 2-53 FLMALLOC macro specification for SYSUT1 work DD

The following parameters are used in this example:

IOTYPE=W The value W tells SCLM to allocate a temporary sequential data set for translator use.

DDNAME This identifies the ddname to be allocated. In this case SYSUT1.

RECNUM The value 15000 tells SCLM to allocate enough space in this data set to hold 15000 records.

Allocate SYSLIN DD to the object module

Figure 2-54 shows the FLMALLOC macro specification for SYSLIN DD.

```
*
           FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,                C
              RECNUM=9000,DFLTTYP=OBJ
```

Figure 2-54 FLMALLOC macro specification for SYSLIN DD

The following parameters are specified in this macro:

IOTYPE=O The value O tells SCLM to allocate a temporary sequential data set to ddname SYSLIN. After the language definition completes successfully the temporary sequential data set is copied as a member in the SCLM hierarchy

DDNAME This identifies the ddname to be allocated. In this case SYSLIN.

KEYREF The value OBJ tells SCLM to save what is written to this ddname and keep it under SCLM control, SCLM must be able to determine the member name and the SCLM-controlled data set name in which it is to save this output module. If SCLM is building an architecture definition, it determines the project, group, type and member as follows:

The high-level qualifier is the project identifier that was previously specified.

The group is the level at which the build is taking place. The group name is the second qualifier.

SCLM looks at the architecture definition being built and retrieves the member and type from the architecture statement associated with the keyword OBJ. The type name is the third qualifier.

The construction of the PDS name however still follows the defined rules in FLMCNTRL or FLMALTC based on the SCLM identifiers @@FLMPRJ, @@FLMGRP and @@FLMTYP.

DFLTTYP The value OBJ tells SCLM to save what is written to this ddname to the SCLM type OBJ and keep it under SCLM control. If SCLM is building a source member, it determines the project, group, type and member:

The high-level qualifier is the project identifier that was previously specified.

The group is the level at which the build is taking place.

The SCLM type is the value of the DFLTTYP= keyword.

The member name defaults to the name of the member being built.

If SCLM is building an architecture definition (and not a source member directly) then the DFLTTYP= value is ignored. Instead, SCLM uses the type associated with the KEYREF= value.

RECNUM The value 9000 tells SCLM to allocate enough space in this data set to hold 9000 records.

Allocate SYSTERM and SYSPUNCH DDs to dummy data sets

These data sets are not needed for any output, so they can be allocated to a dummy data set as shown in Figure 2-55.

```
*
           FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
            FLMCPYLB NULLFILE
*
           FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
            FLMCPYLB NULLFILE
```

Figure 2-55 FLMALLOC macro specifications for SYSTERM and SYSPUNCH DDs

The following parameters are used in this example:

IOTYPE=A The value A tells SCLM to allocate a ddname to the data set identified by the next FLMCPYLB macro.

DDNAME This identifies the ddname to be allocated. In this case SYSTERM and SYSPUNCH.

NULLFILE This allocates the specified DD to a dummy data set.

Allocate SYSLIB DD to one or more partitioned data sets

High Level Assembler uses this ddname to find included members. The FLMINCLS macro described earlier needs to be referenced here to ensure that the compiler is picking up includes from the correct data sets. Since IOTYPE=I allocations default to the default include set shown earlier, this is automatically done. If another name was used on the FLMINCLS macro, that name needs to be referenced here using the INCLS parameter. IOTYPE=I allocates a ddname with a concatenation of all the PDSs in the hierarchy starting with the group specified for the BUILD and ending with the top, or production level, group. First the hierarchy for the INCLUDE type is allocated, followed by the type of the first SINCed member from the architecture definition, or, if no architecture definition is used, the type of the member being built. This is shown in Figure 2-56.

```
*
          FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
```

Figure 2-56 FLMALLOC macro specification for SYSLIB DD

The following parameters are used in this example:

IOTYPE=I The value I tells SCLM to allocate this ddname to a concatenation of SCLM-controlled data sets. The types used in the concatenation are determined by the FLMINCLS macro referenced by the INCLS= parameter on the FLMALLOC macro. In this case, there is no INCLS= parameter so the default FLMINCLS (or include set) is used.

A hierarchy of data sets is concatenated for each type specified for the referenced FLMINCLS macro. The hierarchy begins at the group where the build is taking place and extends to the top of the project's hierarchy.

In this case, the concatenation first contains all of the data sets for the INCLUDES type followed by the data sets for the value substituted into the @@FLMTYP variable. See the KEYREF= parameter to determine the value which is substituted into the @@FLMTYP and @@FLMETP variables.

DDNAME This identifies the ddname to be allocated. In this case SYSLIB.

KEYREF The value SINC tells SCLM that, if you are building an architecture definition, refer to the first SINC statement in that architecture definition for the type that is substituted into the @@FLMTYP macro. The value for @@FLMETP comes from the EXTEND= parameter of the FLMTYPE macro for that type. If you are not building an architecture definition, the type is the type of the member being built.

Allocate SYSPRINT DD to the compiler listing

Figure 2-57 shows the FLMALLOC macro specification for SYSPRINT DD.

```
 *
          FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,KEYREF=LIST,PRINT=Y,     C
             RECFM=FBA,LRECL=121,                                     C
             DFLTTYP=LIST,RECNUM=20000
```

Figure 2-57 FLMALLOC macro specification for SYSPRINT DD

This definition contains the following parameters:

IOTYPE=O The value O tells SCLM to allocate a temporary sequential data set to ddname SYSPRINT. After the language definition completes successfully the temporary sequential data set is copied as a member in the SCLM hierarchy

DDNAME This identifies the ddname to be allocated. In this case SYSPRINT.

KEYREF The value LIST refers SCLM to the LIST record in the architecture definition being built. That record contains the member name and type into which the listing is saved after a successful build. (SCLM copies the data from the temporary data sets into members of the PDSs controlled by SCLM after a successful build.)

PRINT The value Y specifies that this is a listing that should be copied to the Build List data set after the build process completes.

RECFM The value FBA specifies the record format of the temporary data set that SCLM creates. In this example, the record format is fixed blocked with an ASA control character.

LRECL The value 121 specifies the record length, in characters, of the temporary data set that SCLM creates.

DFLTTYP The value LIST specifies the data set type into which this listing is written whenever a module is built directly or when using INCLD in an architecture definition.

RECNUM The value 20000 tells SCLM to allocate enough space in this data set to hold 20000 records.

Language translator summary

These macros together constitute a language definition similar to one in Appendix A, “Chapter 1 listings” on page 691, SCLM01.PROJDEFS.SOURCE(FLM@HLAS). It contains comments to explain the flow of operations. When you are ready to reassemble your project definition, add a COPY statement in your main project definition file to include the this language definition.

2.2.7 z/OS binder/linkage editor

The following describes the steps to setup a language definition for the binder/linkage editor using the supplied sample FLM@L370 as a template.

Step 1: Define the language

First, we tell SCLM that you are defining a new language. To do so, code the following FLMLANGL macro as shown in Figure 2-58.

```
*
         FLMLANGL    LANG=LE370,CANEDIT=N,VERSION=D071108,              C
               LANGDESC='370/LINKAGE EDITOR'
```

Figure 2-58 FLMLANGL macro specification for z/OS Binder/Linkage Editor

In this example, values are specified for four parameters. Defaults are used for the other parameters.:

LANG The name of the language translator SCLM will use when it encounters a linkedit control (LEC) ARCHDEF. The default language in SCLM will use is LE370 unless there is a LKED keyword in the LEC ARCHDEF. If there is a LKED keyword specifying an alternative language translator then there must be a language translator defined with a LANG parameter that matched the language specified by the LKED keyword.

CANEDIT The value N indicates that the linkage editor language cannot be assigned to editable members. (The default is Y.)

VERSION Use up to 8 characters to identify the specific language version with a certain release of the current binder/linkage editor. If you install a new release or version of the binder/linkage editor, you can set this parameter to a different value so that SCLM can mark all LEC ARCHDEFS as needing to be rebuilt. You must then re-assemble and re-link your project definition. A suggested format is *Dyymmdd* to indicate the date the translator was changed.

LANGDESC This is a description of the language, maximum 40 characters.

Notice the continuation character in column 72, in this case a C. If the parameters for a macro will spread beyond 1 line, then a continuation character must be used.

Step 2: Specify the program (linkage editor) that processes the modules

Next, tell SCLM how to invoke the z/OS Binder (linkage editor). To do so, use an FLMTRNSL macro, as shown in Figure 2-59, followed by FLMALLOC and FLMCPYLB macros. The z/OS binder is IEWL, and it resides in the system load library SYS1.LINKLIB.

```
*
          FLMTRNSL   CALLNAM='LKED/370',                              C
               FUNCTN=BUILD,                                          C
               COMPILE=IEWL,                                          C
               VERSION=F64,                                           C
               GOODRC=0,                                              C
               PORDER=3,                                              C
               OPTIONS=(DCBS,MAP)
```

Figure 2-59 FLMTRNSL macro specification for High Level Assembler

You can specify as many parameters as required, and let SCLM supply default values for the others:

CALLNAM This is a character string that appears in the build messages. It names the z/OS Binder.

FUNCTN This tells SCLM that this program gets invoked whenever you want to build an LEC ARCHDEF that invokes the standard linkage editor (LANG=LE370).

COMPILE This identifies the load module name for the z/OS Binder.

VERSION Version of the compiler used for information purposes. This value is stored in the account record for any generated outputs.

GOODRC The value 0 indicates that SCLM is to consider this build unsuccessful if the compiler completes with a return code greater than 0, meaning only informational are acceptable.

PORDER The value 3 tells SCLM that this program expects an options string and a ddname substitution list to the z/OS Binder. See 2.2.1, “Using ddnames and ddname substitution lists” on page 41 for more information on ddname substitution lists.

OPTIONS This specifies the options string to be passed to the z/OS Binder.

DSNAME The DSNAME is not used in this example, so the z/OS Binder module, IEWL, will be located in the system’s ISPLLIB, STEPLIB or LINKLIST concatenations. IEWL is in SYS1.LINKLIB which is defined in the LINKLIST concatenation, so DSNAME is not required in this case.

Let us now examine the ddnames used by the z/OS Binder in detail as specified in the sample as provided. As we have used PORDER=3 we have to allocate all of the ddnames expected by the z/OS binder.

As discussed earlier in 2.2.1, “Using ddnames and ddname substitution lists” on page 41, when you use a ddname substitution list, you must define the ddnames in the order in which they are expected to appear in the ddname substitution list by the translator. The compiler uses temporary ddnames created by SCLM, instead of the standard documented ddnames, such as SYSIN. However in the sample translator as supplied, many of the actual ddnames are specified, so will be allocated with those ddnames.

Archived

Allocate SYSLIN DD to the object module

The first ddname in the z/OS Binder's ddname substitution list is SYSLIN. It is allocated to a partitioned data set from which the binder retrieves the object module as shown in Figure 2-60.

```
*  1       (* SYSLIN *)
           FLMALLOC  IOTYPE=S,KEYREF=INCL,RECFM=FB,LRECL=80,          C
               RECNUM=20000,DDNAME=SYSLIN
```

Figure 2-60 FLMALLOC macro specification for SYSLIN DD

The following parameters are specified in this macro:

IOTYPE=S The value S tells SCLM to allocate a temporary sequential data set and create the input stream for the translator by concatenating the contents of all the members that are SINCed as well as any text specified via CMD cards. This will include all OBJ input to the z/OS Binder.

KEYREF INCL generates a linkage editor include statement for each OBJ member or load module referenced in the link-edit control (LEC) architecture definition.

RECFM The value FB specifies the record format of the temporary data set that SCLM creates. In this example, the record format is fixed blocked.

LRECL The value 80 specifies the record length, in characters, of the temporary data set that SCLM creates.

RECNUM The value 20000 tells SCLM to allocate enough space in this data set to hold 20000 records.

DDNAME This identifies the ddname to be allocated. In this case SYSLIN. The DDNAME parameter could be omitted as we are using PORDER=3 and we could default the ddname to an SCLM generated ddname.

Allocate load module name to the load member being created

The second ddname in the ddname substitution list is the load module name. It resides in a partitioned data set allocated in the next ddname as shown in Figure 2-61.

```
*  2       (* LOAD MODULE NAME *)
           FLMALLOC  IOTYPE=L,KEYREF=LOAD
```

Figure 2-61 FLMALLOC macro specification for load module name

The following parameters are used in this example:

IOTYPE This passes a member name in the ddname substitution list. The KEYREF parameter identifies the member name and type.

This IOTYPE is commonly used to identify the load module name for the S/370™ linkage editor.

IOTYPE=L is only valid when the PORDER parameter in the FLMTRNSL macro is set to 2 or 3.

KEYREF When you build an LEC architecture definition, the FLMALLOC macro with IOTYPE=L passes the member name referenced on the LOAD statement to the Linkage Editor when KEYREF of LOAD is specified.

Allocate SYSLMOD load module name to the destination load library

The next ddname in the ddname substitution list is SYSLMOD. It is allocated to a partitioned data set into which the binder saves the load module as shown in Figure 2-62.

```
*  3       (* SYSLMOD *)
           FLMALLOC  IOTYPE=P,KEYREF=LOAD,RECFM=U,LRECL=0,            C
               RECNUM=500,DIRBLKS=20,DDNAME=SYSLMOD
```

Figure 2-62 FLMALLOC macro specification for SYSLMOD DD

The following parameters are used in this example:

IOTYPE The value of P allocates a temporary partitioned data set to contain output from the translator. After the language definition completes successfully the temporary partitioned data set member is copied as a member in the SCLM hierarchy.

KEYREF The value of LOAD identifies the target member into which the translator output will be copied.

RECFM The value of U specifies the record format of the temporary load library data set that SCLM creates. In this example, the record format is undefined.

LRECL The value of 0 specifies the record length, in characters, of the temporary data set that SCLM creates.

RECNUM The value of 500 tells SCLM to allocate enough space in this data set to hold 500 records.

DIRBLKS The value of 20 tells SCLM to allocate 20 directory blocks for the temporary load library.

DDNAME This identifies the ddname to be allocated. In this case SYSLMOD. The DDNAME parameter could be omitted as we are using PORDER=3 and we could let SCLM default the ddname to an SCLM generated ddname.

Allocate SYSLIB DD to resolve external load modules dependencies

The ddname in position 4 of the ddname substitution list must be allocated to one or more specific partitioned data sets. The z/OS Binder uses these ddnames to find included load module members as shown in Figure 2-63.

```
*  4       (* SYSLIB  *)
           FLMALLOC  IOTYPE=A,DDNAME=SYSLIB
            FLMCPYLB CEE.SCEELKED
            FLMCPYLB CICS.TS31.CICS.SDFHLOAD
            FLMCPYLB DB2.V810.SDSNLOAD
            FLMCPYLB SYS1.LINKLIB
*
```

Figure 2-63 FLMALLOC macro specification for SYSLIB DD

A description of the parameters follows:

IOTYPE The value of A tells SCLM to allocate a ddname to one or more specific data set(s). Each of those data sets are subsequently identified by using an FLMCPYLB macro.

Archived

DDNAME This identifies the ddname to be allocated. In this case SYSLIB. The DDNAME parameter could be omitted as we are using PORDER=3 and we could default the ddname to an SCLM generated ddname.

Customize FLMCPYLB macros for the external data sets used at your location. Also, if you are linking in any third party load modules, include FLMCPYLB macros for the data sets where these load libraries are stored.

Skip over position 5

The ddname in position 5 of the ddname substitution list is not used. Create an FLMALLOC macro with IOTYPE=N to tell SCLM to fill this name field with hex zeros and to continue to the next ddname as shown in Figure 2-64.

```
*  5       (* N/A *)
           FLMALLOC  IOTYPE=N
```

Figure 2-64 FLMALLOC macro specifications to skip over position 5

Allocate ddname SYSPRINT to the compiler listing

The ddname in position 6 of the ddname substitution list must be allocated to a DD that will contain the listing output from the z/OS binder as shown in Figure 2-65.

```
 *  6       (* SYSPRINT *)
           FLMALLOC  IOTYPE=O,KEYREF=LMAP,RECFM=FBA,LRECL=121,         C
              RECNUM=2500,PRINT=Y,DDNAME=SYSPRINT
```

Figure 2-65 FLMALLOC macro specification for SYSPRINT DD

This definition contains the following parameters:

IOTYPE=O The value O tells SCLM to allocate a temporary sequential data set to ddname SYSPRINT. After the language definition completes successfully the temporary sequential data set is copied as a member in the SCLM hierarchy

KEYREF The value LMAP Refers SCLM to the LMAP record in the architecture definition being built. That record contains the member name and type into which the listing is saved after a successful build. (SCLM copies the data from the temporary data sets into members of the PDSs controlled by SCLM after a successful build.)

RECFM The value FBA specifies the record format of the temporary data set that SCLM creates. In this example, the record format is fixed blocked with an ASA control character.

LRECL The value 121 specifies the record length, in characters, of the temporary data set that SCLM creates.

RECNUM The value 2500 tells SCLM to allocate enough space in this data set to hold 2500 records.

PRINT The value Y specifies that this is a listing that should be copied to the Build List data set after the build process completes.

DDNAME This identifies the ddname to be allocated. In this case SYSPRINT.

Archived

Skip over positions 7 through 11

The ddnames in position 7 through 11 of the ddname substitution list are not used. Create an FLMALLOC macro with IOTYPE=N to tell SCLM to fill these name fields with hex zeros and to continue to the next ddname as shown in Figure 2-66.

```
*
*  7       (* N/A *)
           FLMALLOC  IOTYPE=N
*
*  8       (* N/A *)
           FLMALLOC  IOTYPE=N
*
*  9       (* N/A *)
           FLMALLOC  IOTYPE=N
*
* 10       (* N/A *)
           FLMALLOC  IOTYPE=N
*
* 11       (* N/A *)
           FLMALLOC  IOTYPE=N
```

Figure 2-66 FLMALLOC macro specifications to skip over positions 7 through 11

Allocate SYSTERM DD to dummy data set

This data set is not needed for any output, so it can be allocated to a dummy data set as shown in Figure 2-67.

```
*
           FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
            FLMCPYLB NULLFILE
```

Figure 2-67 FLMALLOC macro specifications for SYSTERM DD

The following parameters are used in this example:

IOTYPE=A The value A tells SCLM to allocate a ddname to the data set identified by the next FLMCPYLB macro.

DDNAME This identifies the ddname to be allocated. In this case SYSTERM.

NULLFILE This allocates the specified DD to a dummy data set.

Language translator summary

These macros together constitute a language definition similar to one in Appendix A, “Chapter 1 listings” on page 691, SCLM01.PROJDEFS.SOURCE(FLM@L370). It contains comments to explain the flow of operations. When you are ready to reassemble your project definition, add a COPY statement in your main project definition file to include the this language definition.

Archived

2.2.8 Summary

Do the following steps to customize an existing SCLM language definition, or to write a new SCLM language definition:

1. Define the language name to the SCLM project definition.
2. Define include-sets for the language to identify the locations of included members. For external libraries use the FLMSYSLB macro before the FLMLANGL macro.
3. List the various programs (parsers, compilers, and so on) used to parse, build your source or issue an action during promote. Use the correct FUNCTN option on the FLMTRNSL macro.
4. For each program (or translator), look up the ddname substitution list (usually in the Programmer's Guide for the compiler), or list the ddnames used by the program.
5. Write an FLMTRNSL macro for each program or translator.
6. Write an FLMALLOC macro for each ddname to be allocated to the translator.
7. Use the information in the translator program documentation to determine which IOTYPE value to specify as well as which other FLMALLOC keywords are appropriate.

2.3 Writing build/copy processors

In 2.2, “Language definitions based upon samples ” on page 40, we discuss how SCLM supports traditional translators, such as z/OS compilers and linkage editors. SCLM also supports user-written translators. You can customize SCLM to interface with objects which it cannot control directly, through user-written translators. In this section, we talk about how you can extend SCLM to support some of the advanced user requirements using this capability.

Let us begin by becoming familiar with all types of translators SCLM supports.

2.3.1 Types of translators

You have seen some of these types of translators in the previous sections relating to the traditional translator for the z/OS compilers, in particular PARSE and BUILD.

You can invoke a software process as an SCLM translator to perform the following SCLM functions:

PARSE A parse translator is called during the SAVE action and may gather statistics and dependencies. Parse translators run during migration, when the member is saved in an edit session, or when the SAVE or PARSE service is called. You can also use a parse translator to define user data and change codes for the member.

VERIFY A verify translator performs validation in addition to default SCLM validation. The verify translator can be used to check the change codes or user data defined for members. Another example could be verification of data that is related to an SCLM-controlled member but is not under SCLM control itself. Verify translators run during build and promote verification.

For builds, SCLM invokes a verify translator to verify inputs to build translators. For example, when an LEC architecture definition is being built, the source member is verified before compiling and the object member is verified before linking.

Archived

For promotes, SCLM invokes a verify translator to verify build inputs and outputs. For example, when an LEC architecture definition is being promoted, the source, object, and load members are verified before the promote copy phase.

BUILD A build translator can assemble, compile, link, or otherwise process a member so that the outputs have different formats than the inputs. For example, building a COBOL source program generates a listing and an object module.

COPY A copy translator is invoked when Promote copies an SCLM-controlled member to the next group in the hierarchy. Copy translators are invoked before Promote copies the SCLM-controlled member. If the copy translators for a member fail, Promote does not attempt to copy the controlled member. Copy translators can be used to copy data that is related to an SCLM-controlled member but is not under SCLM control itself.

PURGE Purge translators can be used to purge data that is related to an SCLM-controlled member but is not under SCLM control itself. Purge translators are invoked whenever SCLM performs a delete operation on an SCLM-controlled member during build or promote.

As described in 2.2, “Language definitions based upon samples ”, you invoke SCLM translators through the FLMTRNSL language definition macro. You provide the required information about invocation of the translator to SCLM through various parameters, such as its function, location, calling method, and so on.

2.3.2 Developing new translators: Introduction

Let us now talk about how you can write your own translators and execute them under SCLM for the specific needs of your site and environment. To illustrate the process for writing new SCLM translators, we present here an example scenario in which we developed a couple of simple REXX translators, and developed or tailored sample language definitions to invoke them.

Note: The purpose of the “Develop New Translators” example discussed in the next section is to illustrate the process of writing new SCLM translators and not to recommend one or the other approach to configure the Debug Tool for z/OS. There is a complete chapter on discussions to determine the most appropriate implementation of Debug Tool with SCLM in Chapter 14, “Debugging and fault analysis with SCLM” on page 301

2.3.3 Developing new translators: Example problem description

Let us assume that your site has IBM Debug Tool for z/OS, Debug Tool Utilities and Advanced Functions and IBM Fault Analyzer installed. You want to Share common Debug Tool side files with Fault Analyzer. Therefore, you decide to keep the side file out of the control of SCLM. However, you must still ensure that the side file which is generated by the Enterprise COBOL compiler moves up your overall configuration management hierarchy, when you promote your COBOL source module. Also, you do not want to delete the side file from the lower level group at promotion time.

2.3.4 Developing new translators: Example problem analysis

Consider the following factors to arrive at an optimum solution for this requirement:

- When Enterprise COBOL invokes Debug Tool through the TEST(ALL,SYM,SEP) option, it places the output SYSDEBUG side file in an RECFM=FB,LRECL=1024 data set.
- You allocate a PDS or a PDSE for the SYSDEBUG side file with above attributes and sufficient DASD space and directory blocks (PDS only), along with the rest of your project data sets.
- The normal process is to place the SYSDEBUG side file under SCLM control. However, you need to keep it outside SCLM control due to the unique requirement of your project.
- When you build a source module, you must capture the SYSDEBUG output for the source module into the PDS/PDSE member of the build group with the same name as the source module.
- When you promote a source module, you must replicate the SYSDEBUG side file for the source module into the PDS/PDSE member of the promote group with the same name as the source module.
- You must ensure that the SYSDEBUG side file stays in sync with the COBOL source member at all times, even though it is not under SCLM control.
- Unlike JCL and TSO, the SCLM FLMALLOC macro does not create a new PDS/PDSE member, when allocated with DISP=SHR. Instead, it returns an error condition, if the member does not exist already.

2.3.5 Developing new translators: Example technical approach

The above discussion is likely to lead you to develop a technical approach for your requirement as follows:

- Verify that a PDS/PDSE Member with DISP=SHR is allocated for the SYSDEBUG side files at build and promote groups.
- Perform a TSO allocate for a PDS/PDSE Member with DISP=SHR for each of the SYSDEBUG side files. The TSO allocate creates the SYSDEBUG member, if not already allocated, to prepare for build and promote of the source module.
- Develop a new REXX build translator to perform the SYSDEBUG verification and allocation functions.
- Have the SCLM build translator, Enterprise COBOL compiler with Debug, generate the SYSDEBUG output side file for the source module, and capture it into the PDS/PDSE member of the build group with the same name as the source module.
- Also, have the Enterprise COBOL translator change the language of the output compiler listing data set to a language associated with the SCLM promote copy translator.
- Enhance the SCLM Enterprise COBOL language definition to call the verification and allocation REXX translator for the build group, to capture the output SYSDEBUG side file of the Enterprise COBOL compiler and to change the language of the compiler listing data set.
- Develop a new REXX promote copy translator to replicate the SYSDEBUG side file for the source module from the build group into the PDS/PDSE member of the promote group with the same name as the source module. Use the IEBCOPY utility to copy the SYSDEBUG side file member from the build group to the promote group.
- Develop a new SCLM LIST language definition to invoke new promote verification and promote copy translators to replicate the SYSDEBUG side file into the PDS/PDSE member of the promote group.

2.3.6 Developing new translators: Example source code

We are now ready to present the source code of the new REXX build and promote copy translators, and the new and enhanced language definitions to support new translators.

ALLOCDBG - Allocate Debug Side File

This REXX translator allocates a PDS/PDSE member to the SYSDEBUG side file with DISP=SHR. When invoked from a build translator, it allocates the member at the build group. When invoked from a promote copy translator, it allocates the member at the promote group.

See Appendix B-3, “ALLOCDBG Listing” on page 718 for the listing of ALLOCDBG.

COPYMEMB - Replicate Debug Side File to Promote Group

This REXX translator replicates (copies) a SYSDEBUG side file PDS/PDSE member with DISP=SHR from the build group to the promote group. You invoke it from a promote copy translator. It calls the IEBCOPY utility to perform the copy of the PDS/PDSE member.

See Appendix B-4, “COPYMEMB Listing” on page 719 for the listing of COPYMEMB.

LIST - Invoke Promote Copy Translators

This new language definition invokes new REXX promote copy translators , ALLOCDBG (promote group) and COPYMEMB.

See Appendix B-5, “LIST Language Definition” on page 719 for the LIST Language Definition.

COBDTC (Enhanced FLM@COBE) Language Definition

This enhanced language definition invokes the new REXX build translator ALLOCDBG to allocate the SYSDEBUG side file for the build group. It also assigns a new language LIST to the Enterprise COBOL listing file to trigger invocation of the ALLOCDBG and COPYMEMB promote copy translators.

See Appendix B-6, “COBDTC Enhanced Language Definition” on page 720 for the COBDTC Enhanced Language Definition.

Archived

Archived

3

Writing DB2 translators and bind processing

SCLM has robust support for applications using DB2. You can create DB2 Database Request Modules (DBRMs), as well as bind DB2 application packages and application plans using SCLM.

The language translator you use to compile your SQL source program creates the DBRM. The DB2 precompiler DSNHPC interprets the SQL, creates the DBRM, and passes the modified source through to the compiler. The DB2 coprocessor for Enterprise COBOL and Enterprise PL/I, when invoked from the compiler, interprets the SQL and creates the DBRM. The DBRM type will be an SCLM defined type. This means that the build map for the source member will contain outputs for DBRM as well as OBJ and LIST.

The key to the SCLM processing is how to let SCLM know that a DB2 bind is required when a module containing embedded SQL statements is built. This is handled through special purpose translators and sample language definitions shipped with SCLM.

© Copyright IBM Corp. 2007. All rights reserved.

3.1 Overview

SCLM supports DB2 processing through Bind control objects that are referred to as DB2CLIST objects. This terminology is historical, as when DB2 support was added to SCLM, CLIST was the interpretive language of choice. These days most sites use REXX. For this chapter we continue to use the term DB2 CLIST for consistency between this Redbooks publication and the *SCLM Guide and Reference*, even though we will use REXX in our examples. A DB2 CLIST specifies DBRM(s) to be bound into application plans or collections. SCLM maintains accounting information for the DB2 CLIST for DBRM dependencies captured through an include statement for each DBRM in the DBRM comment block.

During Build, the DB2 CLIST member Binds DBRM(s) in the “current” SCLM group. A non-editable copy of the DB2 CLIST is placed in the type to be used during Promote. During the Promote Copy phase, the non-editable copy of the DB2 CLIST binds in the “to” group. During the Promote Purge phase, it frees in the “from” group.

3.1.1 Bind control object (DB2 CLIST)

To create the link between the compilable object, namely the SQL source program, the load module, and the application plan or package, we must create a DB2CLIST which references each bindable object, namely DBRM.

The DB2CLIST specifies the names of the DBRMs to be bound. It sets up any variables required for the bind and then either performs the bind directly or calls an external bind processor to take the parameters and perform the bind. The DB2CLIST is under SCLM control, contains accounting information and is buildable.

For example, if your site binds DBRMs directly to application plans, the DB2CLIST contains entries for all DBRMs bound to the application plan. If your site binds DBRMs to collections to produce application packages, the DB2CLIST contains just one DBRM entry for the application package. These scenarios are discussed with examples in subsequent sections of this chapter.

You can reference the DB2CLIST from architecture definitions. The link edit control (LEC) architecture definitions control compilation and linking of source objects. Generic architecture definitions control binding of DBRMs through DB2CLIST objects. You can link both types of architecture definitions by using High Level (HL) architecture definitions

3.1.2 Processing of DB2 SQL program in SCLM

The processing of a program containing embedded SQL in SCLM has the following stages:

1. Create an LEC architecture definition and optionally a CC architecture definition for the program. The architecture definition invokes the appropriate build translators for the program, such as DB2 precompiler, Enterprise COBOL compiler, and z/OS linkage editor. It creates an object module, a load module, and a DBRM as its non-editable outputs.
2. During the Editing stage, you create a DB2CLIST object as described earlier in this section. When the DB2CLIST object is parsed, the DBRMs to be bound are identified by the %INCLUDE statement in the comment block and an entry is placed in the accounting information for the DB2CLIST. SCLM then knows that when that particular DBRM is recreated during compilation, it needs to rebuild the DB2CLIST member.

Archive

3. The BUILD process executes the DB2CLIST member to perform the appropriate BIND or FREE DB2 operation. The process creates an identical copy of the DB2CLIST, referred to as a DB2OUT member, and places it in the type that is used during the PROMOTE process. The DB2OUT is an SCLM output generated by the DB2 bind/free translator and, therefore, is non-editable.
4. The only difference between the original DB2CLIST and the new DB2OUT is the language value. The language for the original DB2CLIST is associated with a language definition that contains the parse and build translators. The SCLM sample for this language definition, ISP.SISPMACS(FLM@BD2), names it DB2CLIST. The language for the new DB2OUT is associated with a language definition that contains the copy and purge translators. The SCLM sample for this language definition, ISP.SISPMACS(FLM@BDO), names it DB2OUT.
5. On promote, SCLM does not execute the DB2CLIST but only promotes it. However, on promote, SCLM executes the COPY and PURGE phases of the translators and as such will run the contents of the DB2OUT member. This is why the DB2 translator creates a DB2OUT. Promote processes the DB2OUT in exactly the same way that BUILD processes the DB2CLIST. On the COPY phase the DB2OUT will be executed with the BIND option and on the PURGE phase the DB2OUT will be executed with the FREE option if you wish to free the plan or package at the from-group level.

In your high-level architecture definitions, always refer to the DB2CLIST used during the Build stage. Do not refer to the DB2OUT used during the Promote stage.

3.2 Enabling DB2 support for your SCLM project

As discussed in Chapter 1, “SCLM project setup” on page 3, the primary function of the SCLM project administrator is to create and manage the project environment. Let us now discuss in-depth what you need to do to enable DB2 support for your SCLM project. Next, we discuss the DB2 bind/free translator and sample DB2-specific language definitions shipped with SCLM in detail.

In this chapter, we present the approach to configure SCLM DB2 support using the SCLM01 sample project built in Chapter 1, “SCLM project setup” on page 3. SCLM01 builds an SCM environment for three simple MVS batch applications, one each developed in Enterprise COBOL, z/OS high level assembler and Enterprise PL/I. We configure DB2 support for this project by adding four MVS batch and online applications, one each developed in Enterprise COBOL with DB2, z/OS high level assembler with DB2, Enterprise PL/I with DB2 and Enterprise COBOL with DB2 and CICS. We also configure DB2 support for binding DBRMs directly to application plans (plan binds) as well as to collections (package binds). Most customers prefer to keep plan binds and package binds separate in their z/OS DB2 environments.

If you encounter any errors during the following steps, use the JES, assembler, and linkage editor messages to correct the problem.

3.2.1 Adding DB2-specific types

Create additional types as required. Here are some suggested types and naming conventions:

DBRM This type contains the source member input to a DB2 bind. It is generated by the DB2 preprocessing step.

PLNBIND This type contains editable REXX/CLIST source members for plan binds. They are used during SCLM Build to control DB2 bind and free functions.

PLNOUT This type contains non-editable build output that is used during SCLM Promote to control BIND and FREE functions for plans. During a build of a PLNBIND, a copy of the PLNBIND is copied to the type PLNOUT into the group that is being built. During a promote, this member is called to bind the plan in the TO group and free the plan in the FROM group. Since the PLNOUT is the “output” from the PLNBIND, SCLM will move both objects together during a Promote.

PKGBIND This type contains editable REXX/CLIST source members for package binds. They are used during SCLM Build to control DB2 bind and free functions.

PKGOUT This type contains non-editable build output that is used during SCLM Promote to control BIND and FREE functions for packages. During a build of a PKGBIND, a copy of the PKGBIND is copied to the type PKGOUT into the group that is being built. During a promote, this member is called to bind the package in the TO group and free the package in the FROM group. Since the PKGOUT is the “output” from the DB2CLIST, SCLM will move both objects together during a Promote.

ARCHLEC This type contains link-edit control (LEC) architecture definitions for source modules.

ARCHBIND This type contains generic architecture definitions to bind DBRMs.

DCLGEN This type contains DCLGEN members of DB2 tables and views.

Specify the additional types with the FLMTYPE macro in the SCLM01 project definition.

Note: While defining new DB2 types, you must preserve the EBCDIC collating sequence. SCLM promotes in EBCDIC collating sequence order. For example, types DBRM, DB2OUT, PLNOUT, PKGOUT are OK, but DBRM with CDB2OUT (for CICS programs) is not. If you use type CDB2OUT, during the promote, SCLM would initiate the bind for the CDB2OUT member before promoting the DBRM.

To have DB2CLIST (PKGBIND/PLNBIND) members and DBRM members with the same name, an FLMINCLS macro needs to be specified in the language definition for the DB2CLIST members. The FLMINCLS macro must list the DBRM type first on the TYPES parameter. Example 3-1 is an example of an FLMINCLS macro to do this.

Example 3-1 FLMINCLS for DB2CLIST language

```
* SPECIFY TYPES TO SEARCH FOR DBRMS THAT ARE TRACKED AS
* INCLUDES TO THE DB2 CLIST MEMBERS
*
                    FLMINCLS TYPES=(DBRM)
```

3.2.2 Allocating project partitioned data sets for DB2 support

Allocate the datasets for all the new types for all groups in your hierarchy. The data set characteristics for the new types are described in Table 3-1.

Table 3-1 Recommended data set attributes for SCLM types for DB2 support

Type	DSORG	RECFM	LRECL	BLKSIZE
DBRM	PO	FB	80	0
PLNBIND	PO	FB	80	0
PLNOUT	PO	FB	80	0
PKGBIND	PO	FB	80	0
PKGOUT	PO	FB	80	0
ARCHLEC	PO	FB	80	0
ARCHBIND	PO	FB	80	0
DCLGEN	PO	FB	80	0

3.2.3 Additional language definitions for DB2 support

Copy the following members from the ISPF systems macro library, ISP.SISPMACS, into the SCLM01.PROJDEFS.SOURCE:

FLM@BDO SCLM Language definition for DB2 bind/free CLIST output
FLM@BD2 SCLM Language definition for DB2 bind/free CLIST
FLM@2ASM SCLM language definition for z/OS High Level Assembler with DB2
FLM@2CBE SCLM language definition for Enterprise COBOL with DB2
FLM@2PLE SCLM language definition for Enterprise PL/I with DB2

Copy either one of the following members listed in Appendix C, “Chapter 3 listings” on page 725 into SCLM01.PROJDEFS.SOURCE:

COBCICD2 3-Step SCLM language definition for Enterprise COBOL with DB2 and CICS (Three-Step Version)

COBCICD2 1-Step SCLM language definition for Enterprise COBOL with DB2 and CICS (One-Step Version)

Change SCLM01.PROJDEFS.SOURCE members FLM@BDO, FLM@BD2, FLM@2ASM, FLM@2CBE, FLM@2PLE and COBCICD2, so that macro libraries, assembler/ precompiler/ compiler libraries, and assembler/ precompiler / compiler options match the libraries and products in use at your location. The changes are specified in the initial comments of each macro member.

3.2.4 Tailoring languages for package bind and plan bind

Do the following steps to tailor languages for package bind and plan bind:

1. Make two copies of the sample language translator FLM@BD2 in SCLM01.PROJDEFS.SOURCE and name them PKGBIND and PLNBIND.
2. Make two copies of the sample language translator FLM@BDO in SCLM01.PROJDEFS.SOURCE and name them PKGOUT and PLNOUT.
3. Modify the LANG values on the FLMLANGL macro for all four new members and the DFLTTYP and LANG values on the FLMALLOC macros in PKGBIND and PLNBIND as follows:
   a. Make the changes in bold for the FLMLANGL and FLMALLOC statements in PLNBIND and PKGBIND in Example 3-2 and Example 3-3.

Example 3-2 PLNBIND example for plan bind

```
   FLMLANGL    LANG=PLNBIND
  .
  .
          FLMALLOC IOTYPE=O,DDNAME=ISRDB2OT,KEYREF=OUT3,LANG=PLNOUT,  C
              RECNUM=750000,LRECL=80,RECFM=FB,DFLTTYP=PLNOUT
```

Example 3-3 PKGBIND example for package bind

```
   FLMLANGL    LANG=PKGBIND
  .
  .
          FLMALLOC IOTYPE=O,DDNAME=ISRDB2OT,KEYREF=OUT3,LANG=PKGOUT,  C
              RECNUM=750000,LRECL=80,RECFM=FB,DFLTTYP=PKGOUT
```

b. Change the LANG values on the PLNOUT and PKGOUT translators, so that LANG=PLNOUT and LANG=PKGOUT. See Example 3-4 and Example 3-5.

Example 3-4 FLMLANGL statement in PLNOUT

```
*
   FLMLANGL    LANG=PLNOUT
```

Example 3-5 FLMLANGL statement in PKGOUT

```
*
   FLMLANGL    LANG=PKGOUT
```

c. Modify the DBRMTYPE values in the OPTIONS parameter on the FLMTRNSL macros in all four DB2 translators to reflect your DBRM type. This tells SCLM which DBRM libraries to allocate during the bind step to the DBRMLIB DD. See Example 3-6.

Example 3-6 Defining DBRMTYPE in DB2 translators

```
*
* BUILD TRANSLATOR(S)
*
          FLMTRNSL   CALLNAM='DB2 BIND',
               FUNCTN=BUILD,
               COMPILE=FLMCSPDB,
               PORDER=1,
               GOODRC=4,
               CALLMETH=LINK,
               OPTIONS=(FUNCTN=BUILD,
               OPTION=BIND,
               SCLMINFO=@@FLMINF,
               PROJECT=@@FLMPRJ,
               ALTPROJ=@@FLMALT,
               DBRMTYPE=DBRM,
               GROUP=@@FLMGRP,
               MEMBER=@@FLMMBR)
```

Archived

3.2.5 Tailoring project definition for DB2 support

Include the above language definitions into the SCLM01 project definition by adding copy statements for them. Assemble and link edit the SCLM01 project definition.

3.2.6 Creating an LEC architecture definition for program with SQL

The enhanced SCLM01 project has assembler, COBOL, and PL/I programs with SQL and CICS statements. To control the building of these programs, you use Linkedit Control (LEC) architecture definitions. A LEC architecture definition defines all the source modules in the load module. It also includes the DB2 Language module. Save all LEC architecture definitions in the ARCHLEC type.

Example 3-7 and Example 3-8 show sample LEC architecture definitions for an MVS batch and an MVS CICS programs with DB2, respectively. Pay particular attention to the CMD statements in both of them.

Example 3-7 Sample LEC architecture definition – DB2 Batch

```
*
  INCLD SCLMDB2P SOURCE
  CMD   INCLUDE SYSLIB(DSNELI)
*
  PARM RMODE(24),AMODE(31)
*
  LOAD  SCLMDB2P LOAD
  LMAP  SCLMDB2P LMAP
```

Example 3-8 Sample LEC architecture definition – DB2 CICS

```
*
 INCLD RDBKC01  SOURCE
 CMD   INCLUDE  SYSLIB(DSNCLI)
 CMD   INCLUDE  SYSLIB(DFHEI1)
*
 LOAD  RDBKC01  CICSLOAD
 LMAP  RDBKC01  LMAP
```

3.2.7 Creating an HL architecture definition to control link-edit and bind

Create a High level (HL) architecture definition to control both the compilation/link-edit and the bind, in order to let SCLM know the correct processing scope. Use the INCLD keyword in the HL architecture definition for the xxxBIND member plus the INCL keyword for the LEC architecture definition. This way the default processing of the translator will run creating the xxxOUT member. Example 3-9 shows a sample HL architecture definition as follows:

Example 3-9 Sample HL architecture definition for compilation, link-edit and bind

```
INCL  RDBKC01  ARCHLEC
INCLD RDBKC01  PKGBIND
```

The INCLD keyword copies the xxxBIND contents to the xxxOUT member. This is so that the xxxOUT member can be used in the bind process for levels in the hierarchy higher than the development group. We recommend that you create such an HL architecture definition, because you need to maintain only one HL and one LEC architecture definition per source component.

Archived

3.2.8 Creating generic and HL architecture definitions

Alternatively, you can use a generic architecture definition for the bind, using the SINC and OUTx keywords, then linking this generic architecture definition to the LEC architecture definition by means of a HL architecture definition. Use the ARCHBIND type for the generic architecture definition and the ARCHLEC type for the LEC architecture definition. See Example 3-10 below for a sample generic architecture definition. See Example 3-8 and Example 3-9 above for samples of LEC architecture definitions.

Example 3-10 Sample Generic architecture definition for ARCHBIND Member

```
SINC  RDBKC01  PKGBIND
OUT3  RDBKC01  PKGOUT
```

Next, create an HL architecture definition to control both the compilation/link-edit and the bind, so that SCLM knows the correct processing scope. This HL architecture definition should just include the two previously defined LEC and generic architecture definitions for the source module. See Example 3-11.

Example 3-11 Sample HL architecture definition for Compilation, Link-Edit and Bind

```
INCL  RDBKC01  ARCHLEC
INCL  RDBKC01  ARCHBIND
```

3.2.9 Creating a bind control member (DB2CLIST)

Create a PLNBIND or PKGBIND member, generically called DB2CLIST, for each DB2 application plan or package, as applicable. The DB2CLIST is a TSO REXX or CLIST procedure that allows you to BIND or FREE the plan or package. This exec should contain code to perform the following functions:

- Allow different DB2 Subsystem names to be assigned to each group.
- Allow different DB2 Owner and Qualifier and other parameters to be assigned to each group.
- BIND the application plan or package.
- Optionally FREE the application plan or package.

The parameters and logic required are shown in Example 3-12.

The DB2CLIST member allows you to specify which DBRMs are bound into the application plan or package by use of the %INCLUDE statement. The INCLUDE statement consists of a %INCLUDE keyword followed by the name of the included DBRM. SCLM parses the DB2CLIST member and keeps a list of included DBRM names, as well as other accounting information in the account record for the DB2CLIST member. The INCLUDE directive and DBRM name must be on the same line. The format of the INCLUDE statement is:

```
/*  %INCLUDE dbrm-name   */
```

You can look at the names of included DBRMs for a DB2CLIST by browsing its accounting information:

1. Select the Utilities option from the SCLM Main Menu. (Option 3)
2. Select the Library option from the SCLM Utilities Menu. (Option 1)
3. From the SCLM Library Utility - Entry Panel, enter the DB2 type to be used during Build, in our examples this is PKGBIND.

Archived

4. From the list of members, select the PKGBIND member that you want to examine and browse its accounting information.
5. From the Accounting Record for the PKGBIND, select the Number of Includes.
6. Finally, you see the list of included DBRMs in the PKGBIND.

Use an architecture definition to build or to promote the DB2CLIST member. Use the SINC or INCLD keyword to reference the member from an architecture definition as shown previously. You can also build or promote The DB2CLIST member directly without using an architecture member. When the DB2CLIST member is built or promoted directly or is processed through an INCLD architecture definition keyword, SCLM uses the defaults defined in the DB2CLIST language definition. See Example 3-12.

Example 3-12 PKGBIND Example – Calls Common Bind EXEC

```
/* REXX */
Arg INPARMS
Parse var INPARMS '(' OPTION ' ' '(' GROUP ' '
/*********************************************************************/
/* SPECIFY AN INCLUDE FOR THE DBRM TO BE INCLUDED IN THE DB2         */
/* PACKAGE. SCLM TRACKS DEPENDENCY ON DBRMS WITH THE COMMENTED       */
/* OUT INCLUDE STATEMENT.                                            */
/*********************************************************************/
/*** THE INCLUDE STATEMENT MUST REMAIN COMMENTED OUT.              ***/
/*********************************************************************/
/*                                                                   */
/* %INCLUDE RDBKC01                                                  */
/*                                                                   */
/*********************************************************************/
/* SET UP THE PARMS FOR A PACKAGE BIND.                              */
/*********************************************************************/
If OPTION = "BIND" Then OPTION = "PACKBIND"
If OPTION = "FREE" Then OPTION = "PACKFREE"
MEMBER  = "RDBKC01"
EXPLAIN = "NO"
/*--------------------------------------*/
/* CALL REXXBIND EXEC TO PERFORM BIND    */
/*--------------------------------------*/
PARMS = OPTION GROUP MEMBER EXPLAIN
Address TSO "EX 'SCLM01.PROJDEFS.REXX(REXXBIND)' '"PARMS"'"
EXITCC = RC
EXIT EXITCC
```

In Example 3-12 we can see the following key points:

- The PKGBIND translator will pass in an Option and a Group. The OPTION will be BIND if the translator is executed at Build time or at promote time during the COPY phase. The OPTION will be FREE if the translator is in the PURGE phase of the promote.
- The DBRM name is specified in the commented out %INCLUDE. This will maintain SCLM dependency between the DBRM member and the PKGBIND that needs to be run to bind the DBRM.
- Certain parameters are set up for binding such as EXPLAIN. Other parameters could also be set up, such as ISOLATION and VALIDATE for example. If there are bind options that are specific to a particular member, then they can be set here.

Archived

- The bind processor is called to do the actual bind and set up other parameters based on the Group where the bind needs to occur. The actual bind could be coded in the PKGBIND but it is better practice to keep your bind processor separate from the PKGBIND members. This means that if any bind options change in future, the PKGBIND might not need to be changed, only rebuilt, thus minimizing the changes required.
- It is possible to include a “Select” on the group and set group-specific parameters there rather than in the bind processor. It is also possible to include the bind invocation in the PKGBIND but we do not recommend it, because, if a certain bind parameter changes, then all the PKGBIND members need to change.

See Appendix C-1, “Common bind exec example” on page 725 for the listing of a common bind EXEC example. You have to modify this according your DB2 needs.

3.2.10 Binding on different LPARs

The SCLM bind EXEC or processor in the above examples only works for binds on the machine where SCLM exists. For binds on other LPARs, the bind processor must use file tailoring to create a job to perform the bind.

Assume that your development and test environments reside on the same LPAR that SCLM is running on, but your PROD database and executables are on a different LPAR. In this case your bind processor would use the DB2CLIST method of binding for the DEV builds and promotes to TEST. However, when you promote to PROD, SCLM can only promote DB2CLIST and DB2OUT members to PROD but cannot do the bind on the prod LPAR by itself.

To bind on the prod LPAR, once SCLM finishes the promote work, it needs to invoke the promote copy or the promote purge user exit. The exit should be coded such that a list of the DBRMs being promoted is extracted from the PROMEXIT file. The processor then builds a job using ISPF file tailoring techniques that will do the bind for all the required members. The generated job is transferred and submitted to the production LPAR.

If the JES queue is shared between the LPARs, then the job can be submitted from the bind exec with the appropriate routing parameter to submit the bind on the other LPAR. If the JES queue is not shared, then a more complicated delivery of the bind JCL will be needed, possibly using FTP.

If you want to implement binds on other LPARs through a submitted job, it is best to create one in the Build user exit or the Promote user exit or both. This way, only one bind job would need be created for a package build or promote. For more information, see 12.11, “Additional SCLM user exit example” on page 277.

3.2.11 Summary

To summarize, here is a checklist of actions you need to perform to enable DB2 support for your SCLM project:

1. Add DB2-specific types, DBRM, PLNBIND, PLNOUT, PKGBIND, PKGOUT, ARCHLEC, ARCHBIND and DCLGEN, to your project definition, and allocate FB 80 PDSs/PDSEs for each of them.
2. Specify an FLMINCLS macro in the language definition for the PLNBIND and PKGBIND members. List the DBRM type first on the TYPES parameter of the FLMINCLS macro.

3. Copy sample language definitions that apply to your DB2 environment. Refer to the list of language definitions provided in Appendix C, “Chapter 3 listings” on page 725. Customize FLM@BD2 and FLM@BDO language definitions for PLNBIND/PLNOUT and PKGBIND/PKGOUT types and languages.
4. Create a LEC architecture definition under the ARCHLEC type for each source Program with embedded SQL with INCLD, LOAD and other statements, as applicable.
5. Create an HL architecture definition under the (existing) architecture definition type to associate the link-edit to the bind with the INCLD statement for the PLNBIND/PKGBIND member and the INCL statement for the LEC architecture definition.
6. Alternatively, create a pair of generic and HL architecture definitions. Code SINC and OUTx statements on the generic architecture definition under the ARCHBIND type, and the INCL statement on the HL architecture definition under the architecture definition type.
7. Create a PLNCLIST member for each application plan, and a PKGCLIST member for each application package, as a TSO REXX or a CLIST procedure.
8. Code a commented out %INCLUDE statement in the DB2CLIST member to have SCLM generate accounting information for the DBRM to be bound.
9. If you want to implement binds on other LPARs, create a Build user exit or a Promote user exit or both to build a job using ISPF file tailoring techniques.

3.3 FLMCSPDB DB2 bind/free translator

This translator is supplied with SCLM and is used to process the DB2CLIST and DB2OUT members. It processes the DB2CLIST at build time and DB2OUT at promote time. When processing the DB2CLIST it runs as a BUILD translator. When processing the DB2OUT, it runs as a COPY translator or a PURGE translator or both. When running at build time, it also copies the DB2CLIST to the DB2OUT for later processing.

3.3.1 FLMCSPDB invocation parameters

The following keyword parameters are expected as input for the FLMCSPDB translator:

ALTPROJ The variable name for the alternate project name. This parameter is required, and must be set to @@FLMALT.

DBRMTYPE The type name where the DBRMs are stored. This parameter is required, and must be set to a valid type for DBRMs defined in the project definition. Note: The FLMDBALC CLIST is run by the FLMCSPDB translator to allocate the DBRM type to the DBRMLIB ddname.

FUNCTN The SCLM function invoking the translator: BUILD, COPY, or PURGE (not to be confused with the FUNCTN= parameter passed to SCLM using the FLMTRNSL macro). This parameter is required.

GROUP The variable name for the group to build into or to promote from. This parameter is required, and must be set to @@FLMGRP.

MEMBER The variable name for the DBRM member name. This parameter is required, and must be set to @@FLMMBR. Also, OPTOVER=Y (default) must be set on the FLMCNTRL macro in the project definition.

OPTION The operation to be performed with the DB2 Plan or package: BIND or FREE. This parameter is required.

Archived

PROJECT The variable name for the project name. This parameter is required, and must be set to @@FLMPRJ.

SCLMINFO The variable name for the SCLM internal pointer. This parameter is required, and must be set to @@FLMINF.

TOGROUP The variable name for the group to promote to. This parameter is required, and must be set to @@FLMTOG. This variable is ignored for FUNCTN=BUILD.

3.4 Sample language definitions for DB2 support

We now examine DB2-specific language definitions more closely. As indicated earlier, a couple of them have been supplied with SCLM, and reside in your SISPMACS library. We also include here two example language definitions for reference purposes. We show the statements needed to define the control structures and SCLM macros. We also show how to gather the information you need and how to specify that information to SCLM in the form of language definition macros.

We discuss the following language definitions in this section:

- Language definitions supplied with SCLM:

  FLM@BD2 DB2CLIST (DB2 bind/free CLIST) language definition (Appendix C-2, “SCLM07.PROJDEFS.SOURCE(PKGBIND)” on page 728)

  FLM@BDO DB2OUT (DB2 bind/free CLIST output) language definition (Appendix C-3, “SCLM07.PROJDEFS.SOURCE(PKGOUT)” on page 730)

- Example language definitions:

  COBCICD2 3-Step COBCICD2 (Enterprise COBOL with DB2 and CICS) multi-step language definition (Appendix C-4, “COBCICD2 - multi-step version” on page 732)

  COBCICD2 1-Step COBCICD2 (Enterprise COBOL with DB2 and CICS) single-step language definition (Appendix C-5, “COBCICD2 - single-step version” on page 736)

We now illustrate the process of defining DB2CLIST and DB2OUT languages to SCLM. We discuss SCLM macros along with the information SCLM needs to control translator modules. Include language definitions in a project definition by means of the COPY statement.

3.4.1 DB2 bind/free language translator

See Appendix C-2, “SCLM07.PROJDEFS.SOURCE(PKGBIND)” on page 728 for the listing of the sample DB2CLIST language definition that we describe below.

Step 1: Define the language

The first step is to tell SCLM that this is a new language definition. The following FLMLANGL macro does this, as shown in Example 3-13.

Example 3-13 FLMLANGL for the DB2CLIST language

```
*
         FLMLANGL    LANG=PKGBIND,                                        C
               LANGDESC='DB2 BIND/FREE PACKAGE TRANSLATOR'
*
```

In this example, values are specified for two parameters. Defaults are used for the other parameters.

LANG Specifies the language name a user must enter on the SPROF panel or on the Migrate Utility panel to request that this language definition be used to drive build and parse operations of the DB2CLIST modules. In our example we have changed the generic DB2CLIST name to the more meaningful PKGBIND name.

LANGDESC Description of Language, max 40 characters.

Step 2: Define include sets to identify the location of included members

After the language is defined, you can specify where SCLM finds included members for the DB2CLIST language. In Example 3-14, the FLMINCLS macro identifies the types that are searched for includes.

Example 3-14 FLMINCLS for the DB2CLIST language

```
*
         FLMINCLS    TYPES=(DBRM)
*
```

In this example, the TYPES parameter of the FLMINCLS macro is used to tell SCLM where to look for DBRMs included in the DB2CLIST member. Because no name is specified, this definition applies to the default include set.

name **FLMINCLS** Specifies the name of the include set that uses this definition. If no name is specified (as in this example), the definition is associated with the default include set. An include set defines a search path for all includes associated with that include set. Refer to Chapter 2.2, “Language definitions based upon samples ” on page 40 for more information on include sets.

TYPES Specifies the name(s) of the types which are searched to find includes. In this case, SCLM searches the DBRM type first, and then the default @@FLMTYP type.

Step 3: Specify the programs that process DB2CLIST members

Next, identify the translators that are used to parse and build the DB2CLIST members. There are two such programs: a parser and a compiler or a processor for this language definition. For each of these programs, code an FLMTRNSL macro and the appropriate FLMALLOC macros and FLMCPYLB macros.

The DB2CLIST parser is FLMLSS, and it resides in the ISPF load library ISP.SISPLPA. This parser requires seven option parameters, PTABLEDD=<null>, SOURCEDD=SOURCE, TBLNAME=FLMPDBRM, STATINFO=@@STP, LISTINFO=@@LIS, LISTSIZE=@@SIZ, CONTIN=0, and EOLCOL=72, and reads the source from ddname SOURCE.

Add this to your language definition as shown in Example 3-15.

Archive

Example 3-15 Parse step for the DB2CLIST language

```
*
          FLMTRNSL CALLNAM='PARSE DB2 BIND',                           C
               FUNCTN=PARSE,                                           C
               COMPILE=FLMLSS,                                         C
               PORDER=1,                                               C
               OPTIONS=(PTABLEDD=,                                     C
               SOURCEDD=SOURCE,                                        C
               TBLNAME=FLMPDBRM,                                       C
               STATINFO=@@FLMSTP,                                      C
               LISTINFO=@@FLMLIS,                                      C
               LISTSIZE=@@FLMSIZ,                                      C
               CONTIN=0,                                               C
               EOLCOL=72)
*
```

The following parameters are included in this example:

CALLNAM A character string that appears in messages during the specified FUNCTN (in this case PARSE). This value will assist in recognizing which translator was executing during the specified FUNCTN.

FUNCTN The value PARSE tells SCLM that this program is to be invoked whenever you parse a module with language DB2CLIST.

COMPILE Member name of the load module for the DB2CLIST parser translator.

DSNAME Not used here, because ISP.SISPLPA is defined in the LINKLIST concatenation.

PORDER The value 1 tells SCLM that this program expects an options string but not a ddname substitution list.

OPTIONS Specifies the options string to be passed to the parser. Strings that start with @@FLM are SCLM variables, and they are replaced by their current values before the string is passed to the parser.

PTABLEDD The ddname assigned to the parser data set load module. This parameter is required.

TBLNAME The name of the parser table load module, FLMPDBRM, DBRM table. This parameter is required.

CONTIN The value 0 indicates to the parser not to concatenate continued lines. The default is column 72.

EOLCOL The value 72 tells the parser the maximum number of characters from each input line to process. The default is 0.

Since the parser reads its source from a ddname, you must tell SCLM how to allocate that ddname. To do this, use an FLMALLOC macro and an FLMCPYLB macro as shown in Example 3-16.

Example 3-16 Parse step continued for the DB2CLIST language

```
*  ------- (* SOURCE *)
           FLMALLOC IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
```

A description of the parameters follows:

IOTYPE=A Tells SCLM to allocate a ddname to one or more specific data set(s). Each of those data sets are subsequently identified by using an FLMCPYLB macro.

DDNAME Identifies the ddname to be allocated. In this case SOURCE.

@@FLMDSN(@@FLMMBR) Identifies the member to be parsed. When the two SCLM variables are resolved, you get the member of the data set in which you are interested.

Now you can tell SCLM how to invoke the FLMCSPDB DB2 bind/free translator. To do so, use an FLMTRNSL macro followed by one or more FLMALLOC and FLMCPYLB macros. FLMCSPDB resides in the ISPF load library ISP.SISPLPA as shown in Example 3-17.

Example 3-17 FLMCSPDB step of the DB2CLIST language

```
*
          FLMTRNSL CALLNAM='DB2 BIND',                          C
               FUNCTN=BUILD,                                    C
               COMPILE=FLMCSPDB,                                C
               PORDER=1,                                        C
               GOODRC=4,                                        C
               CALLMETH=LINK,                                   C
               OPTIONS=(FUNCTN=BUILD,                           C
               OPTION=BIND,                                     C
               SCLMINFO=@@FLMINF,                               C
               PROJECT=@@FLMPRJ,                                C
               ALTPROJ=@@FLMALT,                                C
               DBRMTYPE=DBRM,                                   C
               GROUP=@@FLMGRP,                                  C
               MEMBER=@@FLMMBR)
```

Specify as many parameters as required, and let SCLM supply default values for the others:

CALLNAM Names the DB2 bind/free translator. This name appears in build messages.

FUNCTN Tells SCLM that this program gets invoked whenever you want to build a member with language DB2CLIST.

COMPILE Identifies the load module name FLMCSPDB for the DB2 bind/free translator.

DSNAME Not used here, because ISP.SISPLPA is defined in the LINKLIST concatenation.

PORDER The value 1 tells SCLM that this program expects an options string but not a ddname substitution list.

GOODRC The value 4 indicates that SCLM is to consider this build unsuccessful if the translator completes with a return code greater than 4, meaning Only informational and warning diagnostics are acceptable.

OPTIONS Specifies the options string to be passed to the translator. At translator run time, the SCLM variable @@FLMMBR is resolved to the member name being built.

DBRMTYPE The value DBRM is the type where DBRMs are stored.

FUNCTN The value BUILD specifies the SCLM function invoking the translator.

OPTION The value BIND specifies the operation to be performed with the DB2 Plan or package.

Archived

The FLMCSPDB translator uses four ddnames for the build function, and does not mandate a ddname substitution list, but does support one.

Let us examine the ddnames used by the FLMCSPDB translator in detail:.

Allocate ddname ISRPROXY to the DB2CLIST member to be processed as shown in Example 3-18.

Example 3-18 ISRPROXY ddname for the DB2CLIST language

```
*  1       -- ISRPROXY --
           FLMALLOC IOTYPE=S,DDNAME=ISRPROXY,KEYREF=SINC,              C
               CATLG=Y,RECNUM=750000,LRECL=80,RECFM=FB
```

The parameters used in the example are as follows:

IOTYPE The value S tells SCLM to allocate a temporary sequential data set.

DDNAME This identifies the ddname to be allocated. In this case ISRPROXY.

KEYREF The value SINC tells SCLM that, if you are building a DB2CLIST member directly, SCLM copies that member to this temporary data set. If you are building a generic architecture definition, SCLM copies the members listed on the SINC statement to this data set.

CATLG The value Y tells SCLM to cataloged this temporary data set with the SCLM02 high-level qualifier, and to delete it upon completion of all translator functions.

RECFM The value FB specifies the record format of the temporary data set that SCLM creates. In this example, the record format is fixed block.

LRECL The value 80 specifies the record length, in characters, of the temporary data set that SCLM creates.

RECNUM The value 5000 tells SCLM to allocate enough space in this data set to hold 5000 records (records that are fixed block and 80 characters in length).

Allocate ddname ISRDB2OT to the DB2OUT copy of the DB2CLIST member as shown in Example 3-19.

Example 3-19 ISRDB2OT ddname for the DB2CLIST language

```
*  2       -- ISRDB2OT --
           FLMALLOC IOTYPE=O,DDNAME=ISRDB2OT,KEYREF=OUT3,LANG=PKGOUT,  C
               RECNUM=750000,LRECL=80,RECFM=FB,DFLTTYP=PKGOUT
```

The following parameters are specified in this macro:

IOTYPE The value O tells SCLM to allocate a temporary sequential data set to ddname DB2OUT.

DDNAME This identifies the ddname to be allocated. In this case ISRDB2OT.

KEYREF The value OUT3 tells SCLM to save what is written to this ddname and keep it under SCLM control, SCLM must be able to determine the member name and the SCLM-controlled data set name in which it is to save this output member. If SCLM is building an architecture definition, it determines the project, group, type, and member as follows:

1. The high-level qualifier is the project identifier that was previously specified.

2. The group is the level at which the build is taking place. The group name is the second qualifier.

3. SCLM looks at the architecture definition being built and retrieves the member and type from the architecture statement associated with the keyword OUT3. The type name is the third qualifier.

LANG The value DB2OUT allows the build output of a DB2CLIST member to be assigned a different language, DB2OUT, than the build input, DB2CLIST. If this parameter is not specified, then build outputs are assigned the same language as inputs. See the next section for a detail discussion of the DB2OUT language definition. In our example we have changed the default DB2OUT language to the more meaningful PKGOUT.

RECFM The value FB specifies the record format of the temporary data set that SCLM creates. In this example, the record format is fixed block.

LRECL The value 80 specifies the record length, in characters, of the temporary data set that SCLM creates.

RECNUM The value 750000 tells SCLM to allocate enough space in this dataset to hold 750000 records (records that are fixed block and 80 characters in length).

DFLTTYP The value DB2OUT tells SCLM to save what is written to this ddname and keep it under SCLM control, SCLM must be able to determine the member name and the SCLM-controlled data set name in which it is to save this output member. If SCLM is building a source (DB2CLIST) member, it determines the project, group, type and member as follows:

1. The high-level qualifier is the project identifier that was previously specified.

2. The group is the level at which the build is taking place.

3. The type is the value of the DFLTTYP keyword.

4. The member name defaults to the name of the member being built.

If SCLM is building an architecture definition (and not a source member directly) then the DFLTTYP value is ignored. Instead, SCLM uses the type associated with the KEYREF value.

In our example we have changed the default DB2OUT default type to the more meaningful PKGOUT.

Allocate ddname DSNTRACE to a temporary work data set for the translator to invoke DB2 trace facility using IOTYPE=W and PRINT=I as shown in Example 3-20.

Example 3-20 DSNTRACE ddname for the DB2CLIST language

```
*  3       -- DSNTRACE --
           FLMALLOC IOTYPE=W,DDNAME=DSNTRACE,PRINT=I,                  C
               RECFM=F,LRECL=132,BLKSIZE=132
```

Allocate ddname DUMMYDD to one or more partitioned data sets. FLMCSPDB uses this ddname to find included DBRMs. The FLMINCLS macro described earlier needs to be referenced here to ensure that the compiler is picking up includes from the correct data sets. Since IOTYPE=I allocations default to the default include set shown earlier, this is automatically done. If another name was used on the FLMINCLS macro, that name needs to be referenced here using the INCLS parameter.

IOTYPE=I allocates a ddname with a concatenation of all the PDSs in the hierarchy starting with the group specified for the BUILD and ending with the top, or production level, group. First the hierarchy for the INCLUDE type is allocated, followed by the type of the first SINCed member from the architecture definition, or, if no architecture definition is used, the type of the member being built. This is shown in Example 3-21.

Example 3-21 DUMMYDD ddname for the DB2CLIST language

```
*  4       -- DUMMY DD - REQUIRED FOR FLMINCLS MACRO BUT NOT TRANSLATOR
           FLMALLOC  IOTYPE=I,DDNAME=DUMMYDD,KEYREF=SINC
```

The parameters used with this macro are as follows:

IOTYPE The value I tells SCLM to allocate this ddname to a concatenation of SCLM-controlled data sets. The types used in the concatenation are determined by the FLMINCLS macro referenced by the INCLS parameter on the FLMALLOC macro. In this case, there is no INCLS parameter so the default FLMINCLS (or include set) is used.

A hierarchy of data sets is concatenated for each type specified for the referenced FLMINCLS macro. The hierarchy begins at the group where the build is taking place and extends to the top of the project's hierarchy.

In this case, the concatenation first contains all of the data sets for the INCLUDES type followed by the data sets for the value substituted into the @@FLMTYP variable. See the KEYREF parameter to determine the value which is substituted into the @@FLMTYP and @@FLMETP variables.

KEYREF The value SINC tells SCLM that, if you are building an architecture definition, refer to the first SINC statement in that architecture definition for the type that is substituted into the @@FLMTYP macro. The value for @@FLMETP comes from the EXTEND= parameter of the FLMTYPE macro for that type. If you are not building an architecture definition, the type is the type of the member being built.

These macros together constitute a language definition similar to one in Figure 3-15, SCLM02.PROJDEFS.SOURCE(FLM@BD2). It contains comments to explain the flow of operations.

3.4.2 DB2 bind/free output language translator

See Appendix C-3, “SCLM07.PROJDEFS.SOURCE(PKGOUT)” on page 730 for the listing of the sample DB2OUT language definition described below.

Step 1: Define the language

The first step is to tell SCLM that this is a new language definition. The following FLMLANGL macro does this as shown in Example 3-22.

Example 3-22 FLMLANGL statement for the DB2OUT language

```
*
         FLMLANGL    LANG=PKGOUT,                                        C
               LANGDESC='DB2 BIND/FREE PACKAGE TRANSLATOR'
*
```

In this example, values are specified for two parameters. Defaults are used for the other parameters:

LANG This specifies the language name of this language definition to drive promote copy and promote purge operations of DB2OUT members. In our example we have changed the generic DB2OUT name to the more meaningful PKGOUT name.

LANGDESC This is a description of the language, maximum 40 characters.

Step 2: Specify the program that promote copies DB2OUT members

Next, identify the translator program that is used to perform the promote copy operation for the DB2OUT members. The translator you need to invoke to do this is the same one which builds the DB2CLIST members, namely the FLMCSPDB DB2 bind/free translator, discussed in the previous section. For this program, code an FLMTRNSL macro and the appropriate FLMALLOC macros and FLMCPYLB macros as shown in Example 3-23. FLMCSPDB resides in the ISPF load library ISP.SISPLPA.

Example 3-23 FLMTRNSL step of the DB2OUT language

```
*
         FLMTRNSL CALLNAM='DB2 PROM BIND',                             C
               FUNCTN=COPY,                                            C
               COMPILE=FLMCSPDB,                                       C
               PORDER=1,                                               C
               GOODRC=4,                                               C
               CALLMETH=LINK,                                          C
               PDSDATA=Y,                                              C
               OPTIONS=(FUNCTN=@@FLMFNM,                               C
               OPTION=BIND,                                            C
               SCLMINFO=@@FLMINF,                                      C
               PROJECT=@@FLMPRJ,                                       C
               ALTPROJ=@@FLMALT,                                       C
               DBRMTYPE=DBRM,                                          C
               GROUP=@@FLMGRP,                                         C
               TOGROUP=@@FLMTOG,                                       C
               MEMBER=@@FLMMBR)
```

The following parameters are used in this example:

CALLNAM This names the DB2 bind/free translator and appears in build messages.

FUNCTN This tells SCLM that this program gets invoked whenever you want to promote copy a member with language DB2OUT.

COMPILE This identifies load module name FLMCSPDB for DB2 bind/free translator.

DSNAME This is not used here, because ISP.SISPLPA is defined in the LINKLIST concatenation.

PORDER The value 1 tells SCLM that this program expects an options string but not a ddname substitution list.

GOODRC The value 4 indicates that SCLM is to consider this build unsuccessful if the translator completes with a return code greater than 4, meaning Only informational and warning diagnostics are acceptable.

OPTIONS This specifies the options string to be passed to the translator. At translator run time, the SCLM variable @@FLMMBR is resolved to the member name being promoted.

Archived

DBRMTYPE The value DBRM is the type name where DBRMs are stored.

FUNCTN The value @@FLMFNM resolves to the SCLM function promote copy invoking the translator.

OPTION The value BIND specifies the operation to be performed with the DB2 Plan or package.

The FLMCSPDB translator uses two ddnames for the promote copy function, and does not mandate a ddname substitution list, but does support one.

Let us now examine the ddnames used by the FLMCSPDB translator in detail.

Allocate ddname ISRPROXY to the DB2OUT member to be processed as shown in Example 3-24.

Example 3-24 ISRPROXY ddname of the DB2OUT language

```
*  1       -- ISRPROXY --
           FLMALLOC  IOTYPE=A,DDNAME=ISRPROXY
            FLMCPYLB @@FLMDSF
```

The following parameters are used in this example:

IOTYPE=A This tells SCLM to allocate a ddname to one or more specific data set(s). Each of those data sets are subsequently identified by using an FLMCPYLB macro.

DDNAME This identifies the ddname to be allocated.

FLMCPYLB The value @@FLMDSF identifies the DB2OUT data set member SCLM located (found) at the promote group to be processed. When this SCLM variable is resolved, you get the member of the data set in which you are interested.

Allocate ddname DSNTRACE to a temporary work data set for the translator to invoke DB2 trace facility using IOTYPE=W and PRINT=I as shown in Example 3-25.

Example 3-25 DSNTRACE ddname of the DB2OUT language

```
*  2       -- DSNTRACE --
           FLMALLOC  IOTYPE=W,DDNAME=DSNTRACE,PRINT=I,                C
              RECFM=F,LRECL=132,BLKSIZE=132
```

Step 3: Specify the program that promote purges DB2OUT members

Next, identify the translator program that is used to perform the promote purge operation for the DB2OUT members. The translator you need to invoke to do this is the same one which performs promote copy for the DB2OUT members, namely FLMCSPDB, discussed earlier in this section. For this program, code an FLMTRNSL, as shown in Example 3-26, macro and the appropriate FLMALLOC macros and FLMCPYLB macros. FLMCSPDB resides in the ISPF load library ISP.SISPLPA.

Archived

Example 3-26 FLMTRNSL for the DB2OUT language

```
*
         FLMTRNSL CALLNAM='DB2 PROM FREE',                             C
               FUNCTN=PURGE,                                           C
               COMPILE=FLMCSPDB,                                       C
               PORDER=1,                                               C
               GOODRC=4,                                               C
               CALLMETH=LINK,                                          C
               PDSDATA=Y,                                              C
               OPTIONS=(FUNCTN=@@FLMFNM,                               C
               OPTION=FREE,                                            C
               SCLMINFO=@@FLMINF,                                      C
               PROJECT=@@FLMPRJ,                                       C
               ALTPROJ=@@FLMALT,                                       C
               DBRMTYPE=DBRM,                                          C
               GROUP=@@FLMGRP,                                         C
               TOGROUP=@@FLMTOG,                                       C
               MEMBER=@@FLMMBR)
```

The following parameters are used in this example:

FUNCTN The value of PURGE in the first occurrence tells SCLM that this program gets invoked whenever you want to promote purge a member with language DB2OUT.

FUNCTN The value @@FLMFNM in the second occurrence resolves to the SCLM function promote purge invoking the translator.

OPTION The value FREE is the operation to be performed with the DB2 Plan or package.

See Step 2 for a detailed description of other parameters used in this example.

The FLMCSPDB translator uses two ddnames for the promote purge function, and does not mandate a ddname substitution list, but does support one.

Let us now examine the ddnames used by the FLMCSPDB translator:.

Allocate ddname ISRPROXY to the DB2OUT member to be processed as shown in Example 3-27.

Example 3-27 ISRPROXY ddname for DB2OUT language

```
*  1       -- ISRPROXY --
           FLMALLOC  IOTYPE=A,DDNAME=ISRPROXY
            FLMCPYLB @@FLMDSF
```

See Step 2 for a detailed description of the parameters used in this example.

Allocate ddname DSNTRACE to a temporary work data set for the translator to invoke DB2 trace facility using IOTYPE=W and PRINT=I as shown in Example 3-28.

Example 3-28 DSNTRACE ddname for DB2OUT language

```
*  2       -- DSNTRACE --
           FLMALLOC  IOTYPE=W,DDNAME=DSNTRACE,PRINT=I,
               RECFM=F,LRECL=132,BLKSIZE=132
```

Archived

These macros together constitute a language definition similar to one in Figure 3-16, SCLM02.PROJDEFS.SOURCE(FLM@BDO). It contains comments to explain the flow of operations.

3.4.3 Enterprise COBOL with DB2 and CICS

SCLM offers sample language definition macros to support high-level languages, such as COBOL, PL/I. C/C++, JAVA, FORTRAN, PASCAL, etc., to work with DB2 batch and online interfaces. We refer to three of these macro members, FLM@2ASM, FLM@2CBE and FLM@2PLE earlier in this chapter. However, these macro members are very similar to their plain MVS batch counterparts, FLM@HLAS, FLM@COBE and FLM@PLIE, respectively. Hence, we do not cover them in detail in this book. Refer to 2.2, “Language definitions based upon samples ” on page 40, and initial comments of each macro member, should you need to customize any language definition for your requirements.

We do include listings of two COBOL with DB2 and CICS (COBCICD2) versions; a three-step and one-step version. Appendix C-4, “COBCICD2 - multi-step version” on page 732 and Appendix C-5, “COBCICD2 - single-step version” on page 736 have the listings for these two versions of the COBCICD2 language.

We include them for reference purposes, since they are not covered in the shipped samples. The multi-step version of COBCICD2 uses the traditional three step process consisting of the DB2 precompiler, CICS precompiler and Enterprise COBOL compiler. The single step version of COBCICD2 uses the built in CICS and SQL processing of Enterprise COBOL compiler.

Note: There are some differences in behavior between the stand-alone COBOL DB2 precompiler used in the three-step version and the integrated COBOL DB2 coprocessor used by the one step version.

3.4.4 Storing DB2 declarations in a different library from program includes

Something many customers have a requirement to do is to store DB2 table declare members separate from the COBOL or PL/I language includes. This means potentially having the same named member in two different libraries. With some recent improvements in SCLM in z/OS 1.8 that have been retrofitted back to z/OS 1.6 and z/OS 1.7, this has now been made possible. This support has been implemented into the two most common parsers:

FLMLPCBL COBOL parser

FLMLPGEN General purpose parser used for PL/I and Assembler

By specifying some additional parameters on the parser step in the language translators, it is now possible to utilize standard PL/I and COBOL techniques to allow separate include sets for normal includes and DB2 includes.

The previously mentioned samples provided in ISP.SISPMACs, as well as FLM@2CCE, do not utilize this support. So the samples provided in Example C-4 on page 732, Example C-5 on page 736, and Example C-6 on page 739 show this support, and we refer to these translators in the following explanations of the support.

COBOL support: Multi-step language definition

The sections of the multi-step Enterprise COBOL compiler with separate CICS translator and DB2 preprocessor that we have modified are described below. In order to implement this support, there are three things that need to be changed:

- Adding multiple FLMINCLS to support multiple include sets
- Adding new parser parameters to let SCLM to know to create different include sets
- Adding DB2 include set statements to DB2 preprocessor SYSLIB allocation

COBOL support: Single-step language definition

The sections of the single-step Enterprise COBOL compiler with integrated CICS translator and DB2 coprocessor that we have modified from the shipped sample are as described below. In order to implement this support, there are three things that need to be changed:

- Adding multiple FLMINCLS to support multiple include sets
- Adding new parser parameters to let SCLM to know to create different include sets
- Adding additional FLMALLOC statements for additional DB2 include sets

Specify different include sets for COBOL copybooks and DB2 includes

The FLMINCLS macro specifies the name of the include sets that define to SCLM where included copybooks and DB2 DCLGEN includes will be resolved from. Let us look at the definition as shown in Example 3-29.

Example 3-29 Include sets for copybook and DCLGEN support

```
*
         FLMINCLS TYPES=(SOURCE,COPYLIB,DCLGENC)
DCLGENC  FLMINCLS TYPES=(DCLGENC)
*
```

The first FLMINCLS definition is associated with the default include set because no name is specified in the left most position. This means that if there is a *COPY XYZ* statement in the COBOL source, SCLM will build a SYSLIB concatenation that searches the source followed by the copybook libraries and then followed by the DCLGENC library. The second FLMINCLS definition is associated with the DCLGENC include set and SCLM will use this alternate include set to search for DCLGENs as follows:

- When the source code is being processed by the multi-step COBOL language definition with a separate DB2 preprocessor step and there is a EXEC SQL INCLUDE statement in the COBOL source then the DCLGENC include set will be used to find the DCLGENs.
- When the source code is being processed by the single-step COBOL language definition with the integrated DB2 coprocessor and there is a *COPY XYZ OF DCLGENC* statement in the COBOL source, then the DCLGENC include set will be used to find the DCLGENs.

Specify parser parameters to handle DB2 includes

There are additional and relatively new parameters on the COBOL parser, FLMLPCBL, to provide support for multiple include sets as shown in Example 3-30.

Example 3-30 FLMLPCBL parser with new DB2 include set statements

```
*
          FLMTRNSL   CALLNAM='SCLM COBOL PARSE',                       C
               FUNCTN=PARSE,                                           C
               COMPILE=FLMLPCBL,                                       C
               PORDER=1,                                               C
               OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,SQL=DCLGENC,INCLSET)
```

We see the new parameters highlighted in Example 3-30. The definitions are as follows:

SQL= This parameter has a maximum of eight characters specifying the name of the include set assigned to EXEC SQL INCLUDE dependencies.

INCLSET When INCLSET is present, include set dependencies will be generated. That is, when COPY ABC OF XYZ statements are encountered, a dependency for copybook ABC is generated with an include set name of XYZ. This facilitates having copybooks with same name from different sources. This parameter is only needed in the single-step COBOL language definition.

What this means is that, when using the same named member for both COBOL copybooks and DB2 DCLGENs and the source is being processed by the single step COBOL language definition, the DB2 declare needs to be included with the COPY XYZ OF DCLGENC rather than specifying an EXEC SQL INCLUDE member. So the following COBOL code would be used to specify a copybook of RDBKTB01 and a DCLGEN or RDBKTB01 as shown in Example 3-31.

Example 3-31 COBOL code with include for COPYBOOK, DB2 table declare and SQLCA

```
*
 COPY RDBKTB01.
*
 COPY RDBKTB01 OF DCLGENC.
*
     EXEC SQL
         INCLUDE SQLCA
     END-EXEC.
*
```

However, when the source code is processed by the multi-step COBOL language that uses the DB2 preprocessor the EXEC SQL INCLUDE statements can be used to include DCLGENs.

Warning: If your COBOL code current uses EXEC SQL INCLUDE statements to include your DB2 DCLGENs and you cannot or choose not to recode them with the COPY XYZ OF DCLGENC syntax, and you have same named members for both COBOL copybooks and DB2 DCLGENs, then you must code your SCLM DB2 languages using the multi-step language definition that uses the DB2 preprocessor.

Regardless of which language variation is being used, when the member is saved, the include list stored in the account record will look like Example 3-32.

Example 3-32 Include list specified in the account record

```
Include   Include-set
--------  -----------
RDBKTB01  DCLGENC
RDBKTB01
DFHAID
ORDRSET
```

Specify additional FLMALLOC for new DB2 include set

The final required change for the single step language definition is the addition of an additional FLMALLOC statement to the COBOL compile translator step as shown in Example 3-33.

Example 3-33 FLMALLOC statement to specify DB2 include set

```
*
          FLMALLOC  IOTYPE=I,DDNAME=DCLGENC,KEYREF=SINC,INCLS=DCLGENC
```

This FLMALLOC can be specified at the end of the translator step for the COBOL compile. The IOTYPE=I specifies this as an include definition. You can see that the INCLS parameter is used. This is the reference to the FLMINCLS with the name of DCLGENC. SCLM will build a DD name of DCLGENC with a list of the types specified in the DCLGENC FLMINCLS statement. Normal COBOL COPY statements will be resolved through the normal SYSLIB concatenation.

Specify additional new DB2 include set

The final required change to the multi-step COBOL language definition is to specify the include step to the DB2 preprocessor SYSLIB allocation as shown in Example 3-34.

Example 3-34 SYSLIB statement to specify DB2 include set

```
*  4       -- SYSLIB --
           FLMALLOC  IOTYPE=I,KEYREF=SINC,INCLS=DCLGENC
*
```

The IOTYPE=I specifies this as an include definition. You can see that the INCLS parameter is used. This is the reference to the FLMINCLS with the name of DCLGENC. SCLM will build a SYSLIB concatenation with a list of the types specified in the DCLGENC FLMINCLS statement. Normal COBOL COPY statements will be resolved through the normal SYSLIB concatenation.

The complete language translators can be found in Example C-4 on page 732 and Example C-5 on page 736.

PL/I support

The sections of the single-step Enterprise PL/I compiler with DB2 coprocessor that we have modified from the shipped sample are as described below. In order to implement this support there are three things that need to be changed:

- Adding multiple FLMINCLS to support multiple include sets
- Adding new parser parameters to let SCLM to know to create different include sets
- Adding addition FLMALLOC statements for additional DB2 include set

Specify different include sets for PL/I includes and DB2 includes

The FLMINCLS macro specifies the name of the include sets that define to SCLM where included PLINCLs and DB2 DCLGEN includes will be resolved from. Let us look at the definition as shown in Example 3-35.

Example 3-35 Include sets for COPYBOOK and DCLGEN support

```
*
          FLMINCLS TYPES=(PLI,PLINCL,DCLGENP)
DB2DCLS   FLMINCLS TYPES=(DCLGENP)
*
```

The first FLMINCLS definition is associated with the default include set because no name is specified in the left most position. This means that if there is a *%INCLUDE XYZ* statement in the PL/I source, SCLM will build a SYSLIB concatenation that searches the PL/I source followed by the PLINCL libraries and then followed by the DCLGENP library. The second FLMINCLS definition is associated with the DCLGENP include set. This means that if there is a *%INCLUDE DB2DCLS(XYZ)* statement in the PL/I source, then SCLM builds an alternate include set for the PLINCLs that are in the DCLGENP include set.

Specify parser parameters to handle DB2 includes

There is an additional and relatively new parameter on the general parser, FLMLPGEN, used by PL/I, to provide support for multiple include sets as shown in Example 3-36.

Example 3-36 FLMLPGEN parser with new DB2 include set statement

```
*
          FLMTRNSL   CALLNAM='SCLM PL/I PARSE',                        C
               FUNCTN=PARSE,                                           C
               COMPILE=FLMLPGEN,                                       C
               PORDER=1,                                               C
               OPTIONS=(SOURCEDD=SOURCE,                               C
               STATINFO=@@FLMSTP,                                      C
               LISTINFO=@@FLMLIS,                                      C
               LISTSIZE=@@FLMSIZ,                                      C
               LANG=I(I))
```

We see the new parameters highlighted in Example 3-36. The following definition applies:

LANG=I(I) The LANG=I parameter is the current parameter that specifies that this parser is for the PL/I language. By appending the (I) parameter it generates include set dependencies.

What this means is when using the same named member for both PL/I INCLUDEs and DB2 DCLGENs that the DB2 declare needs to be included with the %INCLUDE DCLGENP(XYZ) rather than specifying an EXEC SQL INCLUDE member. So the following PL/I code would be used to specify a PLINCL of RDBKTB01 and a DCLGEN or RDBKTB01 as shown in Example 3-37.

Example 3-37 PL/I code with include for PLINCL, DB2 table declare and SQLCA

```
%INCLUDE RDBKTB01;

/*****************************************************/
/* SQL INCLUDE FOR SQLCA                             */
/*****************************************************/
EXEC SQL
   INCLUDE SQLCA;

/*****************************************************/
/* SQL DECLARATION FOR TABLE SCLMTB01                */
/*****************************************************/
%INCLUDE DB2DCLS(RDBKTB01);
```

When the member is saved, the include list stored in the account record will look like Example 3-38.

Example 3-38 Include list specified in the account record

```
Include   Include-set
--------  -----------
RDBKTB01  DB2DCLS
RDBKTB01
```

Specify additional FLMALLOC for new DB2 include set

The only change required to the PL/I compile translator step in the shipped sample, FLM@2PLE, is the addition of an additional include FLMALLOC statement as shown in Example 3-39.

Example 3-39 FLMALLOC statement to specify DB2 include set

```
*
            FLMALLOC  IOTYPE=I,DDNAME=DB2DCLS,KEYREF=SINC,INCLS=DB2DCLS
```

This FLMALLOC can be specified at the end of the translator step for the PL/I compile. The IOTYPE=I specifies this as an include definition. You can see that the INCLS parameter is used. This is the reference to the FLMINCLS with the name of DB2DCLS. SCLM will build a DD name of DB2DCLS with a list of the types specified in the DB2DCLS FLMINCLS statement. Normal PL/I %INCLUDE statements will be resolved through the normal SYSLIB concatenation.

The complete language translator can be found in Appendix C-6, “PLEDB2 - single-step version” on page 739.

Archived

Archived

4

Migrating to SCLM

After you have tested your new SCLM project, you are now ready to fully populate it with all of your source members, create architectural definitions (ARCHDEFs) for all members that need to be compiled and linked, and optionally compile and link (build) all source members and finally cut over to your new SCLM project. Before we present this migration process, let us first review the big picture of setting up and converting to SCLM.

Here are main steps:

1. Planning/kickoff meeting
2. Setting up SCLM:
   a. Defining project hierarchy
   b. Allocating library (data set) types
   c. Defining language definitions (processes)
   d. Setting up project controls
   e. Setting up any needed user exits.
3. Testing
4. Source migration
5. Creating architecture definitions and optionally building them
6. Cutover to SCLM

In the previous chapters we have discussed SCLM planning, setup, and testing. This chapter deals with the last three steps (4, 5, and 6) of converting to SCLM.

© Copyright IBM Corp. 2007. All rights reserved.

4.1 Principles of migration

The basic steps to migrating to SCLM are the same regardless of whether you are coming from a library management product (for example, Endevor, Librarian, Panvalet, or Changeman) or from datasets. The steps are as follows:

1. Create a migration plan. See section 4.2, “Creating a migration plan” for a further explanation of this step.
2. Create an “Alternate Project” for migration to move code directly into the production level in SCLM. See 4.3, “Creating alternate MIGRATE project definition” on page 122 for a further explanation of this step.
3. Establish a production freeze window or begin to track changes and plan for delta migration. See 4.4, “Production freeze versus delta migration” on page 123 for a further explanation of this step.
4. Migrate members into SCLM:
   a. Extract members from current library management product (if applicable). You must use the utilities of your current product to perform this extraction.
   b. Separate your members by language, language variation and type into a set of extract datasets. See section 4.5, “Separating your source code into extract datasets” for a explanation of this process.
   c. Remove/modify proprietary library code:

      Example: -INC and ++INCLUDE statements

      See 4.5.2, “Removing and/or modifying proprietary library code” on page 125 for a further explanation of this step.
   d. If a copy exit is not being used, modify all JCL and PROCs to point to SCLM load, proc, and control libraries. See 4.6, “Copy exit versus JCL changes” on page 125 for further explanation of this step.
   e. Perform the following two steps for each language extract dataset:
      i. Copy members into SCLM libraries:

         - Create a IEBCOPY job or use standard ISPF 3.3 copy.
         - Perform copy for only one language extract dataset at a time.

      ii. Run the SCLM Migration Utility, which registers the members with SCLM.

      - Assign a language and records dependencies.
      - Execute migrate for only one language extract dataset at a time.

      See section 4.7, “Running the SCLM Migration Utility” for an explanation of how to run the Migration Utility.
5. Create ARCHDEFs for load modules, copy them to the ARCHDEF type, and migrate them into SCLM with the ARCHDEF language. ARCHDEFs should also be created by language type. This topic is covered in section 4.8, “ARCHDEF generation”
6. If SCLM load modules are going to be used when SCLM goes live, use the high level member listing ARCHDEFs to build all members. If a copy exit is going to be used, then build a subset of the members to ensure that they build correctly.
7. Repeat step 4 for the delta migration if there is no production freeze. See 4.4, “Production freeze versus delta migration” on page 123 for further explanation of this step.
8. Cut over to SCLM. This topic is covered in section 4.9, “Cutover to SCLM”.

Archived

4.2 Creating a migration plan

It is important to plan for your migration to SCLM. After you have created your table of migrate datasets (see 4.5.1, “Determining source inventory and map to extract and SCLM datasets”), you need to use them to create a list of migration tasks in a ordered list such as shown in “Example SCLM migration plan”. Then you can break out the plan into as many subtasks as you feel are necessary for you to completely implement it.

Example SCLM migration plan

Here we list the main steps for your migration plan:

1. Copy copybooks to the SCLM production COPYLIB dataset and use the SCLM migration utility (SCLM option 3.3) to migrate them into SCLM with a language of COPY.
2. Copy DCLGENs to the SCLM production DCLGEN dataset and use the SCLM migration utility to migrate them into SCLM with a language of COPY.
3. Copy text types (JCLLIB, etc.) to their respective SCLM dataset and use the SCLM migration utility for each type to migrate them into SCLM with a language of TEXT.
4. Separate out the source code to be moved into SCLM by language type into separate extract datasets, one per language. Use the SCLM language as part of the dataset name. For example, project.MIGRATE.SOURCE.lang.
5. For each language extract dataset, copy the members to the SCLM source dataset and use the SCLM migration utility to migrate them into SCLM. Make sure you migrate them conditionally and each time specify the correct language.
6. For each language extract dataset, create the ARCHDEFs for the members in the dataset based upon the model ARCHDEF for that language and copy and migrate the created ARCHDEFs into SCLM.
7. Build all members or build sample of members if only using copy exit.
8. Perform some testing on the newly created members.
9. Modify existing JCL and PROCs to point to SCLM datasets or create a copy exit.
10. Discontinue use of the old library product and begin using SCLM.

Migrate conditionally

By performing the migration conditionally, you are building up the members in the source dataset by language without affecting the members that were already migrated previously with a different language. A conditional migrate does not affect members that already have a valid language. *Therefore. make sure that the copy and migrate steps are done in pairs by language type.*

COPY language

In this example migration plan, all copybooks and DCLGENs are to be used by COBOL programs. These members need to be parsed when they are saved in SCLM to keep track of any imbedded COPY statements within them so they must be handled by a language that has a COBOL parser. We assigned the copybooks and DCLGENs a language of COPY. The COPY language only has a single step in it and that is the COBOL parse step. The COPY language is equivalent to taking a simple COBOL batch language and removing the COBOL compile step. COBOL copybooks and DCLGENs can be assigned your COBOL batch language, but it may make more sense to handle them with the more simple COPY language since they do not need to be compiled in any way.

Archived

Migration timing

When you are creating a migration plan, you must estimate how long each step will last and how long the migration in total will last. The length of the migration may determine when you will perform it, if you will need a delta migration or if you can just freeze everyone out of the library until the migration is completed, and so on. You will therefore want to perform some test migrations of 100 members or so, for example, to determine how long it will take per member to migrate. Typical timings are 5 seconds per member although significantly slower or faster migrations are possible depending on your CPU's speed and load.

Once you know your timing per member, you can multiply it by the total number of members you have to determine your total minimum migration time. Then you must perform similar calculations for ARCHDEF creation and migration. Of course you have to plan for some time between jobs to check results and debug problems. If you are building your programs, you have to do timings on build time and estimate your total build time. If you are doing a delta migration, you must add in an estimate for it. When you add it all up, you may find that the migration time will last several days or even weeks, which must be figured into your migration plan and schedule.

Ordering your migration

Depending upon your needs some of these migration steps can be performed in a different order and some in parallel. For example:

- While a migration job is running for COBOL code you could be creating the ARCHDEFs.
- If you are modifying your JCL, PROC and Control Cards to point to SCLM datasets then you can make the changes in external datasets while the other migration tasks are being done and copy and migrate them into SCLM near the end of the migration.
- You can run the migration utility on members types that have a large number of members after hours when they can run faster and without supervision.

4.3 Creating alternate MIGRATE project definition

During the migration to SCLM, you will want to migrate all of your members in production into SCLM. The SCLM migration utility can only be run in a group where members can be updated and they can only be edited in the edit or lowest groups of a hierarchy. Therefore for the migration to production it will be necessary to create a "MIGRATE" alternate that only has your production group in it. This makes the production group the lowest (and only group) in the hierarchy and allows changes to be made to members in it.

Follow these steps to create a MIGRATE alternate:

1. Create a member named MIGRATE in your projid.PROJDEFS.SOURCE dataset.
2. Copy into it your base project definition.
3. In your new project definition, delete all FLMGROUP statements except for your production group and delete any FLMATVER statements for groups other than production. Then save and exit the member.
4. Modify your assemble and link job for your project and add in the steps to assemble and link the MIGRATE alternate. You have to change the job to assemble your MIGRATE source project definition and to save the load module with name MIGRATE in your SCLM load library. Execute the job. Verify that this job has created a member named MIGRATE in your projdefs.load dataset.

Once you do this, you can use the alternate language definition by specifying MIGRATE in the alternate field on the SCLM main menu and your production group in the group field.

4.4 Production freeze versus delta migration

When it is time to move from your current library product to SCLM, you have to plan on how you are going to perform that transition. It takes time to perform the migration to SCLM after you have extracted all of your members from your current product and during that time you do not want to lose track of code that is being changed while the migrate jobs run.

You can either implement a production freeze while the migration is on going so that there will be no changes to track, or you must keep track of all changes to production members while the migration jobs are running. Then when the migration jobs end, you have to do a second shorter migration to SCLM. In this second migration or delta migration you copy all members that were changed during the initial migration to SCLM and run a conditional SCLM migration on them. It will not be necessary to specify a language for the member. The current language will be used for the member. SCLM will just update the accounting records. During this delta migration it will be necessary to have a freeze. In fact once the freeze starts your current product will never need to be used again because once the delta migration ends you can go live with SCLM.

4.5 Separating your source code into extract datasets

This is a very time consuming but very critical step of the migration process. The members need to be separated in extract datasets for several reasons.

First, all members that are copied into SCLM need to have a SCLM language associated with them. This is performed by the SCLM migration utility. The migration utility must be run for each and every member that is being moved into SCLM and for each member you must supply the correct language that the member is going to have in SCLM. If you have thousands of members you could copy them all into SCLM and then run the utility thousands of times, once for each member.

This approach would be very time consuming and labor intensive. A better way would be for you to copy all members of a particular language — that is, all members of a language extract dataset — into SCLM and then run the migration utility with a wildcard (*) in the member field and the particular language in the language field. As long as you run a conditional migration, only the newly copied members will be migrated into SCLM with the specified language. Existing members in SCLM will not be affected. So you can repeat the copy and migrate actions for each set of language members without affecting members already in SCLM.

The second reason you need to separate out the members by language is that you must create ARCHDEFs for each member that needs to be linked, and the ARCHDEFs must be created by language type. If the members are separated into a set of datasets by language then it is a simple matter to create a member list for each dataset to find the list of members by type that the ARCHDEFs must be created for. This is discussed in 4.8.2, “Creating ARCHDEFs” on page 128.

Archive

The separation process consists of several steps:

1. Determine an inventory of what is to be moved into SCLM and plan for the mapping to SCLM. This process is covered in section 4.5.1, “Determining source inventory and map to extract and SCLM datasets” that follows.
2. Create a set of migrate extract datasets based upon the inventory and SCLM languages and language variation they map to.
3. Copy the members by language type to the extract datasets. How you extract these members from your current library management product by language and type is up to you. This is no trivial task, but it is beyond the scope of this book to discuss it, as the process is specific to particular library management products.

4.5.1 Determining source inventory and map to extract and SCLM datasets

This step consists of creating an inventory of all members to be migrated into SCLM and where they are currently located and then mapping that inventory to the extract datasets and then mapping the extract datasets to SCLM by type and language. Here is an example of an inventory mapping.

Table 4-1 is an example for a Panvalet migration. You must come up with a similar mapping for your conversion. In the first and second columns are your source datasets and member counts. The member counts will be useful when you are doing your migration timing estimates and when you want to verify copy and migrate counts. In the third column are your migrate extract datasets that you create with a name of your choice. By convention, we put the language in the last node of the dataset name. For the text type members, we use the SCLM type as the last node. In the fourth column we put the SCLM datasets that the extract datasets map to. In the last column, we specify the SCLM language for the members in each extract dataset.

Table 4-1 Example Panvalet migration plan

Source	Count	Migrate Dataset	SCLM Dataset	SCLM Language
PROD.SOURCE.PANLIB	895	SCLM.MIGRATE.COPYLIB	SCLM.PROD.COPYLIB	COPY
PROD.SOURCE.PANLIB	240	SCLM.MIGRATE.DCLGEN	SCLM.PROD.DCLGEN	COPY
PROD.SOURCE.PANLIB	127	SCLM.MIGRATE.ASM	SCLM.PROD.SOURCE	ASM
PROD.SOURCE.PANLIB	1792	SCLM.MIGRATE.COBOL	SCLM.PROD.SOURCE	COB
PROD.SOURCE.PANLIB	349	SCLM.MIGRATE.COBDYN	SCLM.PROD.SOURCE	COBDYN
PROD.SOURCE.PANLIB	651	SCLM.MIGRATE.COBDB2	SCLM.PROD.SOURCE	COBDB2
PROD.SOURCE.PANLIB	406	SCLM.MIGRATE.COBDB2C	SCLM.PROD.SOURCE	COBDB2C
PROD.SOURCE.PANLIB	552	SCLM.MIGRATE.COBCICS	SCLM.PROD.SOURCE	COBCICS
PROD.JCL.PANLIB	5534	SCLM.MIGRATE.JCLLIB	SCLM.PROD.JCLLIB	TEXT
PROD.PARM.PANLIB	1595	SCLM.MIGRATE.PARMLIB	SCLM.PROD.PARMLIB	TEXT
PROD.PARM.PROCLIB	366	SCLM.MIGRATE.PROCLIB	SCLM.PROD.PROCLIB	TEXT

4.5.2 Removing and/or modifying proprietary library code

Any proprietary statements such as -INC and ++INCLUDE statements in the extracted code must be converted to statements that can be handled by normal compilers. For example, these statements could be converted to COBOL COPY statements if they are found in COBOL code.

4.6 Copy exit versus JCL changes

One of the key decisions you will make as you move to SCLM is if you want to execute your members out of your new SCLM datasets or out of your existing libraries (load, JCL, PROC and control, and so on.) A copy exit can be written as described in the User Exit chapter that will copy your executable members from SCLM to your existing datasets when they are changed in SCLM via a build or a promote. Note that a copy exit can only copy text members such as JCL or PROCs during a promote. They will not be processed by a build exit.

There are several advantages to using a copy exit:

- You do not have to build all of your members to recreate all of your load modules in SCLM.
- You do not need to rebind all of your DB2 programs.
- You do not need to test as many of your newly created load modules.
- You do not need to modify your JCL, PROCs and CICS start up JCL to point to the new SCLM load, PROC, and control card datasets.
- You do not need to reschedule all of the jobs in your job scheduler to execute out of the SCLM datasets.
- You do not need an execution as well as a code freeze while you are performing the final steps of your migration to SCLM.

These are the disadvantages of using a copy exit:

- You must maintain two versions of each member, one in SCLM and one in your old libraries.
- Copy exits can fail and therefore must be verified after each build and promote in SCLM.
- There may be an audit issue with having two versions of each dataset as the external dataset could be potentially changed outside of the library process.

For most customers, the advantages of a copy exit outweigh the disadvantages of it. This is particularly true for establishments with an large installed base of code.

Archived

4.7 Running the SCLM Migration Utility

The SCLM Migration Utility is option 3 on the SCLM Utilities menu. It is used to register members in SCLM and performs all parsing and dependency scanning necessary for the member. To run it, fill in the group and type of the member or members you want to migrate into SCLM. The members should already be in the dataset (project.group.type). For mass migrations the member name can be *. Fill in the language for the members. Make sure the mode is conditional and set up the output options based upon the mode that you want to run it, as shown in Figure 4-1.

```
                    SCLM Migration Utility - Entry Panel
Command ===>

Selection criteria:
  Project  . : SCLM
  Group  . . . PROD
  Type . . . . SOURCE
  Member . . . *           (Pattern may be used)

Member information:
  Authorization code  . .              Mode  . . . 1  1. Conditional
  Change code . . . . . .                             2. Unconditional
  Language  . . . . . . . COBDB2C                     3. Forced
Output control:
               Ex Sub                  Process . . 2  1. Execute
  Messages . . 1  2    1. Terminal                    2. Submit
  Report   . . 3  2    2. Printer
  Listings . . 3  2    3. Data set     Printer . . *
                       4. None         Volume  . .
```

Figure 4-1 SCLM Migration Utility panel

Figure 4-2 shows sample output messages from the migration utility. They show that two members have been successfully migrated into SCLM. You will want to ensure that you get a zero return code from your migrations.

```
FLM32101 - MIGRATION UTILITY INITIATED - 05:10:49 ON 2007/01/11
FLM06501 - TRANSLATOR RETURN CODE FROM ===> SCLM COBOL PARSE     ===> 0
FLM87107 - SAVE SUCCEEDED FOR MEMBER SCLMTST1 AT 05:10:49, CODE: 0
FLM06501 - TRANSLATOR RETURN CODE FROM ===> SCLM COBOL PARSE     ===> 0
FLM87107 - SAVE SUCCEEDED FOR MEMBER SCLMTST2 AT 05:10:49, CODE: 0
FLM32401 - MIGRATION UTILITY COMPLETED - 05:10:49 ON 2007/01/11
```

Figure 4-2 Migration Utility messages

Archive

4.8 ARCHDEF generation

Every member in SCLM that needs to be compiled and linked must have a link edit (LEC) ARCHDEF created for it. In this section we present a method to create simple LEC ARCHDEFs. You have to modify this process if you have a more complicated ARCHDEF structure. There are several steps to creating your ARCHDEFs:

1. Create a member listing for each extract dataset that contains a list of members that need ARCHDEFs created for them. This can be done by a REXX program.
2. Create a dataset to hold your generated ARCHDEFs, like SCLM.MIGRATE.ARCHDEF.
3. Create a REXX program to mass-create the ARCHDEFs.
4. Run the ARCHDEF creation REXX by ARCHDEF type for each member listing.
5. Copy and migrate the generated ARCHDEFs into SCLM.

4.8.1 Creating member listings

The easiest way to create member listings is to run a small REXX routine that creates the listing from the contents of the extract datasets. See Example 4-1 for an example REXX that does this. You must modify the LISTDS statement for the naming convention of your extract datasets and the ALLOC statement for the name of your dataset that will hold your generated ARCHDEFS. Each of the listing members will be created in the format of a high level ARCHDEF so that you can use them after member migration to build all of the members. The argument passed to the REXX is the language type that should also be part of the extract dataset name. The name of the saved listing is @lang.

Example 4-1 MEMBLIST - REXX to create member listings

```
/* REXX */
/*********************************************************************/
/* REXX PROGRAM TO CREATE A HIGH LEVEL ARCHDEF FROM ALL THE MEMBERS  */
/* OF AN INPUT PDS (PER LANGUAGE).                                   */
/*********************************************************************/
TRACE O

ARG MEMIN
SAY 'MEMIN = 'MEMIN'*'

DUMMY = OUTTRAP("CMD_OUTPUT_LINE.","*")

"LISTDS 'SCLM.MIGRATE."MEMIN"' MEMBERS"
NUMBER_OF_MEMBERS = CMD_OUTPUT_LINE.0 - 6

DO I = 7 TO CMD_OUTPUT_LINE.0
  J = I - 6
  MEMB = SUBSTR(CMD_OUTPUT_LINE.I,3,8)
  OUTPUT_LINE.J = 'INCL 'MEMB'  ARCHDEF'
END

"ALLOC DA('SCLM.MIGRATE.ARCHDEF(@"MEMIN")') F(OUTFILE) OLD"
"EXECIO * DISKW OUTFILE (STEM OUTPUT_LINE. FINIS"
"FREE FI(OUTFILE)"
SAY "CREATING HL ARCHDEF 'SCLM.MIGRATE.ARCHDEF(@"MEMIN")'"
EXIT
```

You need to run the above REXX for each extract dataset that needs ARCHDEFs created for them. See Example 4-2 for an example REXX that does this.

Example 4-2 Running member listing REXX for each language

```
/* REXX */
Trace o

"EX 'SCLM.PROJDEFS.REXX(MEMBLIST)' 'ASM'"
"EX 'SCLM.PROJDEFS.REXX(MEMBLIST)' 'COBOL'"
"EX 'SCLM.PROJDEFS.REXX(MEMBLIST)' 'COBDYN'"
"EX 'SCLM.PROJDEFS.REXX(MEMBLIST)' 'COBCICS'"
"EX 'SCLM.PROJDEFS.REXX(MEMBLIST)' 'COBDB2'"
"EX 'SCLM.PROJDEFS.REXX(MEMBLIST)' 'COBDB2C'"

Exit
```

4.8.2 Creating ARCHDEFs

You have to create a REXX routine to create your ARCHDEFs. See Example 4-3 for an example of a REXX routine to do this. You must modify this REXX to handle each of your ARCHDEF types based upon your model ARCHDEFs. See the following chapter for a discussion on model archefs. You will also need to modify the INDSN variable to point to the dataset in which you want to store your ARCHDEFs. The argument that you pass to this routine is the same language argument that is passed to the member list creation REXX documented in the previous section. If some of your programs need two ARCHDEFs, such as your DB2 programs, you have to create two versions of this REXX, one to create each ARCHDEF type needed by the language and run each of the REXX programs for each language type that needs ARCHDEFs.

Example 4-3 BUILDARC - Mass ARCHDEF creation routine

```
/* REXX */
/*********************************************************************/
/*     EXEC TO BUILD ARCHDEFS                                        */
/*********************************************************************/
TRACE O
ARG MEMIN

INDSN  = "'SCLM.MIGRATE.ARCHDEF'"
BKMEMIN = MEMIN
L = LENGTH(MEMIN)
IF L = 8 THEN DO
   MEMIN = LEFT(MEMIN,7)
   END

MEM    = @||MEMIN
MAXLEN = 80
ADDRESS ISPEXEC
"LMINIT DATAID(MAPS) DATASET("INDSN") ENQ(SHR)"

IF RC = 0 THEN DO
  "LMOPEN DATAID("MAPS") OPTION(INPUT)"
  IF RC = 0 THEN DO
    "LMMFIND DATAID("MAPS") MEMBER("MEM")"
    IF RC = 0 THEN DO
```

```
      SAY 'CREATING ARCHDEFS ... PLEASE WAIT'
      DO UNTIL RC <> 0
        ADDRESS ISPEXEC
        "LMGET DATAID("MAPS") MODE(INVAR) DATALOC(RECIN)
               DATALEN(RECLEN) MAXLEN("MAXLEN")"
          IF RC = 0 THEN DO
            KEYW = STRIP(SUBSTR(RECIN,1,4),T)
            IF KEYW = INCL THEN DO
                MAPNM = STRIP(SUBSTR(RECIN,6,8),T)
                ADDRESS TSO
                CALL ARCHOPE MAPNM
                NEWSTACK
/* Modify the following section to create the proper archdefs for each */
/* of your language types. */
     SELECT
        WHEN MEM = '@ASM' THEN DO
           QUEUE "*"
           QUEUE "* NAME: "MAPNM
           QUEUE "* LINK EDIT CONTROL(LEC) ARCHDEF FOR ASSEMBLER"
           QUEUE "*"
           QUEUE "INCLD  "MAPNM"   SOURCE      * COMPILE SOURCE"
           QUEUE "LOAD   "MAPNM"   LOAD        * LINK EDIT LOAD MODULE"
           QUEUE "LMAP   "MAPNM"   LINKLIST    * LINK LIST"
        END
        WHEN MEM = '@COBOL' THEN DO
           QUEUE "*"
           QUEUE "* NAME: "MAPNM
           QUEUE "* LINK EDIT CONTROL(LEC) ARCHDEF FOR COBOL BATCH"
           QUEUE "*"
           QUEUE "INCLD  "MAPNM"   SOURCE      * COMPILE SOURCE"
           QUEUE "LOAD   "MAPNM"   LOAD        * LINK EDIT LOAD MODULE"
           QUEUE "LMAP   "MAPNM"   LINKLIST    * LINK LIST"
        END
        WHEN MEM = '@COBCICS' THEN DO
           QUEUE "*"
           QUEUE "* NAME: "MAPNM
           QUEUE "* LINK EDIT CONTROL(LEC) ARCHDEF FOR CICS COBOL"
           QUEUE "*"
           QUEUE "INCLD  "MAPNM"   SOURCE    * COMPILE SOURCE"
           QUEUE "LOAD   "MAPNM"   LOADCICS  * LINK EDIT LOAD MODULE"
           QUEUE "LMAP   "MAPNM"   LINKLIST  * LINK LIST"
           QUEUE "COPY   CICSLINK  ARCHDEF   * INCLUDE CICS INTERFACE"
           QUEUE "LKED   LE370C"
        END
        WHEN MEM = '@COBDB2' THEN DO
           QUEUE "*"
           QUEUE "* NAME: "MAPNM
           QUEUE "* LINK EDIT CONTROL(LEC) ARCHDEF FOR BATCH DB2 COBOL"
           QUEUE "*"
           QUEUE "INCLD  "MAPNM"   SOURCE      * COMPILE SOURCE"
           QUEUE "LOAD   "MAPNM"   LOAD        * LINK EDIT LOAD MODULE"
           QUEUE "LMAP   "MAPNM"   LINKLIST    * LINK LIST"
           QUEUE "COPY   DB2LINK   ARCHDEF     * INCLUDE DB2 INTERFACE"
           QUEUE "LKED   LE370DB"
        END
```

Archived

```
        WHEN MEM = '@COBDB2C' THEN DO
          QUEUE "*"
          QUEUE "* NAME: "MAPNM
          QUEUE "* LINK EDIT CONTROL(LEC) ARCHDEF FOR CICS / DB2 COBOL"
          QUEUE "*"
          QUEUE "INCLD  "MAPNM"   SOURCE      * COMPILE SOURCE"
          QUEUE "LOAD   "MAPNM"   LOADCICS    * LINK EDIT LOAD MODULE"
          QUEUE "LMAP   "MAPNM"   LINKLIST    * LINK LIST"
          QUEUE "COPY   CDB2LINK  ARCHDEF     * INCLUDE DB2 INTERFACE"
          QUEUE "LKED   LE370DC"
        END

        OTHERWISE
      END
           CALL ARCHWR MAPNM
           DELSTACK
END
          END   /*IF INCL*/
        END   /*IF RC=0*/
       END   /*DO UNTIL*/
     END   /*LMMFIND*/
  END   /*LMOPEN*/
END   /*LMINIT*/

"LMCLOSE DATAID("MAPS")"
"LMFREE DATAID("MAPS")"

EXIT
/*********************************************************************/
/*                                                                   */
/* ARCHOPE : OPEN OUTPUT PDS MEMBER FOR WRITING ARCHDEF              */
/*                                                                   */
/*********************************************************************/
ARCHOPE:
     PARSE UPPER ARG XMAP
     XMAP1 = STRIP(XMAP)
     ARCH = "SCLM.ARCHDEF."BKMEMIN"("XMAP1")"
    "ALLOC DA('"ARCH"') FI(ARCHDEF) SHR REUSE"
     RETURN RC

/*********************************************************************/
/*                                                                   */
/* ARCHWR : WRITE ARCHDEF MEMBER TO PDS                              */
/*                                                                   */
/*********************************************************************/
ARCHWR:
     PARSE UPPER ARG XMAP
     DO WHILE QUEUED() > 0
            "EXECIO 1  DISKW  ARCHDEF"
     END
    "EXECIO 0  DISKW  ARCHDEF (FINIS"
     RETURN RC
/*********************************************************************/
/*                                                                   */
```

Archived

```
/* LECOPE : OPEN OUTPUT PDS MEMBER FOR WRITING LECDEF                */
/*                                                                   */
/*********************************************************************/
LECOPE:
     PARSE UPPER ARG XMAP
     XMAP1 = STRIP(XMAP)
     ARCH = "SCLM.LECDEF."BKMEMIN"("XMAP1")"
    "ALLOC DA('"ARCH"') FI(LECDEF) SHR REUSE"
     RETURN RC

/*********************************************************************/
/*                                                                   */
/* LECWR : WRITE LECDEF MEMBER TO PDS                                */
/*                                                                   */
/*********************************************************************/
LECWR:
     PARSE UPPER ARG XMAP
     DO WHILE QUEUED() > 0
            "EXECIO 1  DISKW  LECDEF"
     END
    "EXECIO 0  DISKW  LECDEF (FINIS"
     RETURN RC
return
```

You have to execute the above REXX for each language. This REXX uses the member listings created in the prior step so do not run it before running the member listing REXX.

Example 4-4 is an example REXX of calling your ARCHDEF generating REXX for each language.

Example 4-4 REXX to run mass ARCHDEF creation REXX

```
/* REXX */
Trace o

"EX 'SCLM.PROJDEFS.REXX(BUILDARC)' 'ASM'"
"EX 'SCLM.PROJDEFS.REXX(BUILDARC)' 'COBOL'"
"EX 'SCLM.PROJDEFS.REXX(BUILDARC)' 'COBDYN'"
"EX 'SCLM.PROJDEFS.REXX(BUILDARC)' 'COBCICS'"
"EX 'SCLM.PROJDEFS.REXX(BUILDARC)' 'COBDB2'"
"EX 'SCLM.PROJDEFS.REXX(BUILDARC)' 'COBDB2C'"

Exit
```

Archived

4.9 Cutover to SCLM

The final step of your migration to SCLM is the cutover to SCLM. This is when you stop using your current product and begin using SCLM. Here are some considerations for this cutover.

Planning

You must have held planning meetings with the staff responsible for all aspects of code management and execution including your librarian, DBAs, operations, management, etc. All potential aspects of the cutover to SCLM should be discussed and you have to come up with a migration plan in some detail for the last few days leading up to the cutover.

Training

You must have trained all of your developers that are to use SCLM in the use it. The training sessions will have covered, at a minimum, how to edit, build, test, and promote their code. Training should also cover how to create a new member in SCLM including how to create a ARCHDEF for it. You must document your hierarchy, groups, types, languages and ARCHDEF variations. You will also need to document how to use special processing that you set up such as copy exits, bind processing, or the use of Breeze.

Code freeze

You will want to lead up to the cutover with some sort of source code freeze of your current library product. This freeze could last the entire time of your migration to SCLM or can last just during your final delta migration.

Clearing out the test groups

It is best to have all intermediate groups cleared out before you migrate to SCLM. For example, if you have code in test in a TEST or QA group then you should plan your migration for a time when you have no members in these groups or as few members in these test groups as possible. There are several reasons for this. First, it adds to the migration time and effort to move intermediate code into SCLM. Second, it is difficult to move members into the middle of the hierarchy as you will see below. Third, and perhaps most importantly, you may have to repeat many of the tests you have done on these members. This is because you will have a brand new untested load module in SCLM, as the code will have been newly built.

If you do have members that need to be moved into the intermediate test groups, then you have to wait until your production migration (main and delta) are completed to move them into SCLM. Then starting from the top group downward, perform the following steps:

1. Extract the members in test out of your old library.
2. Copy them to the development group in SCLM.
3. Edit and save them in SCLM to register them with SCLM or user the SCLM Migration Utility to do this.
4. Build their ARCHDEF. If the members are new to SCLM, then you must create a ARCHDEF for them before you can build them.
5. Promote them to the group in SCLM corresponding to the level they were at in your old product.

After you have done this for the intermediate members that are in the group closest to the production group, you can repeat this for the intermediate members in the next closest group until all intermediate members have been migrated into SCLM.

In flight development code

We suggest that you let your developers move any code changes they are working on into SCLM when it first goes live. Their last act in your current library product can be to extract the code they are in the process of changing into their personal datasets. Then when SCLM is live, they can perform the following steps for each of their members to bring them into SCLM.

1. Edit the production member of the member to the development group.
2. Delete all lines of code in the edit session.
3. Copy into the edit session the newer version of the member that the developer had saved to their private dataset.

Security

You must verify that all security (RACF, EAC) is in place for your team (developers, testers, operations, DBAs) and for the Breeze started task, etc. to use SCLM and at the same time plan on removing the update access to your old product. If developers are binding their DB2 programs in SCLM, you have to ensure that they have bind authority.

Archiving

If your old product stores its records in a proprietary format you may need to extract all old code and audit records to a format that is readable outside of the tool so that when your old tool is no longer on your system you can still have access to the old records.

Backups

Since SCLM will be maintained in a new set of datasets you must ensure that your backups (daily, weekly, monthly, etc.) are in place for the new SCLM datasets.

Disaster recovery

You may want to examine your disaster recovery plan for references to your library product and update it as needed to reference SCLM.

Breeze considerations

If you have installed Breeze, you will want to ensure that all approvers, approver groups, and junction records are in place for the approval process that you have come up with, that all approvers are aware of how to approve packages, that all users have update access to the Breeze dataset, that the Breeze exits are in place and that the sweep job has been scheduled if needed.

Using new load modules

If you have decided to implement a copy exit and continue to use your current load modules in you current load library then your migration effort will be limited to your source migration and ARCHDEF creation.

Archived

However, if you have decided to execute out of your new SCLM load libraries once you go live with SCLM, then you will have some ,additional migration steps:

1. You have to recreate all of your load modules in SCLM by building all of your programs. You must plan for enough time to perform all of the builds and to debug all of the compile and link problems you will have. For all DB2 programs, the build jobs must bind all newly built programs just before you are ready to run them.
2. You have to modify all JCL and PROCs to point to SCLM datasets and then reschedule them in your job scheduler to run out of SCLM libraries.
3. You have to bring down your production CICS regions, modify their startup JCL to point to the SCLM load libraries and then start the regions again.
4. You have to test a subset of the newly created load modules by running some of your production jobs and transactions and compare the results to the same jobs and transactions running in your old environment.

Archived

5

SCLM architecture definition considerations

An architecture definition (ARCHDEF) describes the configuration of a member or group of members under SCLM control and how they are to be built and promoted. They are a key component of the configuration management functionality of SCLM. They perform the following functions in SCLM:

- Define how to process a member or group of members.
- Specify inputs and outputs for each language translator.
- Specify option overrides for language translators.
- Determine how members are tracked during the build and promote operation.

An ARCHDEF is usually required for every member that must be linked.

Architecture definitions can reference other architecture definitions, thus providing a simple building block tool for complex application definitions. An ARCHDEF that includes another ARCHDEF is called a high-level ARCHDEF.

ARCHDEFs control builds and promotes

Architecture definitions are the controlling member for the SCLM Build and Promote operations. They can specify the following information to the build function:

- Where inputs to translators (for example, compilers) are to come from
- Where outputs from translators are to be stored
- What parameters are needed by a translator

All data that is directly or indirectly referenced by an architecture definition is promoted when that architecture definition is promoted. This encompasses included architecture definitions, along with the system components they describe and other associated components. Thus, specifying a single high-level architecture definition for promotion can cause an entire application or fix to an application to be promoted.

© Copyright IBM Corp. 2007. All rights reserved.

5.1 Types of ARCHDEF members

There are four kinds of SCLM architecture definition members: high-level ARCHDEFs, link-edit ARCHDEFs, compilation control ARCHDEFs, and generic ARCHDEFs.

5.1.1 High-level (HL) ARCHDEF

High-Level (HL) architecture members are used to package various members together for processing. Therefore they can be used to define complete applications and sub-applications. They are also commonly used to define fixes to these same applications. They allow you to categorize groups of related load modules, object modules, and other software. They can also refer to and include non-compilable members associated with the application such as JCL, PROCs, HTML, XML, documentation, and so on.

By using an HL architecture definition as input to the build or promote functions, you can ensure that the entire application is up-to-date or is promoted to the next group in the project hierarchy. A build or promote of an HL architecture member results in the building or promotion of every software component referenced. In this way, you can guarantee the integrity of an entire application.

A high-level ARCHDEF has the following format as shown in Example 5-1.

Example 5-1 Format of high-level ARCHDEF

```
* comment lines - describe the content of the archdef
*
INCL  memberxx  archtype    * comments
*
PROM  memberyy texttype
*
INCLD memberzz sourcetype  * comments
*
```

Comments can be included into ARCHDEFs in two ways:

- Any line beginning with a * is a comment.
- Any characters to the right of the last argument is treated as a comment. Using an '*' is not necessary, but helps to separate out the comments for readability. Do not add comments to PROM statements.

Programs that need to be compiled and linked should be included into the high-level ARCHDEF with an INCL of the member's ARCHDEF. The ARCHTYPE variable should be replaced with your ARCHDEF type, for example, ARCHDEF.

Programs that are processed with languages that are self contained (no link edit is needed, or the link is integrated into the language) should be included into the high-level ARCHDEF with an INCLD. The sourcetype variable should be replaced with the member's type where the source is stored.

Members that have no language processing should be included into the ARCHDEF with a PROM statement. PROM refers to PROMOTE. These members will be promoted without processing.

Archive

Example: High-level ARCHDEF

Example 5-2 is an example of a high-level ARCHDEF. It is for an SCLM project that stores high-level ARCHDEFs in the PACKAGE type. In the PACKAGE type we have a high-level ARCHDEF for some code that is being fixed for change control number 2007601. This number was defined in a change control product and represents the 601st change for the year 2007. In this example we use this change control number as part of the package name. All ARCHDEFs are stored in the SCLM hierarchy and therefore their name must conform to standard TSO naming conventions thus they can not be all numeric. The change control number is therefore prefixed with a 'P' to create a proper member name.

Other ARCHDEFs, most commonly link-edit ARCHDEFs, are included into the package with an INCL keyword, and members that do not need to be built are included into the ARCHDEF using the PROM keyword. Comment lines are included into the ARCHDEF by starting the line with a '*'.

Example 5-2 Example high-level ARCHDEF

```
* NAME: P2007601
* DESC: PACKAGE FOR 2007601
* DEFECT: 2007601
*
* ADD SOME NEW FUNCTIONALITY AND FIXING A DEFECT
*
*    ARCHDEF MEMBERS
*
INCL  SCLMTST1   ARCHDEF      * Batch Program
INCL  SCLMTST2   LECDEF       * Batch DB2 Program
INCL  SCLMTST2   LECDEF       * Online DB2 Program
INCL  SCLMTST3   ARCHDEF      * Online Program
*
*    JCL MEMBERS
*
PROM  PAY63702   JCLLIB       * Job Control Language
*
*    DOC MEMBERS
*
PROM  P2007601   DOC          * Documentation
```

5.1.2 Link-edit control ARCHDEFs

Standard linkage editors produce load modules as output. To define software components with load module outputs from standard linkage editors, use Linkedit Control (LEC) architecture members. LEC architecture members contain all the information necessary to produce a complete load module. They identify the following items:

- The input sources to the language translators whose objects are to be linked together into the load module
- Other load modules the load module is to contain
- The load module name and the type in which you want it saved
- The linkage editor listing name and the type in which you want it saved
- Linkedit control statements and linkage editor options.

LEC architecture members must have at least one LINK, INCL, INCLD, or SINC statement and one LOAD statement. Linkedit Control (LEC) architecture members can be constructed by referencing any combination of source members, CC architecture members, generic architecture members, or LEC architecture members.

Archived

Inputs to LEC architecture members are identified in the same way that inputs to CC architecture members are identified. The one difference is that by default, LEC architecture members include object and load modules generated by the OBJ and LOAD statements in the input stream to the linkage editor.

SINC statements can be used in LEC architecture members to identify object modules or load modules which are generated outside of the project. If SINC statements are being used to include load modules, the input ddname for the build translator must specify KEYREF=INCL.

The LINK statement can also be used to identify an input to the linkage editor. It identifies an output of another ARCHDEF in the project that does not need to be rebuilt before being included in the input stream of the linkage edit. SCLM usually verifies that the inputs to the LEC architecture member are up to date before link-editing the inputs. SCLM will rebuild any inputs that are outputs of building other members in the project when those outputs are out-of-date. The inputs specified on LINK statements are an exception. These inputs will not be rebuilt.

You can override default linkage editor options by using the PARM statement. Use the statement as many times as necessary to specify all options you want.

SCLM uses the standard S/370 linkage editor as defined by the LE370 language definition unless an LKED statement is used to override the default.

You can specify in the LEC ARCHDEF that SCLM pass linkage edit control statements directly to the linkage editor by using the CMD statement. Insert the control statements along with the object and load modules by careful positioning in the LEC architecture member. The CMD statement can be used to include object modules and load modules that are in data sets outside of the project.

The format of an LEC ARCHDEF is shown in Example 5-3.

Example 5-3 Format of LEC ARCHDEF

```
* comment lines
* comment lines
INCLD memberxx sourcetype * include main module OR
INCL  memberxx ccdeftype  * include main module via CCDEF
INCLD memberyy sourcetype * include any statically linked subroutines
CMD   ...                 * send commands to linkage editor
*                         * commonly used to include external load modules
COPY  copymbr  archtype   * used to include statements from other archdefs
*                         * usually used to include commonly used CMD statements
LOAD  memberxx loadtype   * write out load module; must be included
LMAP  memberxx linklist   * write out link listing
LKED LEzzz                * use alternate link edit language
PARM zzz
```

Just like a high-level ARCHDEF, a comment can be included into ARCHDEFs in two ways, starting a line with a * to make the whole line a comment or adding comment to the end of any line. Do not add comments to PARM statements.

The main module should be included into the LEC ARCHDEF with the first INCLD of the members source type. Enter the sourcetype variable as the type where the source member is included. Alternately, if the source program has a compilation control ARCHDEF, it should be included into the LEC ARCHDEF with an INCL of the member's CCDEF.

Any statically linked subroutines should be similarly included with an INCLD statement. Subroutines can also be included with an INCL of its CCDEF if it has one.

External load modules should be included with a CMD INCLUDE SYSLIB statement. Any other linkage editor commands such as a ORDER statement can also be included with a CMD statement.

Common CMD statements for a collection of programs can be grouped together into an ARCHDEF and included into the LEC ARCHDEF with a COPY statement.

A LOAD statement must be included into the LEC ARCHDEF. Follow the LOAD statement with the load module name and then the load module type. The name of the load module does not have to match the name of any included source or of the ARCHDEF itself. Usually though the name of the ARCHDEF, the load module and the main source program will match.

An LMAP statement can be used to save the link listing if you set up your project and link language to store link listings.

An LKED statement can be used to override the default linkage editor (LE370).

A PARM statement can be used to override default linkage editor statements.

Example: LEC ARCHDEFs in the ARCHDEF type

Example 5-4 and Example 5-5 are examples of LEC ARCHDEFs. They are stored in both the ARCHDEF type and the LECDEF type. In the ARCHDEF type we have stored the four ARCHDEFs referenced in the previous example package.

Example 5-4 Example LEC ARCHDEF for a batch program

```
* NAME: SCLMTST1
* DESC: LINK EDIT CONTROL (LEC) ARCHDEF FOR COBOL
*
INCLD  SCLMTST1  SOURCE     * CREATE OBJECT
LOAD   SCLMTST1  LOADLIB    * CREATE LOAD MODULE
LMAP   SCLMTST1  LMAP       * LINK-EDIT LISTING
```

The above ARCHDEF is an LEC ARCHDEF. The INCLD keyword specifies the source program to be compiled and linked into the load module. The source is stored in the SOURCE type. The name of the load module is specified with the LOAD statement and is stored into the LOADLIB type. LMAP points to the link listing and is stored in the LMAP type.

Example 5-5 Example DB2 LEC ARCHDEF for a batch program

```
* NAME: SCLMTST2
* DESC: HIGH LEVEL (HL) ARCHDEF FOR COBOL DB2
*
INCL   SCLMTST2  LECDEF     * COMPILE AND LINK
INCLD  SCLMTST2  BIND       * INITIATE DB2 BINDS
```

The above ARCHDEF is a special high-level ARCHDEF. It is for a DB2 program and a DB2 program needs to have two functions performed for it, a build and a bind. If you have set up SCLM to have binds controlled from an ARCHDEF, then you should have set up your ARCHDEF similar to this.

This ARCHDEF includes the control members for the two functions that are to be performed, an LECDEF member to perform the build and a BIND member to perform the BIND. Normally high-level ARCHDEFs by convention are not stored in the ARCHDEF type, but rather in another type such as the PACKAGE type. In this case this high-level ARCHDEF is used to build a single load module as are LEC ARCHDEFs, and when built, it performs all necessary actions for the DB2 programs, the build and the bind. Therefore we decided to include the DB2 ARCHDEFs in the ARCHDEF type. This is where the developers expect to find all the members to build single load modules, as shown in Example 5-6.

Example 5-6 Example DB2 LEC ARCHDEF for a online program

```
* NAME: SCLMTST3
* DESC: HIGH LEVEL (HL) ARCHDEF FOR ONLINE COBOL DB2
*
INCL   SCLMTST3  LECDEF     * COMPILE AND LINK
INCLD  SCLMTST3  BIND       * INITIATE DB2 BINDS
```

The foregoing exampleis also a special DB2 high-level ARCHDEF, but it is for an ONLINE program. The ONLINE differences are specified in the LECDEF member as shown in Example 5-7.

Example 5-7 Example LEC ARCHDEF for a ONLINE program

```
* NAME: SCLMTST4
* DESC: LINK EDIT CONTROL (LEC) ARCHDEF FOR ONLINE COBOL
*
INCL   SCLMTST4  CCDEF      * CREATE OBJECT
LOAD   SCLMTST4  CICSLOAD   * CREATE LOAD MODULE
LMAP   SCLMTST4  LMAP       * LINK-EDIT LISTING
COPY   CICSLINK  ARCHDEF    * CICS INTERFACE
LKED LE370C
```

The above ARCHDEF is an LEC ARCHDEF for an online (CICS) program. This ARCHDEF has one new feature, which is an include for a CCDEF. Compare how the line for the CCDEF is coded versus the first ARCHDEF above, which has an INCLD statement to bring in the source. Whenever you need to code a CCDEF to override a compile PARM, you must change the ARCHDEF to point to it. Change the INCLD that includes the SOURCE to an INCL for the CCDEF.

This ARCHDEF also has two additional statements. The COPY statement includes a common ARCHDEF that includes statements that must be included into each CICS program. These include the CMD statement that links in the CICS interface module. The ARCHDEFs referenced by the COPY statements are documented below.

The LKED statement specifies that the LE370C link language is to be used to link this member instead of the default LE370 link language. LE370C is identical to LE370 except that it has the CICS load libraries in the SYSLIB concatenation.

Common ARCHDEFs in the ARCHDEF type

The following three common ARCHDEFs are included into our example LEC ARCHDEFs. They are stored in the ARCHDEF type.

Member CICSLINK:

```
CMD    INCLUDE   SYSLIB(DFHECI)
CMD    ORDER     DFHECI
```

Member DB2LINK:

```
CMD    INCLUDE   SYSLIB(DSNELI)
```

Member DB2CLINK:

```
CMD    INCLUDE   SYSLIB(DSNCLI)
```

Example: LEC ARCHDEFs in the LECDEF type

A second ARCHDEF is sometimes needed to define the processing for programs and in our examples they are needed for the DB2 programs. For our example these ARCHDEFs are stored in the LECDEF type. In this type we have the two LECDEFs referenced by the example ARCHDEFs as shown in Example 5-8.

Example 5-8 Example LEC ARCHDEF for a DB2 program stored in the LECDEF type

```
* NAME: SCLMTST2
* DESC: LINK EDIT CONTROL (LEC) ARCHDEF FOR COBOL DB2
*
INCLD  SCLMTST2  SOURCE            * CREATE OBJECT
LOAD   SCLMTST2  LOADLIB           * CREATE LOAD MODULE
LMAP   SCLMTST2  LMAP              * LINK-EDIT LISTING
COPY   DB2LINK   ARCHDEF           * DB2 INTERFACE
LKED LE370DB
```

The above ARCHDEF is an LEC ARCHDEF for a DB2 program. Its only difference compared to the first ARCHDEF is that this ARCHDEF has two additional statements. The COPY statement includes a common ARCHDEF that includes statements that must be included into each batch DB2 program. These include the CMD statement that call the INCLUDE statement to link in the DB2 interface module.

The LKED statement specifies that the LE370D link language is to be used to link this member instead of the default LE370 link language. LE370D is identical to LE370 except that it has the DB2 load libraries in the SYSLIB concatenation as shown in Example 5-9.

Example 5-9 Example LEC ARCHDEF for a online DB2 program stored in the LECDEF type

```
* NAME: SCLMTST3
* DESC: LINK EDIT CONTROL (LEC) ARCHDEF FOR ONLINE COBOL DB2
*
INCL   SCLMTST3  CCDEF             * CREATE OBJECT
LOAD   SCLMTST3  CICSLOAD          * CREATE LOAD MODULE
LMAP   SCLMTST3  LMAP              * LINK-EDIT LISTING
COPY   DB2CLINK  ARCHDEF           * DB2 INTERFACE FOR CICS
COPY   CICSLINK  ARCHDEF           * CICS INTERFACE
LKED LE370DC
```

The above ARCHDEF is an LEC ARCHDEF for an online (CICS) DB2 program. This ARCHDEF also has a include for a CCDEF. In addition, the COPY statements includes the common ARCHDEFs that includes statements that must be included into each CICS / DB2 program. These include the CMD statement that links in the CICS interface and the DB2 CICS interface for DB2 modules.

The LKED statement specifies that the LE370DC link language is to be used to link this member instead of the default LE370 link language. LE370DC is identical to LE370 except that it has the CICS and DB2 load libraries in the SYSLIB concatenation.

Archived

5.1.3 Compilation control architecture members

Including a compilable source member in a link-edit ARCHDEF using an INCLD statement is the usual method of creating object modules. The language definition of the source member from the project definition determines which translators are called and where outputs are saved during the build.

The other method of creating object modules is through calling Compilation Control architecture definitions (CCDEFs) from the link-edit ARCHDEF using an INCL statement. They are most commonly used to override compile parameters for individual translator steps. They can also be used to rename outputs.

When used, CCDEFs provide:

- The inputs to the compiler and other translators
- The outputs of the compiler and other translators and what they are to be named
- Compiler option overrides

SINC statements are used to identify input to the compiler. CC architecture members must have at least one SINC statement and one output (OBJ) statement.

You can append translator options to the options specified in the language definition by using the PARM statement. Use the statement as many times as necessary to specify all options you want (up to a string length of 512 characters). You can pass parameters directly to specific build translators defined in the language definition by using the PARM*x* statement, coupled with the use of the PARMKWD parameter of the FLMTRNSL macro.

SCLM orders compiles to ensure that outputs (such as DB2 DBRMs) are produced before building a member that references them. SCLM only orders compiles that are within the scope of the build.

SCLM generates compiler listings to temporary listing data sets so that you can view them online during the build. You can save these listings to members in the database by using the optional LIST architecture member statement. If you do not specify the LIST statement, the online compiler listings are not saved.

Example 5-10 shows the format of a compilation control ARCHDEF.

Example 5-10 Format of compilation control ARCHDEF

```
* comments
*
SINC  memberxx  sourcetype   * comments
OBJ   memberxx  objectype    * comments
LIST  memberxx  listing type * comments
OUTx  memberxx  dbrmlibtype  * comments
OUTx  memberxx  othertype    * comments
PARMx overriding parameters
```

Just like the other ARCHDEFs, a comment can be included into CCDEFs in two ways, starting a line with a * to make the whole line a comment or adding comment to the end of any line. Do not add comments to PARM statements.

The program should be included into the CCDEF ARCHDEF with a SINC of the members source type. The sourcetype variable should be entered as the type where the source member is included.

An OBJ statement must be included. Specify the member name and type to save the compile output object.

A LIST statement must be included if the language includes the LIST keyref to indicate the name and type to save the compile listing.

OUTx statements must be included for all OUTx keyrefs in the language to indicate the name and type to save these outputs. OUT1 is commonly used for DB2 languages for the DBRMLIB output.

A PARMx statement can be included to override any compile options in language steps that have a PARMKWD. Specify the correct PARMx for the language step you want to override followed by the parameter value you want to override.

Example CCDEFs

Example 5-11 and Example 5-12 are examples of two CCDEFs. They are the CCDEFs that are called by the two ARCHDEFs above. They are stored in the CCDEF type and are used to override the default DYN compiler parameter.

Example 5-11 Example CCDEF for a DB2 program

```
* NAME: SCLMTST3
* DESC: COMPILATION CONTROL (CC) ARCHDEF FOR SCLMTST3
* INCL: POINTED TO BY LEC ARCHDEF SCLMTST3
*
SINC  SCLMTST3  SOURCE
OBJ   SCLMTST3  OBJ
LIST  SCLMTST3  SRCLIST
OUT1  SCLMTST3  DBRMLIB
OUT2  SCLMTST3  SYSDEBUG
PARM1 NODYN
```

The example above shows a CCDEF for a DB2 program. If you compare this CCDEF to the one below, you will see that it has a statement for OUT1, while the one below does not. This points out that CCDEFs need to be customized to the language type of the program that the CCDEF is being written for. A CCDEF needs one line for each keyref in the language of the program. The keyref will be the same name as the keyword. In the example above, there are keyrefs for SINC, OBJ, LIST, OUT1 and OUT2 in the COBDB2 language, which is the language of SCLMTST3. After the keyword comes the member name, which is usually the same for each line, but you can rename outputs here by changing the name. After the member name is the SCLM type where the source is located for the SINC statement and the output types for each output. These are usually the same as the default type coded in the language definition, but you can change them if needed to redirect them to another type.

Example 5-12 Example CCDEF for a CICS program

```
* NAME: SCLMTST4
* DESC: COMPILATION CONTROL (CC) ARCHDEF FOR SCLMTST3
* INCL: POINTED TO BY LEC ARCHDEF SCLMTST4
*
SINC  SCLMTST4  SOURCE
OBJ   SCLMTST4  OBJ
LIST  SCLMTST4  SRCLIST
OUT2  SCLMTST4  SYSDEBUG
PARM1 NODYN
```

5.1.4 Generic architecture members

Generic architecture members are used to process members that do not generate object modules. Examples of the outputs that might be produced are documentation and panels. Generic architecture members are almost the same as Compilation Control (CC) architecture members. The difference is that generic architecture members cannot generate object modules using the OBJ statement. If an OBJ statement is added to a Generic architecture member, it becomes a CC architecture member. Other output statements such as LIST and OUT1 are used in generic architecture members to identify the listings, documentation, panels, or other outputs produced.

5.2 Understanding the ARCHDEF language

All ARCHDEF members are stored with the ARCHDEF language which processes them. The ARCHDEF language requires ARCHDEFs to follow a specified format.

ARCHDEF format

Most architecture definition statements have the format:

```
<keyword>  <member>  <type>  <comment>
```

- **<keyword>** is an SCLM reserved word. See the *SCLM Guide and Reference* for a complete list of keywords.
- **<member>** is the name of a part under SCLM control.
- **<type>** is the third qualifier of the PDS where the member resides.
- **<comment>** is user information appended to the statement.

No special character is required to indicate a comment at the end of a line.

5.2.1 Rules for coding ARCHDEFs

The rules for coding ARCHDEFs are as follows:

- Only one ARCHDEF statement per line can be used.
- Use columns 1-72 only.
- No continuation to the next line is allowed.
- Uppercase or lowercase are allowed.
- Blanks are free format except for CMD, PARM, and PARMx keywords.
- The order of statements is generally free format except when multiple inputs must be concatenated together as one input stream.

Use the MODEL command under Edit session, SCLM option 2 to see ARCHDEF keyword formats.

Archived

5.2.2 Common ARCHDEF keywords

The following list gives the common ARCHDEF keywords:

***** Comment

*** <this is a comment>**

CMD Identifies command statements as input to the linkage editor. Used to include external references, set entry points, etc.

CMD <command statement>

COPY Identifies another ARCHDEF that is to be inserted into this ARCHDEF member.
COPY <member> <type> <comment>

INCL Includes another ARCHDEF that this ARCHDEF references. Cannot reference source members directly.

INCL <member> <type> <comment>

INCLD Includes a source member that this ARCHDEF references. Used to identify inputs to a compiler.

INCLD <member> <type> <comment>

LIST Identifies member and type for compiler listing.

LIST <member> <type> <comment>

LKED Identifies an override linkage editor.

LKED <language> <comment>

LMAP Identifies member and type for link edit listing.

LMAP <member> <type> <comment>

LOAD Identifies load module name and type.

LOAD <member> <type> <comment>

OBJ Identifies object member and type for output.

OBJ <member> <type> <comment>

OUTx Identifies output member and type.

OUTx <member> <type> <comment>

PARM Parameters (options) passed to all translators in the language definition.

PARM <parameters>

PARMx Parameters (options) passed to specific translators; PARMx must be coded on one translator in the language definition.

PARMx <parameters>

PROM Identifies a text member to be promoted.

PROM <member> <type> <date check>

SINC Identifies source member and type input. Used primarily in a CC ARCHDEF.

SINC <member> <type> <comment>

Archived

5.2.3 ARCHDEFs naming convention

The following arrangement is a suggested naming convention for ARCHDEF members:

- Store all high-level ARCHDEF in the PACKAGE type for each grouped promotion from any group. Packages should be named to reflect the change or release indicator.
- Store all link-edit ARCHDEFs in the ARCHDEF type first and in an LECDEF type second. Use the LECDEF type only if the ARCHDEF in the ARCHDEF type needs to refer to a same named ARCHDEF.
- Store all compilation control ARCHDEFs in the CCDEF library using the same name as the source member. CCDEF members are required for overriding the default compiler PARMs for a program.
- Statically linked subroutines must be included with an INCLD keyword in the Link Edit Control ARCHDEFs.
- If special link link-edit control statements are needed for CICS, DB2, SYBASE, MQ Series, etc. modules then include these statements into the LEC ARCHDEF using CMD keywords or into a common ARCHDEF and then include them into the LEC ARCHDEF with a COPY statement.

5.2.4 Creating model ARCHDEFs

As you are developing your various language definitions for your SCLM project you will be testing the languages by compiling and linking using test members. You will have at least one test member for each language or language variation. Each of the test members will need their own ARCHDEFs and each of these will usually have slight variations. For example:

- If your project has Assembler, PLI and COBOL source code and you store them into different types then the INCLD statement in each LEC ARCHDEF should point to the correct source type.
- If your batch and CICS load modules are stored in different load libraries then the LOAD statement must be adjusted according.
- Different program types, such as CICS, DB2, MQ Series, will need their own combination of CMD INCLUDE statements to link in the interface modules for the program types.
- Some program variations will need parameter overrides either in LEC ARCHDEFs or in a CCDEF. In the former case, a PARM statement must be added to the LEC ARCHDEF. In the latter case, the LEC ARCHDEF must be changed to point to a CCDEF and the CCDEF must be created with PARM or PARMx statements.

By the time you have finished your language testing you will have created test ARCHDEFs for each of your test members. You may have 10, 20 or more ARCHDEF variations. You will then need to document these variations for two reasons.

1. When it comes time to mass migrate your programs into SCLM you must mass-create ARCHDEFs for each program and the ARCHDEFs must be created according to individual member needs. If you have thousands of programs this can be very difficult. It is likely however that you can break those thousands of members into subsets based upon the language type and language variation. This process was discussed in the previous chapter (in the section “Separating your source code into extract datasets”). You will likely have one extract dataset for each of the test programs and test ARCHDEFs that you created during initial project setup. These test ARCHDEFs become the model ARCHDEFs that can be used when you setup the mass migrate ARCHDEF process.
2. The other reason that you need to document your ARCHDEFs variations is so that your programmers know how to code new ARCHDEFs for new programs.

Documenting model ARCHDEFs

There are two ways to document your ARCHDEFs. First, you can document them in a word doc and then distribute the word doc to the programmers with your other SCLM procedures. The second way is to create a set of model ARCHDEFs under your SCLM PROJDEFS datasets.

Here is an example of documenting your model ARCHDEFs under your PROJDEFS datasets. All members are named after the language that these ARCHDEFs are a model for. Note that CCDEF examples are created for each language. Not all programs need them, but they should be documented so that they are available when needed.

To use the model ARCHDEFs, the programmer must:

1. Edit an ARCHDEF of the proper ARCHDEF type with the new member name.
2. In the empty edit session, execute the COPY command and point to the samples library for the ARCHDEF type.
3. Choose the member named after the language of the source member that they are creating the ARCHDEF for.
4. Once the member is copied into the edit session, do a change all command on 'XXXXXXXX' to the new member name.

The above steps should be repeated for each ARCHDEF type in the sample library for the language.

Model ARCHDEFs in SCLM.PROJDEFS.SAMPLES.ARCHDEF

Here we list the model ARCHDEFS:

Member COBOL:

```
* NAME: XXXXXXXX
* DESC: LINK EDIT CONTROL (LEC) ARCHDEF FOR COBOL
*
INCLD  XXXXXXXX  SOURCE           * CREATE OBJECT
LOAD   XXXXXXXX  LOADLIB          * CREATE LOAD MODULE
LMAP   XXXXXXXX  LMAP             * LINK-EDIT LISTING
```

Member COBCICS:

```
* NAME: XXXXXXXX
* DESC: LINK EDIT CONTROL (LEC) ARCHDEF FOR ONLINE COBOL
*
INCLD  XXXXXXXX  SOURCE           * CREATE OBJECT
LOAD   XXXXXXXX  CICSLOAD         * CREATE LOAD MODULE
LMAP   XXXXXXXX  LMAP             * LINK-EDIT LISTING
COPY   CICSLINK  ARCHDEF          * CICS INTERFACE
```

Member COBDB2:

```
* NAME: XXXXXXXX
* DESC: HIGH LEVEL (HL) ARCHDEF FOR COBOL DB2
*
INCL   XXXXXXXX  LECDEF     * COMPILE AND LINK
INCLD  XXXXXXXX  BIND       * INITIATE DB2 BINDS
```

Archived

Member COBDB2C:

```
* NAME: XXXXXXXX
* DESC: HIGH LEVEL (HL) ARCHDEF FOR ONLINE COBOL DB2
*
INCL   XXXXXXXX  LECDEF     * COMPILE AND LINK
INCLD  XXXXXXXX  BIND       * INITIATE DB2 BINDS
```

Model LECDEFs in SCLM.PROJDEFS.SAMPLES.LECDEF

Here we list the model LECDEFS:

Member COBDB2:

```
* NAME: XXXXXXXX
* DESC: LINK EDIT CONTROL (LEC) ARCHDEF FOR COBOL DB2
*
INCLD  XXXXXXXX  SOURCE           * CREATE OBJECT
LOAD   XXXXXXXX  LOADLIB          * CREATE LOAD MODULE
LMAP   XXXXXXXX  LMAP             * LINK-EDIT LISTING
COPY   DB2LINK   ARCHDEF          * DB2 INTERFACE
```

Member COBDB2C:

```
* NAME: XXXXXXXX
* DESC: LINK EDIT CONTROL (LEC) ARCHDEF FOR ONLINE COBOL DB2
*
INCLD  XXXXXXXX  SOURCE           * CREATE OBJECT
LOAD   XXXXXXXX  CICSLOAD         * CREATE LOAD MODULE
LMAP   XXXXXXXX  LMAP             * LINK-EDIT LISTING
COPY   DB2CLINK  ARCHDEF          * DB2 INTERFACE FOR CICS
COPY   CICSLINK  ARCHDEF          * CICS INTERFACE
```

Model CCDEFs in SCLM.PROJDEFS.SAMPLES.CCDEF

Here we list the model CCDEFS:

Member COBOL:

```
* NAME: XXXXXXXX
* DESC: COMPILATION CONTROL (CC) ARCHDEF FOR XXXXXXXX
* INCL: POINTED TO BY LEC ARCHDEF XXXXXXXX
*
SINC  XXXXXXXX SOURCE
OBJ   XXXXXXXX OBJ
LIST  XXXXXXXX SRCLIST
OUT2  XXXXXXXX SYSDEBUG
```

Member COBCICS:

```
* NAME: XXXXXXXX
* DESC: COMPILATION CONTROL (CC) ARCHDEF FOR XXXXXXXX
* INCL: POINTED TO BY LEC ARCHDEF XXXXXXXX
*
SINC  XXXXXXXX SOURCE
OBJ   XXXXXXXX OBJ
LIST  XXXXXXXX SRCLIST
OUT2  XXXXXXXX SYSDEBUG
```

Archived

Member COBDB2:

```
* NAME: XXXXXXXX
* DESC: COMPILATION CONTROL (CC) ARCHDEF FOR XXXXXXXX
* INCL: POINTED TO BY LEC ARCHDEF XXXXXXXX
*
SINC  XXXXXXXX SOURCE
OBJ   XXXXXXXX OBJ
LIST  XXXXXXXX SRCLIST
OUT1  YYYYYYYY DBRMLIB
OUT2  XXXXXXXX SYSDEBUG
```

Member COBDB2C:

```
* NAME: XXXXXXXX
* DESC: COMPILATION CONTROL (CC) ARCHDEF FOR XXXXXXXX
* INCL: POINTED TO BY LEC ARCHDEF XXXXXXXX
*
SINC  XXXXXXXX SOURCE
OBJ   XXXXXXXX OBJ
LIST  XXXXXXXX SRCLIST
OUT1  XXXXXXXX DBRMLIB
OUT2  XXXXXXXX SYSDEBUG
```

Archived

Archived

Part 2

Using SCLM

In this part of the book, we describe basic usage scenarios. We show you how a developer will edit, build, and promote a program. We discuss program development from a work unit perspective, using SCLM's option to drive the edit, build, and promote from an architecture definition that defines the Unit of Work. We show you some new features in SCLM relating to leaving copybooks behind at the development group and why this can be important to you.

To best utilize SCLM, we show you some SCLM best practices relating to a number of topics, including comparing versions, versioning, and administrator responsibilities. We also provide some solutions for handling SCLM issues such as B37/E37 errors and predecessor verification errors.

Discussion in the chapters in this part is not designed to replace the *SCLM Guide and Reference*, but rather to give an overview of the normal processing cycle a developer will take while using SCLM. The *SCLM Guide and Reference* talks in more depth about aspects of each of the SCLM options.

© Copyright IBM Corp. 2007. All rights reserved.

Archived

6

Using SCLM edit

In this chapter we describe the process of going into SCLM for the first time and using edit in SCLM. We also look at some of the tools you can use while in the edit session, such as the ISPF Compare tool.

The scenario used in this chapter shows you how the developer will edit a COBOL program. In the following chapters we look at further aspects of code development through building and promoting the scenario. The process of working with members in SCLM is the same regardless of what language the member is written in.

Archived

© Copyright IBM Corp. 2007. All rights reserved.

6.1 Going into SCLM for the first time

SCLM is usually found as option **10** on the ISPF primary menu. However, your systems administrator might have moved it to be another option. The SCLM primary menu can also be displayed by using the **TSO SCLM** command from any command line.

Before you can start working with a member, you require some information about the SCLM project that you are working in. Your SCLM administrator will have provided you with an SCLM project name. This is the repository location of the members that are stored in SCLM. The SCLM administrator will also have informed you of the development group where you will work on your code changes. Your environment might be set up with a single development group, a development group per team, or even a single development group for each developer. Your administrator might also have set up the SCLM project using alternate project definitions. These can provide different views and processing options within a project hierarchy. As a developer, you do not need to fully understand these concepts, but you need to know how your administrator has set up the SCLM project so that you can begin using it.

Entering the name of your SCLM project and development group can be seen in Figure 6-1.

```
------------------------------------------------------------------------------
                                SCLM Main Menu
Option ===>

Enter one of the following options:

   1  View        ISPF View or Browse data
   2  Edit        Create or change source data in SCLM databases
   3  Utilities   Perform SCLM database utility/reporting functions
   4  Build       Construct SCLM-controlled components
   5  Promote     Move components into SCLM hierarchy
   6  Command     Enter TSO or SCLM commands
   6A Easy Cmds   Easy SCLM commands via prompts
   7  Sample      Create or delete sample SCLM project
   A  SCLM Admin  Maintaining SCLM administrators
   X  Exit        Terminate SCLM


SCLM Project Control Information:
   Project . . . . SCLM07    (Project high-level qualifier)
   Alternate . . .           (Project definition: defaults to project)
   Group . . . . . DEV1      (Defaults to TSO prefix)
```

Figure 6-1 SCLM Main Menu - Entering project name and development group

Just as in ISPF, it is possible to access members in a number of different ways. You can use option **1** (View) and option **2** (Edit) from the SCLM Main Menu to specify members individually or list the members in a certain type. You can then use option **4** (Build) to compile the member and option **5** (promote) to move the member up the hierarchy. When you are working on a member, though, your normal process will be to Edit, then compile. So rather than jumping from option to option, SCLM provides a member list processing panel similar to ISPF option 3.1. For the following scenarios, we drive all the processing from this member list panel.

6.2 SCLM edit

The first thing you, as a developer, need to know how to do is edit a member that you need to work on, or create a new member. SCLM uses ISPF edit, so all of the features that you are accustomed to using in ISPF are available in SCLM edit, including the edit compare facility and edit macros.

SCLM does not use a specific check-out/check-in mode of operation. If a member exists in the SCLM hierarchy already, for example, in the production libraries, you just begin editing the member. SCLM will look up the hierarchy from the specified development group, until it finds the member. SCLM then locks the member at your development group to stop anyone else editing the same member. When you make a change to the member SCLM will, on exiting the member, save the member at the development group specified. Alternatively if you do not make a change or cancel out of the edit SCLM will not have changed the member and so the member will still reside in the production library.

Similarly, if you create a new member, SCLM will create an initial lock on the member name. Once you save the member by exiting the edit session, SCLM will store the member in the development group specified.

6.2.1 Creating a new member in SCLM

From the SCLM Main Menu, enter option **3** to go to the SCLM Utilities Menu. From here, select option **1**, the Library option. When the panel is displayed, the project and development group are filled in from the SCLM Main Menu. The first time you enter this panel, the type will be blank. Here you should specify the type that will contain the member you are about to add. SCLM uses ISPF's *project.group.type* library terminology. Again, your SCLM administrator will have told you the types that you will need to work in. Enter the member name of the new member and then enter an **E** on the command line as shown in Figure 6-2 and press Enter.

Archive

```
------------------------------------------------------------------------------
                      SCLM Library Utility - Entry Panel
Option ===> E

blank Display member list               V View member
    A Browse accounting record          C Build member
    M Browse build map                  P Promote member
    B Browse member                     U Update authorization code
    D Delete member, acct, bmap         T Transfer ownership
    E Edit member

SCLM Library:
  Project  . : SCLM07
  Group  . . . DEV1
  Type . . . . COBOL
  Member . . . STARTAPP    (Blank or pattern for member selection list)

Select and rank member list data  . . TAM  (T=TEXT, A=ACCT, M=BMAP)

Enter "/" to select option
/  Hierarchy view                      Process . . 3  1. Execute
/  Confirm delete                                     2. Submit
/  View processing options for Edit                   3. View options
   Show Member Description
```

Figure 6-2 Specify member to add to SCLM

You are now presented with the SCLM Edit Entry Panel as shown in Figure 6-3 on page 157. You can suppress the display of this panel by deselecting the **View processing options for Edit** option on SCLM Library Utility Entry Panel shown in Figure 6-2.

On the SCLM Edit Entry Panel there are two fields where you can optionally enter information, the Change code and Authorization code fields. A change code is an 8-character identifier used to indicate the reason for an update or modification to a member controlled by SCLM. The change code entered should be externally defined, but can be verified by the user defined change code exits. Change code exits are discussed in Chapter 12, “SCLM user exit processing” on page 241.

An authorization code is an identifier used by SCLM to control authority to update and promote members within a hierarchy. These codes can be used to allow concurrent development without the risk of module collisions (overlaid changes). As we discussed in chapter one, each group in the hierarchy must have at least one authorization code associated with it and if parallel development is needed then additional authorization codes can be defined for a group. The first authorization code defined for a group is its default. An authorization code can be entered on this panel if you want the member to be saved with other than the default authorization code. If your project has multiple development groups and a same named member has already been saved in another development group using the default authorization code then you must enter one of the additional defined authorization codes in order to enter the edit session.

Pressing Enter on the SCLM Edit Entry Panel takes you into the standard ISPF editor where you can code the module you need. You can at this time use any of the normal tools that ISPF provides to copy in code from a data set external to SCLM, or use ISPF **cut** and **paste** to bring code in.

```
------------------------------------------------------------------------------
                          SCLM Edit - Entry Panel
Command ===>

SCLM Library:
   Project . . : SCLM07
   Group . . . : DEV1     . . . TEST     . . . PROD     . . .
   Type  . . . . COBOL
   Member  . . . STARTAPP         (Blank or pattern for member selection list)

Initial Macro  . .
Profile Name . . .                (If blank, defaults to data set type)

Options
/  Confirm Cancel/Move/Replace
   Mixed Mode
   Edit on Workstation
   Preserve VB record length

Change code . . . . . .
Authorization code  . .           (If blank, the default auth code is used)
Parser Volume . . . . .           (If blank, the default volume is used)
```

Figure 6-3 SCLM Edit - Entry Panel

Once you have finished editing the member, you can press PF3 to save it, or type **save** on the command line. At this time SCLM needs to know some additional information about the member. Primarily you need to associate the member with a language. The language is one of the most important aspects of SCLM, as it is the language that dictates how a member will be processed during build (compile) and promote processing. Your SCLM administrator will have set up a number of languages for use in your SCLM project, which invoke, for example, the COBOL compiler. Select the language that is appropriate for the member you are saving as shown in Figure 6-4.

```
_______________________________________________________________________________
                          SCLM Edit Profile
Command ===>

    SCLM Library: SCLM07.DEV1.COBOL
    Member: STARTAPP

Press the Enter key with the language field blank to view a list of
valid languages or enter the desired values and press Enter.

Enter the Cancel command to exit with no change.

Language . . . . COBDT     COBOL II WITH DEBUG TOOL SIDEFILE
Change code  . .           (Use "=" to retrieve last entry)
Description  . .
```

Figure 6-4 Assigning a language to the member

Archived

Pressing Enter when the language is blank displays a list of languages that have been set up by your SCLM administrator. From z/OS version 1.8 onwards, or by applying the appropriate PTF that applies APAR OAxxxxx, a language description is also available to make selection of the appropriate language easier as shown in Figure 6-5.

```
 Select one of the following valid languages for member
  in Project SCLM07

    S   Language Description
    -   -------- ------------------------------------
        ARCHDEF  ARCHITECTURE DEFINITION
        CICSCBE  ENTERPRISE COBOL WITH CICS
        CICSPLE  ENTERPRISE PL/I WITH CICS
        COBDT    COBOL II WITH DEBUG TOOL SIDEFILE
        COBE     ENTERPRISE COBOL
        COB2     COBOL II
        COB3DB2  ENTERPRISE COBOL WITH DB2
        HLAS     OS/VS HIGH LEVEL ASSEMBLER
        PLEDB2   PLI ENTERPRIZE WITH DB2
        PLIDB2X
        PLIE     ENTERPRISE PL/I
        TEXT     UNTRANSLATED TEXT
******************************** Bottom of data ********************************
```

Figure 6-5 List of available languages

Once saved, the SCLM Member List panel is displayed where the member you have just worked on is listed along with some statistical information. The list of available options is displayed on the panel also, so you can work on a number of members directly from this member list. In the panel shown in Figure 6-6 there are three columns shown labeled **`Text`**, **`Account`** and **`Bld Map`**. These tell us certain information about the member. The **`Text`** refers to the location of the physical member source in the SCLM hierarchy. For a new member this will be the development group specified. The **`Account`** refers to the location of the SCLM accounting information or Meta-data. This is additional information SCLM stores about the member for configuration purposes. The final column, **`Bld Map`**, relates to the build map. The build map is some additional SCLM information that is stored that relates generated outputs, such as OBJ and LIST, to the source they were generated from.

```
_____________________________________________________________________________
Member List : SCLM07.DEV1.COBOL - HIERARCHY VIEW -              Member 1 of 1
Command ===>                                                 Scroll ===> CSR

A=Account      M=Map        B=Browse       D=Delete      E=Edit
V=View         C=Build      P=Promote      U=Update      T=Transfer

   Member   Status      Text     Chg Date   Chg Time   Account    Bld Map
   STARTAPP             DEV1     2006/11/15 05:33:15   DEV1
******************************** Bottom of data ********************************
```

Figure 6-6 SCLM Member List

You have now created a new member in SCLM.

6.2.2 Editing an existing member in SCLM

From the SCLM Main Menu enter option **3** to go to the SCLM Utilities Menu. From here select option 1, the Library option. When the panel is displayed the project and development group are filled in from SCLM Main Menu. Enter the type where the member you wish to work on is stored and a member patter if you wish. This is shown in Figure 6-7.

```
------------------------------------------------------------------------------
                       SCLM Library Utility - Entry Panel
Option ===>

blank Display member list               V View member
    A Browse accounting record          C Build member
    M Browse build map                  P Promote member
    B Browse member                     U Update authorization code
    D Delete member, acct, bmap         T Transfer ownership
    E Edit member

SCLM Library:
  Project  . : SCLM07
  Group  . . . DEV1
  Type . . . . COBOL
  Member . . . *APP        (Blank or pattern for member selection list)

Select and rank member list data  . . TAM  (T=TEXT, A=ACCT, M=BMAP)

Enter "/" to select option
/  Hierarchy view                    Process . . 3  1. Execute
/  Confirm delete                                   2. Submit
/  View processing options for Edit                 3. View options
   Show Member Description
```

Figure 6-7 List members to edit

Press Enter and a list of members that met the specified pattern are displayed as shown in Figure 6-8. SCLM looks for members matching the specified criteria starting at the specified development group and looking up the SCLM project hierarchy until a match is found. If you only wish to search for members in the group specified then uncheck the **Hierarchy View** option on the SCLM Library Utility panel shown in Figure 6-7.

In the member list panel shown below, there are three columns labeled **Text**, **Account** and **Bld Map**. These tell us certain information about the member. The **Text** refers to the location of the physical member source in the SCLM hierarchy. In the example shown the source for this program is currently stored in the PROD group in the hierarchy. The **Account** refers to the location of the SCLM accounting information or meta-data. This is additional information SCLM stores about the member for configuration purposes. The final column **Bld Map** relates to the build map. The build map is some additional SCLM information that is stored that relates generated outputs, such as OBJ and LIST, to the source they were generated from. Information on build maps is covered in “Building members in SCLM” on page 167.

Archived

```
------------------------------------------------------------------------
Member List : SCLM07.DEV1.COBOL - HIERARCHY VIEW -              Member 1 of 2
Command ===>                                                 Scroll ===> CSR

A=Account      M=Map        B=Browse       D=Delete      E=Edit
V=View         C=Build      P=Promote      U=Update      T=Transfer

    Member   Status      Text     Chg Date   Chg Time   Account    Bld Map
    PRINTAPP             PROD     2006/11/15 07:07:17   PROD       PROD
    STARTAPP             PROD     2006/11/15 05:33:15   PROD       PROD
****************************** Bottom of data ********************************
```

Figure 6-8 List of members that met specified criteria

To work on a member, enter **E** next to the required member and press Enter. The SCLM Edit Entry Panel as shown in Figure 6-3 on page 157 is displayed, where you can optionally enter a change code and authorization code as we discussed previously. Press Enter again and you are presented with the member for you to begin work on.

Once you have finished editing, press PF3 or type save to save the member. When you press PF3 you are taken back to the member list panel where the information presented has ben updated to show the new location of the member in the hierarchy. As you have saved the member SCLM took a copy and saved it in the development group, plus updated the SCLM accounting information to reflect this. This is shown in Figure 6-9.

```
------------------------------------------------------------------------
Member List : SCLM07.DEV1.COBOL - HIERARCHY VIEW -              Member 1 of 2
Command ===>                                                 Scroll ===> CSR

A=Account      M=Map        B=Browse       D=Delete      E=Edit
V=View         C=Build      P=Promote      U=Update      T=Transfer

    Member   Status      Text     Chg Date   Chg Time   Account    Bld Map
    PRINTAPP *EDITED     DEV1     2006/11/15 07:42:13   DEV1       PROD
    STARTAPP             PROD     2006/11/15 05:33:15   PROD       PROD
****************************** Bottom of data ********************************
```

Figure 6-9 Status updated once a member has been edited

The build map is still shown as being at PROD, as you have not built the member yet to reflect your changes. Information on the build process is covered in “Building members in SCLM” on page 167.

6.2.3 Dependency processing

When you have finished editing and then save a member in SCLM, now SCLM will check through the source and look for include dependency information. As part of the language translators your SCLM Administrator will have set up for you to use, there is a parse step. This parse step uses one of the supplied parsers or one customer written by your SCLM Administrator, normally the supplied parsers provide all the support required. On the save of the member the parse will look in the source code for statements such as COPY if the member is COBOL or %INCLUDE if the member is PL/I. The list of includes is then stored in the account record for the source member.

This list of dependencies is used by SCLM at build time. If you change a copybook and then build an architecture definition that includes all of your applications, SCLM will know all of the source members that include that copybook and will therefore build them. The build process is discussed in more depth in “Building members in SCLM” on page 167.

6.2.4 Additional options in SCLM edit

There are some additional tools provided by SCLM over and above the standard ISPF tools to assist you in creating and editing members. These additional commands are invoked while in edit on the member.

SPROF command

The SPROF command is mostly used to change the language of a member. For example, you may have added some CICS or DB2 calls to a certain member, so the language will need to change to ensure that the required compile processes are called. It is also possible to add a member description that is then associated with the member. When the SPROF command is entered, the SPROF panel shown in Figure 6-4 is displayed. Utilizing member description is discussed in the following sections.

SCREATE command

The SCREATE command is similar to the ISPF **`create`** command. SCLM will create a copy of the member with the new name specified in the development group you are working in. SCLM creates the account information for the member also, so the member will have the same language assigned to it as the member it was created from. To use the SCREATE command, you do not need to specify the CC line commands as you do in ISPF **`create`**.

SMOVE command

The SMOVE command is similar to the ISPF **`move`** command. It will move an existing SCLM controlled member into the current member and then delete the existing member and its accounting information from SCLM.

SREPLACE command

The SREPLACE is similar to the ISPF **`replace`** command. It allows you to replace an existing SCLM member in the current development group. If the member you are replacing does not exist in the development group, but exists higher in the hierarchy, then SCLM will not create a development version of the member.

6.2.5 ISPF edit / compare command

The ISPF edit / **`compare`** primary command can be used within an SCLM edit session to compare the member being edited to another member either in or out of the SCLM hierarchy, highlight the differences between the members, and retrofit any changes or fix work as needed. This command is documented fully in the ISPF manual, *Edit and Edit Macros*, SC34-4820, It is being documented here because it can be very useful to developers and they may not be aware of its existence.

To use the **`compare`** command:

1. Edit the member under SCLM.
2. Enter **`HILITE ON RESET`** on the command line.

   This highlights the member in different colors according to the syntax of its language. This command is only needed if **`HILITE`** is off.

3. Enter a valid COMPARE command on the command line:
   - Enter **COMPARE** with no operands to set options for the COMPARE command. The Edit Compare Settings panel is displayed. This panel enables you to customize the comparison by selecting the relevant SuperC options to use.
   - Enter **COMPARE SESSION** or **COMPARE *** to compare the changes you have made to the edit data since the beginning of the edit session or since the last SAVE command.
   - Enter **COMPARE NEXT** to compare this member to the next same named member higher in the hierarchy.
   - Enter **COMPARE 'pdsname'** to compare against a member that exists in another PDS (in or out of SCLM). Replace **pdsname** with the actual data set name. This command can be very useful when you want to compare your member to a member from a separate leg of the SCLM hierarchy. In this case the **pdsname** would be the fully qualified SCLM data set in the other leg.
   - Enter **COMPARE 'pdsname' X** to show only lines that are different between the two members.
4. Merge the lines from the two compared members as needed:
   - Edit/Compare colors all lines in white that exist in the compare target member but do not exist in the member being edited. These lines can be merged into the member being edited with the **MD** (Make Data) line command. Just type **MD** next to lines you want to merge into your edit session. Use **MDD** to block several lines to merge. Use **MD9999** on the first line to merge all lines in the member.
   - Enter the locate command **L SPECIAL NEXT** to locate the next line for merge consideration. These are the lines colored in white.
5. Locate the lines in your member that are not in the compared to member:
   - All blue lines represent lines in the member being edited that do not exist in the compare target member.
   - Enter the locate command **L LABEL** to locate the next line in blue not found in the compare target member.

6.2.6 Making SCLM edit work the way you want it to

We have looked at the basics of adding new members and listing existing members. There are additional options on the SCLM Library Utility Entry Panel, shown in Figure 6-7 on page 159, that you can use so that edit processing works how you want it to work. Let us look at some of these options.

Select and rank member list data

This is a one, two, or three character string that indicates the kind of information that appears on the member list panel. You can specify strings composed of the following characters:

T To display text data
A To display accounting data
M To display build map data

Each character can only be used once. The order of the characters determines the order of the data on the member list. This option limits the type of data that appears with each member on the list, and only members that have the types of data specified will appear. For example, a member that only has text will not appear if the string AM is specified. All types of data that exist for a member at a particular level are subject to processing by library utility commands.

If only two types of data are specified and one of those is A (accounting), the language associated with the member will also be displayed. If only A is specified, both the language and authorization code will be displayed. Seeing the language at a glance can be a desirable option.

In the previous examples of this panel we have used TAM to display Text, Accounting, and Build Map locations. Figure 6-10 shows the member list when only the A (Accounting) and T (Text) options are used.

```
------------------------------------------------------------------------------
Member List : SCLM07.DEV1.COBOL - HIERARCHY VIEW -              Member 1 of 12
Command ===>                                                  Scroll ===> CSR

A=Account      M=Map        B=Browse       D=Delete      E=Edit
V=View         C=Build      P=Promote      U=Update      T=Transfer

    Member   Status      Account  Language   Text       Chg Date   Chg Time
    BILLINGP            PROD     COBE       PROD       2007/01/13 16:17:33
    CBLDBGEX            PROD     COBE       PROD       2007/01/13 16:25:38
    ENTCOB1             DEV1     COBE       DEV1       2007/01/14 20:08:18
    PETER1              DEV1     COBE       DEV1       2007/01/11 21:49:40
    PETER2              DEV1     CBCID2DT   DEV1       2007/01/03 01:20:39
    PRINTAPP            PROD     COBEDT     PROD       2007/01/13 16:17:19
    RDBKC01             DEV1     CBCID2DT   DEV1       2007/01/28 16:22:28
    RDBKC02             DEV1     COBCICD2   DEV1       2006/11/23 01:38:53
    SCLMTST1            DEV1     CBCID2DT   DEV1       2007/01/11 05:08:30
    SCLMTST2            DEV1     CBCID2DT   DEV1       2007/01/11 05:08:46
    STARTAPP            DEV1     COBEDT     DEV1       2007/01/20 13:30:20
    VCMSCLM             DEV1     COBE       DEV1       2006/12/23 06:01:59
****************************** Bottom of data ********************************
```

Figure 6-10 Member list when AT options used

Hierarchy view

If this option is not selected, then only members from the group you have specified on the panel will be listed. If you are going to be editing members, and they reside further up the hierarchy then you should select this option so that SCLM selects as input the library entered on the panel, as well as all the libraries in its hierarchy view. The hierarchy is searched from the bottom up for the first occurrence of the specified member(s) matching the pattern you have specified.

View processing options for edit

If this option is selected, which it is by default, then the SCLM Edit - Entry Panel, shown in Figure 6-3 on page 157, will be displayed. This confirms the member name you have just selected for processing, and it also gives you some options that you can change prior to going into edit. If, however you never change anything on this Edit entry panel — and in normal operation it is rare that you would need to change anything — then you can suppress the display of this panel by deselecting the **View processing options for Edit** option. This will save you a second or two when editing members.

Show member description

It is also possible to assign a member description to the member when you first save the member or subsequently via the SPROF command discussed above. The member description is stored in the members accounting record and will be promoted with the member as it goes up through the hierarchy. Figure 6-11 shows the SPROF panel that is displayed when you first save the member, or enter the SPROF command. We have added a description that gives other developers some idea what the program does.

```
------------------------------------------------------------------------
                        SCLM Edit Profile
Command ===>

    SCLM Library: SCLM07.DEV1.COBOL
    Member: RDBKC01

Press the Enter key with the language field blank to view a list of
valid languages or enter the desired values and press Enter.

Enter the Cancel command to exit with no change.

Language . . . . CBCID2DT  ENTERPRISE COBOL WITH CICS AND DB2
Change code  . .           (Use "=" to retrieve last entry)
Description  . . Display book number and description
```

Figure 6-11 Adding a member description to a member

To see this member description when you are listing members in a certain type, there is an option on the SCLM Library Utility Entry Panel shown in Figure 6-7 on page 159, which is the **Show Member Description** option. By entering a / next to this option, when the member list is displayed, the member descriptions will also be shown. This can be seen in Figure 6-12.

```
------------------------------------------------------------------------------
Member List : SCLM07.DEV1.COBOL - HIERARCHY VIEW -              Member 1 of 2
Command ===>                                                 Scroll ===> CSR

A=Account      M=Map        B=Browse       D=Delete      E=Edit
V=View         C=Build      P=Promote      U=Update      T=Transfer

    Member   Status      Account  Language   Text       Chg Date   Chg Time

    RDBKC01              DEV1     CBCID2DT   DEV1       2007/01/28 16:22:28
    Display book number and description
    RDBKC02              DEV1     COBCICD2   DEV1       2006/11/23 01:38:53
    Program to list available manuals
****************************** Bottom of data ********************************
```

Figure 6-12 Member list showing member description

Archived

Process — execute, submit, or view options

This option relates to the Build and Promote options if they are selected for the member list. If you always run builds and promotes in batch mode and never change the mode from unconditional, then you can bypass the display of the build and promote options panels by setting process option to **2**. Similarly, set the process option to **1** if you prefer to run your builds and promotes in foreground mode.

Updating authorization codes

When the list of members is displayed in the Library Utility panel (see Figure 6-12 on page 164 for an example) one of the actions that can be performed on a member is to update its authorization code using the **U** action. This action is only valid if multiple authorization codes are defined for a development group. There are two reasons why you might want to update a member's authorization code.

The first use of changing a authorization code is if a member has been edited using the default promotable authorization code and another version of the member needs to be edited in another development group using the default authorization code. Most often this is because a fix needs to be made to the program and it needs to be promoted before the first version of the program is ready to be promoted. Then the authorization code of the first member needs to be changed to an alternate non-promotable authorization code before the second member can be edited using the default authorization code.

You may remember that, for a member to be promotable, its authorization code must match an authorization code assigned to the to group and that only one member in a set of parallel development groups promoting to a to group can be edited with the default promotable authorization code. By promotable we mean the authorization code that matches that of the to group.

If you do not update the authorization code, the second user will get the following error when they try to edit the member in their development group using the default promotable authorization code as shown in Figure 6-13.

```
FLM05010 - MEMBER LOCKED AT ANOTHER GROUP
 INPUT GROUP: DEV2  TYPE: SOURCE  MEMBER: PETER1
 ERROR GROUP: DEV1  AUTHORIZATION CODE: P
```

Figure 6-13 Member locked error message

The second use of changing a authorization code is if the member has been edited using a alternate authorization code and the alternate authorization code is non-promotable. You can then change the member's authorization code to the default authorization code so that it will be promotable.

In Figure 6-14 is the message you will get if you try to promote the member without updating its authorization code to the promotable one.

```
FLM05001 - EXISTING MEMBER'S AUTHORIZATION CODE IS NOT DEFINED TO THE GROUP
           GROUP: TEST       TYPE: COBOL      MEMBER: PETER1
           ERROR GROUP: TEST       AUTHORIZATION CODE: D
```

Figure 6-14 Authorization code promote error

This means that the authorization code, D, is not defined for the to group, TEST. SCLM will stop the promote. The Update Authorization code function of the library utility will need to be used to update the authorization code to match the authorization code of the to group. In our example that would be from D to P, which is defined for the TEST group.

An error could occur when you try to update the authorization code. You might get a predecessor verification error as shown in Figure 6-15. Basically, this error means that the member in the TEST group has changes that need to be merged into the version that you are trying to change the authorization code for. SCLM keeps track of change dates and times so that it knows if a member has been changed elsewhere in the hierarchy. It does this so that if a member is moved ahead of a same named member in the hierarchy it can inform the user of the same named member to merge the code changes between the two versions of the members.

In the example, two developers were working on the same program, one in DEV1 using the D authorization code and one in DEV2 using the promotable, P, authorization code. They made independent changes to the program and the DEV2 version was promoted to the TEST group. Now, when the user attempts to update the DEV1 program's authorization code to P, they get the predecessor error.

```
FLM05002 - PREDECESSOR VERIFICATION FAILED
 INPUT GROUP : DEV1  TYPE: COBOL  MEMBER: PETER1
 ERROR GROUP1: DEV1  DATE: 2006/12/14  TIME: 03:42:06
 ERROR GROUP2: TEST  DATE: 2007/01/11  TIME: 20:52:56
```

Figure 6-15 Predecessor Verification error when attempting to change a authorization code

To fix this problem, the user needs to:

1. Use the Edit Compare function to merge the changes from the TEST version into the DEV1 version.
2. Perform a force migration on the member to synchronize the accounting records. Use the SCLM Migration Utility to perform this function. This utility is the third option on the SCLM utilities panel. Do not use a "wildcard" for the member name. No language needs to be specified. Select Mode 3 to force the migration.

Figure 6-16 shows the messages you will receive when you perform the forced migration.

```
FLM32501 - INVOKING MIGRATION UTILITY
FLM32101 - MIGRATION UTILITY INITIATED - 14:21:41 ON 2001/11/07
FLM32304 - WARNING, A NEW ACCOUNTING RECORD WILL BE GENERATED FOR
           MEMBER: PETER1  GROUP: DEV1     TYPE: COBOL
           BASED ON THE ACCOUNTING RECORD AT GROUP: TEST.
           CHANGES MAY NEED TO BE MERGED BEFORE PROMOTING THE MEMBER.
FLM06501 - TRANSLATOR RETURN CODE FROM ===> SCLM COBOL PARSE     ===> 0
FLM87107 - SAVE WARNINGS FOR MEMBER PETER1 AT 14:21:43, CODE: 4
FLM32401 - MIGRATION UTILITY COMPLETED - 14:21:43 ON 2001/11/07
FLM09008 - RETURN CODE = 4
```

Figure 6-16 Output of migration utility for a forced migration

7

Building members in SCLM

In this chapter we describe the process of building members in SCLM. The function, SCLM Build, is used to apply a set of processes to a source member. The most common example of a build process is that of a compiler. In this sense the build is the compile. The source member that is being built is the input to the compile process. The compiler is called, and outputs such as object code and compile listings are created. But the build can be any process that needs to be performed against a source member. For example, a JCL check procedure could be defined as the build processor. So that when the JCL is built, the JCL checker is called. If there are any problems with the JCL, the build fails.

The build can be done for as small a unit as required, such as an individual module, or for a whole application, many modules generating multiple load modules. By using architecture defininitions, whole applications can be defined that relate to each other.

SCLM Build is also an integral part of the configuration management as it performs an intelligent build. What this means is that SCLM will only build what needs to be built, you do not have to tell SCLM specifically what to build. For example, if a copybook has changed, and a Build is performed with an architecture definition (ARCHDEF), sometimes referred to as a “package” in other products, that points to 5 modules, then only those modules that reference the changed copybook will be selected for compiling and linking.

The only member that will need to be edited in the development group will be the copybook. The only other members that will exist in the development group will be the output objects and load modules related to the changed copybook. This saves the developer time by not requiring the retrieval, pull-down, or processing of any kind for the source code using the changed copybook.

SCLM knows the latest, most current status of inputs. It knows the most efficient way to construct the outputs and will pull the correct level of unchanged members from the hierarchy.

© Copyright IBM Corp. 2007. All rights reserved.

7.1 Building an individual source member

In order to generate a load module from one or multiple pieces of source, you would normally build an architecture definition. However, when you are developing your code and working from a list of source members, it may be more convenient to compile the source directly. You would do this to compile the source to work out any compilation errors.

Once you have finished editing your source member and you are in the member list (Option 3.1), enter a **C** next to the member you wish to build. This will bring up the build options panel as shown in Figure 7-1.

```
------------------------------------------------------------------------------
                          SCLM Build - Entry Panel
Command ===>

Build input:
  Project  . : SCLM07
  Group  . . . DEV1
  Type . . . . COBOL                    Enter "/" to select option
  Member . . . STARTAPP                 /  Error Listings only
                                           Workstation Build

  Mode  . . 1  1. Conditional           Scope . . . 2  1. Limited
               2. Unconditional                        2. Normal
               3. Forced                               3. Subunit
               4. Report                               4. Extended
Output control:
               Ex Sub                   Process . . 2  1. Execute
  Messages . . 1  2    1. Terminal                     2. Submit
  Report . . . 3  2    2. Printer
  Listings . . 3  2    3. Data set      Printer . . H
                       4. None          Volume  . .
```

Figure 7-1 SCLM Build Entry Panel

The build process can be run foreground or background, although your installation may have disabled the foreground capabilities due to site standards. For a build, you as a developer will not have to specify any compiler options. These will all have been defined by your SCLM Administrator in the language translators. There is a possibility for overriding compiler option through what is called a CC (compilation control) architecture definition, but for most installations these are not required.

The build options should not need to be modified; however, you might want to change where the output is sent. There are three types of output from the build:

- Build messages:

  These messages give you a status as SCLM processes through the build. It will contain messages indicating the time the build starts and finishes. Additionally, each build step that is in the language definition will be listed in the messages with a return code from that step. Your SCLM Administrator might have given each of these steps a meaningful name to help you identify what processing is being performed.

Archived

- Build report:

  The Build report contains a list of the outputs that were generated as part of the build. The report also lists as build map entries updated. Additionally the build report will give the reason for rebuild. This can be very important if you start a build of an architecture definition and SCLM starts building something you did not expect. If you check the reason for rebuild, there will probably be a copybook or include listed that someone else has changed, but your program uses.

- Build listing:

  The build listing is the output from the compiler or build process you are running. Default behavior is *not* to create the build listing file if there are no errors. However, this does not mean that the listing is not created and stored in SCLM. For example, if there is a compile error and you run the build in foreground, SCLM will automatically place you in browse on the listing detailing the errors. If you would like to see the compile output directly after the build, you can by removing the `/` on **`Error Listings only`** on the build entry panel shown previously.

Additionally, the mode of build is very important. The build defaults to conditional and as such, for normal operation this is what you should use. However, you need to be aware of all the build modes and what implications they have:

- Conditional:

  The build will stop once the first error is encountered. This is the default behavior.

- Unconditional:

  This option will work the same way as conditional if a single source is being built. However, if a high level ARCHDEF is being built that is going to involve the compilation of multiple members then unconditional continues to build all the members even when errors are found.

  This option is useful when migrating many modules into SCLM and you know that most are going to compile OK. Rather than stopping after each and every error, you can see what all the compile errors are in one go. For example, many modules may have a common copybook missing. With SCLM you just need to invoke the build of the ARCHDEF again and SCLM will only do what has to be retried. Any previously successfully built modules will not be attempted again.

- Forced:

  Be careful using forced build. SCLM will rebuild everything in the scope of the member selected for build. This is not a problem if a source member is force built as the behavior will be the same as for conditional or unconditional. However, if a high level ARCHDEF is selected to be force built, every single piece of source in the scope of that high level ARCHDEF will be recompiled. There may be times when a force build is necessary, however. If, for example, there are some external includes (external to the SCLM project) that have changed that you need to pick up, then a forced build could be what you need.

  Note: If there is an external product that has includes that all your programs need to pick up, then you can approach this situation in one of two ways.

  - Force build a high level ARCHDEF that encompasses the scope of all the modules you need to rebuild
  - Get your SCLM administrator to change the version on the FLMLANGL for all language definitions that include the common includes. Then conditionally build a high level ARCHDEF that encompasses the scope of all the modules you need to rebuild. Any modules with a language that relates to the language definition that has had its version changed will be rebuilt.

Archived

- Report:

  Report mode build will run through SCLM's build verification to work out what is going to be built based on the scope of the member selected for build. You can see if you are going to rebuild the whole project because someone has changed a common copybook used by every module in the project. This is a useful reporting tool if you are unsure of the implications of a change you have made to a copybook or common subroutine.

If you are running build foreground, it is preferable to have the messages return to the terminal while sending the report and listing to data set. When running in batch you can chose to have the output go to data sets or be returned in the job output.

Pressing Enter will invoke the build. If running the build in foreground you can see the progress via the build messages as shown in Figure 7-2. If running in batch, this is also produced in the BLDMSGS DD.

```
FLM49000 - INVOKING BUILD PROCESSOR
FLM09002 - THE BUILD REPORT WILL APPEAR IN DOHERTL.BUILD.REPORT23
FLM09006 - THE BUILD LISTING WILL APPEAR IN DOHERTL.BUILD.LIST23
FLM42000 - BUILD PROCESSOR INITIATED - 03:38:57 ON 2006/11/18
FLM44500 - >> INVOKING BUILD TRANSLATOR(S) FOR TYPE: COBOL    MEMBER: STARTAPP
FLM06501 - TRANSLATOR RETURN CODE FROM ===> ALLOC SYSDEBUG       ===> 0
FLM06501 - TRANSLATOR RETURN CODE FROM ===> COBOL II COMPILE     ===> 0
FLM06501 - TRANSLATOR RETURN CODE FROM ===> COPY SYSDEBUG        ===> 0
FLM46000 - BUILD PROCESSOR COMPLETED - 03:38:59 ON 2006/11/18
FLM09008 - RETURN CODE = 0
```

Figure 7-2 Build messages

The Build report, whether you have directed it to a data set, to the terminal or to the batch job output, will contain information on what was created and why the build occurred. This can be seen in Figure 7-3. This build of this COBOL program created an OBJ, a listing, and a Debug Tool sidefile. The reason the rebuild occurred was because a change was made to the STARTCPY member in the COPYBOOK library.

```
********************************************************************************
    ******* B U I L D   O U T P U T S   G E N E R A T E D *******  Page 1

MEMBER     TYPE       VERSION      KEYWORD
------     ----       -------      -------
STARTAPP   OBJ              5      OBJ
STARTAPP   LIST             5      LIST
STARTAPP   SYSDEBUG         5      OUTX
          ******* B U I L D   M A P S   G E N E R A T E D *******  Page 2

                                  (REASON FOR REBUILD)
MEMBER     TYPE       VERSION      MEMBER     TYPE
------     ----       -------      -------    ----
STARTAPP   COBOL            5      STARTCPY   COPYBOOK
```

Figure 7-3 Build report

Build invocation from within edit

There is an alternative method to building the member without leaving the member. If you are working out your compile errors, then this would be the most efficient method to use. While in edit on the member, you see on the menu bar a Build option as shown in Figure 7-4. If you select that, a pull-down menu gives you some choices. Selecting the first to Save and Build will save the current member and run SCLM's build, however when you return, you are still in edit on the member.

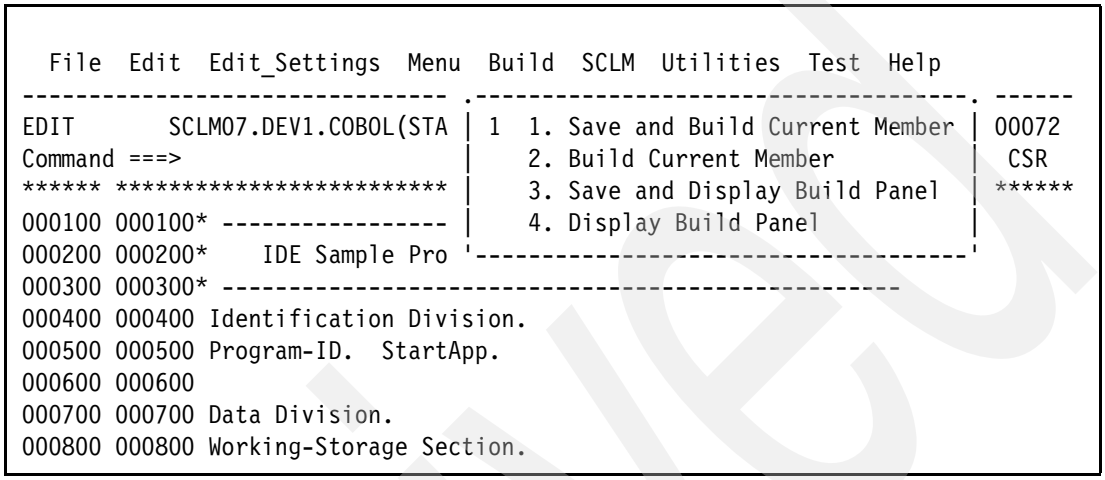


Figure 7-4 Build invocation from within edit

Note: Be careful when building from within edit, or directly building a source member. The build may have unintended consequences, as it will run any build exits that are defined to the project — something that you might not want to occur.

For example, if you have a user exit that does a bind when a new DBRMLIB is created, a build of a COBOL source member that contains DB2 statements will create the DBRM, and the user exit will then do a bind. But as you are not building a large enough scope, in this case the linkedit control ARCHDEF, the linkedit is not performed, so the existing load module cannot be run, because it will get a DB2 -805 or -818 error.

7.2 Creating and building an application architecture definition

When you are developing applications, just building the source may not be sufficient to create the object you need to test and subsequently move up the hierarchy. For example, with COBOL, you need to create a load module. SCLM handles the creation of load modules via what are called architecture definition members, and in particular a link-edit control or LEC architecture definition. Once you have worked out any compile errors, the normal phase of making program changes is to edit the source and then build the ARCHDEF. The ARCHDEF or the above example is quite simple and contains certain keywords to tell SCLM where to find the source and what to create as far as a load module is concerned. This can be seen in Figure 7-5.

```
*
* Include statement for STARTAPP application
*
 INCLD STARTAPP COBOL    * Include COBOL member STARTAPP
 INCLD PRINTAPP COBOL    * Include COBOL member PRINTAPP
*
* LOAD statements
*
 LOAD  STARTAPP LOAD     * Load module name STARTAPP
 LMAP  STARTAPP LMAP     * Linkedit listing stored in type LMAP
```

Figure 7-5 LEC architecture definition

This architecture definition will build your load module using the default options in the linkage editor translator your SCLM administrator has set up. But any standard linkage editor options can be incorporated into a LEC ARCHDEF by using SCLM's supplied parameters. For example, a slightly more complicated LEC ARCHDEF can be seen in Figure 7-6.

```
*
* Include statement for BWBBLD executable module
*
  INCLD BWBBLD   SOURCE   * Include source member BWBBLD
  INCLD BWBCOPYR SOURCE   * Include copyright source member BWBCOPYR
*
* Linkedit command statements
*
  CMD   ORDER    BWBCOPYR * Place BWBCOPYR at front of load module
  CMD   ENTRY    BWBBLD   * Set entry point to BWBBLD
*
* LOAD statements
*
  LOAD  BWBBLD   LOAD     * Load module name BWBBLD
  LMAP  BWBBLD   LMAP     * Linkedit listing stored in type LMAP
```

Figure 7-6 LEC ARCHDEF incorporating standard linkage editor statements

In the above example, the load module consists of two modules. The copyright module BWBCOPYR must come first, however, the entry point must be BWBBLD. Additionally, linkage options such as AMODE and RMODE can be overridden. This is shown in Figure 7-7.

Archive

```
*
* Use LE370X language translator rather than default LE370
*
 LKED  LE370X
*
* Include statement for JCOBPGM executable module
*
 INCLD JCOBPGM2 COBOL              * Include COBOL member JCOBPGM2
 INCLD JCOBPGM  COBOL               * Include COBOL member JCOBPGM
*
* External Includes for JCOBPGM executable module
*
 CMD   INCLUDE SYSLIB(DFHELII)      * External Include for CICS stub DFHELII
 CMD   INCLUDE SYSLIB(EZACICAL)     * External Include for EZACICAL
*
* LOAD statements
*
 LOAD  JCOBPGM  LOAD                * Load module name JCOBPGM
 LMAP  JCOBPGM  LMAP                * Linkedit listing stored in type LMAP
*
* Linkedit statements to override language translator defaults
*
 PARM  MAP,RENT,AMODE=31,RMODE=ANY
```

Figure 7-7 LEC ARCHDEF incorporating linkage editor options override statements

In the foregoing example there are two modules not under SCLM control that are being included, DFHELII and EZACICAL. These two modules will be included from the SYSLIB concatenation that is defined in the language translator. Two modules are included from the SCLM hierarchy, JCOBPGM2 and JCOBPGM. The standard linkage editor statements are overridden with the PARM statement. Finally the LKED statement is used to tell SCLM to use a different language translator, in this case LE370X, instead of the standard default language translator for link-editing, LE370. Your SCLM Administrator will have set this up.

When an architecture member is built the process is the same as though you are building a source member. SCLM uses its dependency checking along with the ARCHDEF to decide what needs to be built. For example, if we take the previous example of just changing a copybook and then building the ARCHDEF we see, through the SCLM messages shown in Figure 7-8, that SCLM knows to recompile the program containing the copybook and then relink the load module containing the program.

Archived

```
FLM49000 - INVOKING BUILD PROCESSOR
FLM09002 - THE BUILD REPORT WILL APPEAR IN DOHERTL.BUILD.REPORT30
FLM09006 - THE BUILD LISTING WILL APPEAR IN DOHERTL.BUILD.LIST30
FLM42000 - BUILD PROCESSOR INITIATED - 06:07:56 ON 2006/11/18
FLM44500 - >> INVOKING BUILD TRANSLATOR(S) FOR TYPE: COBOL    MEMBER: STARTAPP
FLM06501 - TRANSLATOR RETURN CODE FROM ===> ALLOC SYSDEBUG       ===> 0
FLM06501 - TRANSLATOR RETURN CODE FROM ===> COBOL II COMPILE     ===> 0
FLM06501 - TRANSLATOR RETURN CODE FROM ===> COPY SYSDEBUG        ===> 0
FLM44500 - >> INVOKING BUILD TRANSLATOR(S) FOR TYPE: ARCHLEC MEMBER: STARTAPP
FLM06501 - TRANSLATOR RETURN CODE FROM ===> LKED/370             ===> 0
FLM46000 - BUILD PROCESSOR COMPLETED - 06:07:59 ON 2006/11/18
FLM09008 - RETURN CODE = 0
```

Figure 7-8 Build messages from ARCHDEF build

Once again, the build report will tell you what has been generated as part of this build and what was the reason for rebuild. This is shown in Figure 7-9, where you see that this time as well as the OBJ, listing, and debug sidefile, that the load and the LMAP are also created. The reason for rebuild shows us that the STARTAPP COBOL program was rebuilt because the STARTCPY COPYBOOK was changed. Then the STARTAPP ARCHLEC was built because the STARTAPP COBOL member was rebuilt.

```
********************************************************************************
    ******* B U I L D   O U T P U T S   G E N E R A T E D *******  Page 1

MEMBER     TYPE       VERSION      KEYWORD
------     ----       -------      -------
STARTAPP   OBJ              7      OBJ
STARTAPP   LIST             7      LIST
STARTAPP   SYSDEBUG         7      OUTX
STARTAPP   LOAD             3      LOAD
STARTAPP   LMAP             3      LMAP
         ******* B U I L D   M A P S   G E N E R A T E D *******  Page 2

                                 (REASON FOR REBUILD)
MEMBER     TYPE       VERSION      MEMBER     TYPE
------     ----       -------      -------    ----
STARTAPP   ARCHLEC          3      STARTAPP   COBOL
STARTAPP   COBOL            7      STARTCPY   COPYBOOK
    ******* B U I L D   O U T P U T S     D E L E T E D   *******  Page 3
```

Figure 7-9 Build report from ARCHDEF build

For more information on architecture definition members, see Chapter 5, “SCLM architecture definition considerations” on page 135.

7.3 Creating and building high level (HL) architecture definitions

The next level up from building LEC ARCHDEFs is building what, in SCLM, are called high level architecture definitions. These are ARCHDEFs that group together logically related applications and sub-applications. High level ARCHDEFs can include other high level ARCHDEFs and so on down to the lowest level ARCHDEF of an application, which may be the LEC ARCHDEF that defines how to build the load module.

To use an example from ISPF/SCLM development, the highest level ARCHDEF is called PRODUCT, which includes two other high level ARCHDEFS: ISPFPDF and NLS. ISPFPDF includes three other high level ARCHDEFs: ISPF, PDF, and SCLM. Each of these high level ARCHDEFs include other high level ARCHDEFs that further define the individual components of ISPF. Examples of these high level ARCHDEFs are shown in Figure 7-10 and Figure 7-11.

```
INCL    ISPF      ARCHDEF  * Include ISFP ARCHDEF member
INCL    PDF       ARCHDEF  * Include PDF ARCHDEF member
INCL    SCLM      ARCHDEF  * Include SCLM ARCHDEF member
```

Figure 7-10 ISPFPDF architecture definition

```
*
INCL FLMCNTGN ARCHDEF   * CONSTANT NAME VALUE TABLE GENERATOR
*
INCL SCLMBASE ARCHDEF   * SCLM CODE
INCL FLMNENU  ARCHDEF   * SCLM NLS - ENGLISH
```

Figure 7-11 SCLM architecture definition

Again, building these high level architecture definitions is no different than building LEC ARCHDEFs or source. All you are doing is increasing the scope that SCLM will check for what it needs to build. Of course you should realize that the greater the scope, the longer SCLM will take to start performing the builds. SCLM is intelligent enough to know that even if you have only changed one module, and you make the scope the highest possible ARCHDEF, then that is the only module that will need to be recompiled and relinked.

Here, we take our application used previously and create a high level ARCHDEF called ALLAPPL that pulls together the STARTAPP program and the BILLING program as shown in Figure 7-12. High level ARCHDEFs are identified by the fact that they use the INCL statement to include other high level ARCHDEFs.

```
 INCL  STARTAPP ARCHDEF   * Include STARTAPP ARCHDEF member
 INCL  BILLING  ARCHDEF   * Include BILLING ARCHDEF member
```

Figure 7-12 High level ARCHDEF for the STARTAPP application

Modifying the same copybook again, which is included in both of these applications, we see the result of the build of this high level ARCHDEF in Figure 7-13.

Archive

```
FLM49000 - INVOKING BUILD PROCESSOR
FLM09002 - THE BUILD REPORT WILL APPEAR IN DOHERTL.BUILD.REPORT32
FLM09006 - THE BUILD LISTING WILL APPEAR IN DOHERTL.BUILD.LIST32
FLM42000 - BUILD PROCESSOR INITIATED - 07:28:59 ON 2006/11/18
FLM44500 - >> INVOKING BUILD TRANSLATOR(S) FOR TYPE: COBOL   MEMBER: STARTAPP
FLM06501 - TRANSLATOR RETURN CODE FROM ===> ALLOC SYSDEBUG      ===> 0
FLM06501 - TRANSLATOR RETURN CODE FROM ===> COBOL II COMPILE    ===> 0
FLM06501 - TRANSLATOR RETURN CODE FROM ===> COPY SYSDEBUG       ===> 0
FLM44500 - >> INVOKING BUILD TRANSLATOR(S) FOR TYPE: COBOL   MEMBER: BILLINGP
FLM06501 - TRANSLATOR RETURN CODE FROM ===> ALLOC SYSDEBUG      ===> 0
FLM06501 - TRANSLATOR RETURN CODE FROM ===> COBOL II COMPILE    ===> 0
FLM06501 - TRANSLATOR RETURN CODE FROM ===> COPY SYSDEBUG       ===> 0
FLM44500 - >> INVOKING BUILD TRANSLATOR(S) FOR TYPE: ARCHLEC MEMBER: STARTAPP
FLM06501 - TRANSLATOR RETURN CODE FROM ===> LKED/370           ===> 0
FLM44500 - >> INVOKING BUILD TRANSLATOR(S) FOR TYPE: ARCHLEC MEMBER: BILLING
FLM06501 - TRANSLATOR RETURN CODE FROM ===> LKED/370           ===> 0
FLM46000 - BUILD PROCESSOR COMPLETED - 07:29:04 ON 2006/11/18
FLM09008 - RETURN CODE = 0
```

Figure 7-13 Build messages from high level ARCHDEF build

SCLM knows that COPYBOOK STARTCPY is included in both the STARTAPP COBOL program and the BILLINGP COBOL program. By increasing the scope to a high level ARCHDEF, we let SCLM work out what needs to be rebuilt based on the changes made. So in this example, SCLM compiles both source members that include the copybook and link-edits both load modules. The build report is now more extensive as well, as shown in Figure 7-14.

For more information on architecture definition members, see Chapter 5, “SCLM architecture definition considerations” on page 135.

Archive

```
*************************************************************************
    ******* B U I L D   O U T P U T S   G E N E R A T E D *******  Page 1

MEMBER     TYPE       VERSION      KEYWORD
------     ----       -------      -------
BILLINGP   OBJ              2      OBJ
STARTAPP   OBJ              8
BILLINGP   LIST             2      LIST
STARTAPP   LIST             8
BILLINGP   SYSDEBUG         2      OUTX
STARTAPP   SYSDEBUG         8
BILLING    LOAD             2      LOAD
STARTAPP   LOAD             4
BILLING    LMAP             2      LMAP
STARTAPP   LMAP             4
          ******* B U I L D   M A P S   G E N E R A T E D *******  Page 2

                                  (REASON FOR REBUILD)
MEMBER     TYPE       VERSION      MEMBER     TYPE
------     ----       -------      -------    ----
ALLAPPL    ARCHDEF          2      STARTAPP   ARCHLEC
                                   BILLING    ARCHLEC
BILLING    ARCHLEC          2      BILLINGP   COBOL
STARTAPP   ARCHLEC          4      STARTAPP   COBOL
BILLINGP   COBOL            2      STARTCPY   COPYBOOK
STARTAPP   COBOL            8      STARTCPY   COPYBOOK
```

Figure 7-14 Build report from high level ARCHDEF build

As well as the list of outputs generated, we see the reason for rebuild. Both BILLINGP and STARTAPP COBOL members were recompiled because of the change to the STARTCPY COPYBOOK. This in turn caused the BILLING and STARTAPP ARCHLECs to be rebuilt in order to generate the load modules. Finally, this caused the ALLAPPL build map to be updated as both the BILLING and STARTAPP ARCHLECs were rebuilt.

7.4 Running out of space errors (B37 / E37)

You can run out of space in two ways during SCLM edit, build, or promote operations.

1. The SCLM data set can be full, out of directory space (E37), or the volume can be full.

   Possible solutions to this problem are:

   a. Compress the affected PDS if the directory is out of space. This will not help if the directory blocks are all used up.

   b. Ensure that the volume is not full. If it is, your storage administrator will need to move the data set to a volume with free space.

   c. Check to see if the directory blocks or extents are all used. If they are, then the SCLM data set needs to be allocated as a bigger data set. This can be done in three steps:

      i. Rename the current SCLM data set.

      ii. Re-allocate the data set with more space and / or directory blocks.

iii. Copy the members over from the old renamed data set.

Note: It is safest to use an IEBCOPY job to perform this copy. If you use the ISPF 3.3 Copy, you must change the SCLM Setting from its default value Non-SCLM(2) to SCLM(1). This will ensure that the SCLM control bit is maintained after the copy. Failure to maintain this bit will cause accounting errors for all members copied.

2. A temporary data set created during a build can be allocated too small and the build can abend with a B37 error when the temporary data set fills up. Perform the following steps to fix this problem:

   a. Find the language of the member that the build is failing on. The BLDMSGS will tell you which member was being built when the build failed. You can then use the Library Utility to find the language of the member.

   b. Next check BLDMSGS to determine which step (precompile step, compile step, and so on) of the language definition that the build is failing on. The FLM06501 messages will indicate the last good step completed. For example, in Figure 7-15, the DB2 PREPROCESS step completed successfully and the abend occurred in the “COBOL FOR MVS” step.

```
FLM49000 - INVOKING BUILD PROCESSOR
FLM09006 - THE BUILD LISTING WILL APPEAR IN PCECK.BUILD.LIST02
FLM42000 - BUILD PROCESSOR INITIATED - 12:36:39 ON 2007/01/02
FLM44500 - >> INVOKING BUILD TRANSLATOR(S) FOR TYPE: SOURCE   MEMBER: PETER1
FLM06501 - TRANSLATOR RETURN CODE FROM ===> DB2 PREPROCESS       ===> 4
IEC030I B37-04,IFG0554A,PCECK,SERPROC,SYSPRINT,VIO ,     ,SYS07002.T123641.RA0
00.PCECK.SYSPRINT.H01
FLM06501 - TRANSLATOR RETURN CODE FROM ===> COBOL FOR MVS        ===> 11759616
FLM06503 - PROBABLE SYSTEM/USER ABEND FOR TRANSLATOR: COBOL FOR MVS
           HEXADECIMAL VALUE OF RETURN CODE: 00B37000
FLM44513 - TRANSLATOR ERROR FOR MEMBER: PETER1 TYPE: SOURCE
```

Figure 7-15 BLDMSGS showing a B37 space problem from an SCLM Build

   c. Next look in the language definition (in projid.PROJDEFS.SOURCE) for the failing step and in that step look for the FLMALLOC of the DDNAME that was associated with the B37 error. In the example above (Figure 7-15) the DDNAME is SYSPRINT. Note that this example is from a foreground build. For a submitted build the B37 error message would be in the JES messages. Here is the FLMALLOC for the failing SYSPRINT allocation for this language:

```
FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=FBA,LRECL=133,          C
 RECNUM=1000,PRINT=Y,DFLTTYP=SRCLIST,DDNAME=SYSPRINT
```

   If the DDNAME is similar to SLxxxx and appears to be system-generated by SCLM, then check the temporary files under FLMALLOC where the IOTYPE is equal to W. For example, here is a FLMALLOC for SYSUT1 in the language above:

```
FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=100000,DDNAME=SYSUT1
```

   d. Finally increase the RECNUM (for the example above from 1000 to 10000), reassemble and link the project definition and try your build again.

7.5 Building and promoting by change code

You can use architecture definitions to identify and only build and promote the parts associated with a specific change or group of changes. This is discussed in some detail in the Architecture Definition chapter of the *SCLM Guide and Reference*. Following we provide a short summary of how this feature is used, along with some warnings on its usage.

7.5.1 Summary of the building and promoting feature

The Build and Promote by change code function is initiated normally from a high level ARCHDEF that has a CCODE statement added to it. Such an ARCHDEF contains a list of CCODE keywords followed by a change code and include flag. Here is an example of such an architecture definition:

```
* ARCHDEF FOR PACKAGE PKG00001
CCODE CC000001 INCLUDE * Include changes for problem CC000001
INCL SCLMTST1 ARCHDEF * SCLM ARCHDEF
INCL SCLMTST2 ARCHDEF * SCLM ARCHDEF
```

In this example, SCLMTST1 has been edited with change code CC000001 and SCLMTST2 has been edited with change code CC000002, both down to the same development group. Both SCLMTST1 and SCLMTST2 are being called from the same job stream that the developer is testing with and the developer is using the CCODE statement in their package to ensure that the job is only running load modules that have changes for CC000001 so that they can correctly test the CC000001 changes. Let us also assume for this example that SCLMTST2 has been built already including CC000001 changes.

When the package is built, SCLMTST1 will be built, since it was changed with CC000001, and any outputs from a build of SCLMTST2 *will be deleted*, since SCLMTST2 was changed with CC000001. Therefore the job can be tested correctly. The development load module for the build of SCLMTST1 and the production load module for SCLMTST2 will be used in the testing. Note that build by change code both builds AND deletes outputs. It does what is necessary to ensure the outputs within the scope of the package only have the requested change codes for CCODE INCLUDE statements and does not have the designated change code for CCODE EXCLUDES statements.

Once the change has been tested, the package can also be used to promote the changes up the hierarchy. Only the changes associated with CC000001 will be promoted.

Valid values for the CCODE include flag are INCLUDE or EXCLUDE. The default value is INCLUDE. A value of INCLUDE indicates that *only the changes specified are included.* A value of EXCLUDE indicates that *everything except the specified changes are included.*

Here is an example of the CCODE with EXCLUDE being used.

```
* ARCHDEF FOR PACKAGE PKG00002
CCODE CC000002 EXCLUDE * exclude changes for problem CC000002
INCL SCLMTST1 ARCHDEF * SCLM ARCHDEF
```

In this example, a copybook included by SCLMTST1 was being changed by change code CC000002 and the developer did not want this change being included into the changes that they were making to SCLMTST1, nor did they want the copybook promoted when this program is promoted. When this package is built, the development version of the copybook will not be used (since it was edited with CC000002) but rather the version of the copybook that is higher in the hierarchy will be used. The SCLMTST1 load module can now be tested without the CC000002 copybook change in it.

When the change has been tested, the package with the CCODE statement can be used to promote the code while ensuring that the copybook that was changed with CC000002 has been left behind.

7.5.2 Building and promoting by change code warnings

Build and Promote by Change Code will work fine as long as no other developer is working on members that are affected by another developer's builds. *This is because all builds whether normal or build by change code for a development group will store their output in a common dataset*, and this can have unintended consequences as we describe next.

Suppose a second developer was working on the changes for CC000002 in the above example and they had just built SCLMTST1 and SCLMTST2 including CC000002 changes and had began testing them in the development load library and then the first developer had built the packages above that only included CC000001 and thus had eliminated all of the output of the members that they had changed with CC000002 from the development group. Now the second developer would be either very upset or very confused when they tried to test their members and discovered that their changes were no longer present. They might then try to build their package that just includes CC000002 and then they would be able to test while the other developer would now be prevented from testing.

This illustrates that to use Build and Promote by Change Code successfully you will either need to coordinate your work with the developer's that your build can overlap with or ensure that an overlap can never occur. The only way to ensure that overlap of builds can never occur is for the developers to work in private SCLM development groups. Then and only then can they control all their build outputs.

If private development groups cannot be used, then builds and promotes using CCODEs will need to be carefully coordinated. In a large team that is spread out, this might not be convenient. This is not to say that build and promote by change code should not be used, but rather that it should only be used when it is most advantageous or when it is most needed, and then you can take the proper precautions to use it safely. For example:

- If a small team is sharing a SCLM development group and they all sit near each other, they could coordinate the use of build by change code.
- If a developer is using build by change code without coordination, they can immediately copy the load modules they created to private datasets and test there. This way they do not have to worry that their builds will be overwritten. However they still will need to be concerned that simultaneous builds can overwrite each other. If they do another conditional build of their package right after they do their copy and no outputs are created, then they can be sure that programs they just copied are still valid.
- A copybook that is common to many programs is being enhanced over several days or weeks and during this time one of the programs that includes this unfinished copybook needs to be built and promoted without the unfinished code because it is part of a emergency fix that needs to be promoted immediately or it needs to be tested immediately in the TEST group. In this case build by change code with the EXCLUDE option can be used to exclude the copybook from the build. However the code would need to be immediately promoted out of the development group. It will be mandatory that during the short time of the build and promote that all other developers are notified to not do any builds or promotes.

In summary, build and promote by change can be very useful, but it must be used with care by using it in isolation according to one of the techniques discussed above.

8

Promoting up the hierarchy with SCLM

SCLM promote is the action of moving your source, generated objects, and build maps from one level of the SCLM hierarchy to the next level. For example, you might be developing in a group called DEV, and once finished, you might want to integration-test your work with other applications, so you need to promote your work to the INT group. When you give SCLM the name of the development group, SCLM knows the next group in the hierarchy where it will promote the code to.

Once you have built your application and presumably tested it, then you are ready to promote it. SCLM will not allow you to promote something that does not have a build map. So all of your LEC ARCHDEFs must have been successfully built, and also, any of the HL ARCHDEFs that you plan to promote must also be built. The only modules that can be promoted without a build map are those modules that do not have a build process associated with them. For example, your JCL, PROCs, and COPYBOOKs may just be text members. However, to promote them, these members still need to be included using the PROM keyword in an ARCHDEF member that is built.

Promote will use the build map to determine what needs to be promoted. If you select to promote just a COBOL source member that has been built then SCLM will check the build map and see that there is an OBJ and LIST also to be promoted. However you are more likely to be promoting an architecture definition. Even at the lowest level this will be a LEC ARCHDEF that defines the source included in a load module. SCLM in this case promotes all the source, OBJs, LISTs, and the LOAD and LMAP.

© Copyright IBM Corp. 2007. All rights reserved.

8.1 SCLM Promote

To promote, enter a **P** next to the item that you wish to promote. The promote options panel shown in Figure 8-1 will be displayed. Press Enter and SCLM goes through the promote process. Depending on what you are promoting, this might take some time, for example, if you are promoting a high level ARCHDEF that has a scope that includes many applications and load modules. SCLM needs to run through a verification phase to make sure that everything in the scope of the promote is eligible for promotion. If, for example, a high level ARCHDEF has been built and is about to be promoted, but another developer has edited and saved one of the parts in the scope of the ARCHDEF, then the promote will fail verification until a build is performed.

```
------------------------------------------------------------------------------
                          SCLM Promote - Entry Panel
Command ===>

Promote input:
  Project . . . : SCLM07
  From group  . . DEV1
  Type  . . . . . ARCHDEF              Enter "/" to select option
  Member  . . . . ALLAPPL                 Workstation Promote

  Mode  . . 1  1. Conditional          Scope . . . 1  1. Normal
               2. Unconditional                       2. Subunit
               3. Report                              3. Extended

Output control:
               Ex Sub                  Process . . 1  1. Execute
  Messages . . 1  2    1. Terminal                    2. Submit
  Report . . . 3  2    2. Printer
                       3. Data set     Printer . . H
                       4. None         Volume  . .
```

Figure 8-1 SCLM Promote Entry Panel

Note: The group on the promote panel always defaults to the group as specified on the hierarchy view panel, or if using option 5 for promote, the specified development group. This prevents accidental promotion from intermediate groups. If you wish to promote from an intermediate group, TEST for example, then you need to overtype the group on the promote panel to the required intermediate group.

Archived

The promote messages as shown in Figure 8-2 tell you the progress of the promote. SCLM goes through three phases, Verification as discussed above, Copy, and Purge. Copy will copy all of the promotable parts up to the next level of the hierarchy. Once the copy phase is complete, the purge will remove all of the parts from the level being promoted from.

```
FLM59001 - INVOKING PROMOTE PROCESSOR
FLM09002 - THE PROMOTE REPORT WILL APPEAR IN DOHERTL.PROMOTE.REPORT90
FLM51000 - PROMOTE PROCESSOR INITIATED - 06:27:19 ON 2006/11/21
FLM52000 - INITIATING VERIFICATION PHASE - 06:27:20 ON 2006/11/21
FLM55000 - INITIATING COPY PHASE - 06:27:21 ON 2006/11/21
FLM57000 - INITIATING PURGE PHASE - 06:27:23 ON 2006/11/21
FLM57001 - INITIATING PURGE FROM GROUP: DEV1
FLM58000 - PROMOTE PROCESSOR COMPLETED - 06:27:25 ON 2006/11/21
FLM09008 - RETURN CODE =  0
```

Figure 8-2 Promote messages

The promote report as shown in Figure 8-3 gives you a list of all the parts that have been promoted, with the group they came from and the group they are going to.

```
 TYPE: ARCHDEF

                                                COPIED TO  PURGED FROM
MEMBER      DATE        TIME     MESSAGE        TEST       DEV1
________  __________  ________  ________  _____________  ___________

ALLAPPL   2006/11/20  08:07:52                    X            X

                                                                      PAGE    3

 TYPE: ARCHLEC

                                                COPIED TO  PURGED FROM
MEMBER      DATE        TIME     MESSAGE        TEST       DEV1
________  __________  ________  ________  _____________  ___________

BILLING   2006/11/20  08:07:11                    X            X
STARTAPP  2006/11/20  08:07:19                    X            X

                                                                      PAGE    4

 TYPE: COBOL

                                                COPIED TO  PURGED FROM
MEMBER      DATE        TIME     MESSAGE        TEST       DEV1
________  __________  ________  ________  _____________  ___________

BILLINGP  2006/11/18  07:23:56                    X            X
PRINTAPP  2006/11/15  08:23:29                    X            X
STARTAPP  2006/11/18  03:44:11                    X            X
```

Figure 8-3 Promote report

Archived

The mode of the promote can be conditional, unconditional, or report. Report mode is very useful to tell you what is going to be promoted. Additionally, running a promote in report mode prior to the actual promote will inform you if anything has changed in the scope of the object you are promoting, since it was built. For example, one of the objects in the original scope of the promote might have either been edited, or built, thus invalidating the current promote.

Unconditional promote should not be used however, as it does not guarantee application integrity. If there are some verification errors in the promote, then you will get a partial promote, so some parts of the application will be promoted while those with verification errors will not. To this end, it is not advisable to use unconditional promote.

Archived

9

SCLM utilities

In this chapter we describe how you can use some of the utilities that are available in SCLM option 3.

Archived

© Copyright IBM Corp. 2007. All rights reserved.

9.1 SCLM unit of work processing

A relatively new feature of SCLM, released with z/OS 1.6, was SCLM unit of Work (UOW) processing. This looks at code development from the concept of a unit of work, for example, a change request to fix a bug in an application. In SCLM terms the unit of work is an ARCHDEF. The ARCHDEF defines the scope of the piece of work. What this allows is the ability to look at the contents of an ARCHDEF as a list and then perform SCLM functions such as edit and build against those members. This provides an advantage over normal option 3.1 processing, in that you do not have to keep leaving the member list to enter different types, as all types within the scope of the ARCHDEF are listed in the member list.

An SCLM unit of work is accessed via option 3.11 in SCLM. The initial panel can be seen in Figure 9-1.

```
------------------------------------------------------------------------------
                SCLM Unit Of Work processing - Entry Panel
Option ===>


SCLM Library:
  Project  . : SCLM07
  Group  . . . DEV1
  Type . . . . ARCHDEF     (Must contain Architecture Definitions only)
  Member . . .             (Blank or pattern for member selection list)


Enter "/" to select option             Processing mode for build and promote
/  Hierarchy view                       3   1. Execute
/  Confirm delete                           2. Submit
   Show Member Description                  3. View options
/  View processing options for Edit
/  List include members
```

Figure 9-1 SCLM Unit of Work

Many of the options available are the same as with option 3.1. The main difference is that you have to begin with an architecture definition. This does not mean a data set type of ARCHDEF — what this means is that the part you start with must be an architecture definition. An architecture definition is defined by having the macro keyword ARCH=Y on the FLMLANGL macro for the language definition. By default, SCLM uses ARCHDEF as the architecture definition language, but you can, if you wish, define your own. For example, in ISPF/SCLM development there is an SCLM type defined called APARDEF. This is a library that contains the ARCHDEFs that will define what parts are being changed as part of an APAR to fix a problem. The type is APARDEF, but the members contained in this data set all have a language of ARCHDEF.

So, to use the example we have been using throughout this chapter, we enter the member name of an architecture definition or select a architecture definition member from a list. In Figure 9-2, selecting an ARCHDEF with an **S** will cause another list to be built with the contents of that ARCHDEF. Typing **E** next to an ARCHDEF allows you to edit the ARCHDEF. So you should start your unit of work processing beginning at the highest level ARCHDEF that defines your unit of work.

```
------------------------------------------------------------------------
Work Element List for UOW  ALLAPPL  in SCLM07                   Member 1 of 2
Command ===>                                                 Scroll ===> CSR

S=Sel/edit E=Edit V=View P=Prom C=Build U=Upd A=Acct M=Map D=Del B=Brws
Z=Versions
    Member   Type     Status      Account  Language Chg Date         User
    ALLAPPL  ARCHDEF              (Current UOW ARCHDEF)
    BILLING  ARCHLEC              TEST     ARCHDEF  2006/11/20 08:07 DOHERTL
s   STARTAPP ARCHLEC              TEST     ARCHDEF  2006/11/20 08:07 DOHERTL
****************************** Bottom of data ********************************
```

Figure 9-2 SCLM UOW - Work element list

The first entry in the list is the current UOW ARCHDEF. The usefulness of this is evident once we drill down to the next level. We select the STARTAPP ARCHLEC, which is the ARCHDEF library we defined to contain the LEC ARCHDEFs for each load module. Now we see the contents of the STARTAPP ARCHLEC which includes all the COBOL source members and the copybooks, as shown in Figure 9-3.

Note: One restriction is that the copybooks are not shown if the application has not yet been built. This is because UOW uses the build maps to determine what is in the UOW. If there is no build map, then SCLM will parse the ARCHDEF for the contents, but as there are generally no copybooks listed in the ARCHDEFs, they will not be in the member list is there is no build map.

```
------------------------------------------------------------------------
Work Element List for UOW  STARTAPP in SCLM07                  Member 1 of 3
Command ===>                                                 Scroll ===> CSR

S=Sel/edit E=Edit V=View P=Prom C=Build U=Upd A=Acct M=Map D=Del B=Brws
Z=Versions
    Member   Type     Status      Account  Language Chg Date         User
    STARTAPP ARCHLEC              (Current UOW ARCHDEF)
E   PRINTAPP COBOL               TEST     COBDT    2006/11/15 08:23 DOHERTL
    STARTAPP COBOL                TEST     COBDT    2006/11/18 03:44 DOHERTL
    STARTCPY COPYBOOK             TEST     TEXT     2006/11/18 07:28 DOHERTL
****************************** Bottom of data ********************************
```

Figure 9-3 SCLM UOW - Work element list for STARTAPP

At this point we can select or edit the members in the unit of work and make changes to them. This process behaves in the same way as SCLM 3.1. So placing an **E** next to the PRINTAPP COBOL member will take us into SCLM edit for that member. In option 3.1 of SCLM, when you change a COBOL source, you must go back out to the SCLM Library Utility panel to change the type to ARCHLEC to build the load module to incorporate your changes. That is not necessary from the unit of work panel. As the LEC ARCHDEF is listed as the current UOW on the same panel as the source you can issue the build directly from this panel by placing a C next to the STARTAPP ARCHLEC as shown in Figure 9-4 on page 188.

```
------------------------------------------------------------------------
Work Element List for UOW  STARTAPP in SCLM07                  Member 1 of 3
Command ===>                                                 Scroll ===> CSR

S=Sel/edit E=Edit V=View P=Prom C=Build U=Upd A=Acct M=Map D=Del B=Brws
Z=Versions
    Member   Type     Status      Account  Language Chg Date         User
C   STARTAPP ARCHLEC              (Current UOW ARCHDEF)
    PRINTAPP COBOL    *EDITED     DEV1     COBDT    2006/11/21 07:45 DOHERTL
    STARTAPP COBOL                TEST     COBDT    2006/11/18 03:44 DOHERTL
    STARTCPY COPYBOOK             TEST     TEXT     2006/11/18 07:28 DOHERTL
****************************** Bottom of data ********************************
```

Figure 9-4 Building an ARCHDEF through UOW work element list

This invokes the standard SCLM build process for the LEC ARCHDEF. So any changes you have made to multiple modules included in the scope of the ARCHDEF are picked up by SCLM and builds are performed for those source modules prior to the build for the load module itself.

As you use PF3 to go back through the panels, you can issue builds against the higher level ARCHDEFs if you have them thus ensuring that all the build maps are current prior to issuing a promote. Issuing the promote follows the same process, you select the object you want to promote with a **P** and the standard SCLM promote process is invoked as shown in Figure 9-5.

```
------------------------------------------------------------------------
Work Element List for UOW  ALLAPPL  in SCLM07                  Member 1 of 2
Command ===>                                                 Scroll ===> CSR

S=Sel/edit E=Edit V=View P=Prom C=Build U=Upd A=Acct M=Map D=Del B=Brws
Z=Versions
    Member   Type     Status      Account  Language Chg Date         User
P   ALLAPPL  ARCHDEF              (Current UOW ARCHDEF)
    BILLING  ARCHLEC              TEST     ARCHDEF  2006/11/20 08:07 DOHERTL
    STARTAPP ARCHLEC              TEST     ARCHDEF  2006/11/20 08:07 DOHERTL
****************************** Bottom of data ********************************
```

Figure 9-5 Issuing a promote from UOW work element list

9.1.1 Creating your own options with UOW processing

One of the other advantages with the UOW processing is the fact that you can tailor the display for your own personal needs. Standard SCLM options, similar to option 3.1, are provided by default, such as Edit, Build, Promote, and Delete. As you can see in the previous examples, there is also an extra option provided, Z(versions). This is provided as a sample of how you can add extra options to the UOW Work element panel.

Some examples of how you might want to tailor the UOW commands to provide extra function would be:

- Adding a facility to look at the compile or link-edit listing for a particular member
- Adding a function that allows you to quickly test from the UOW member panel the member you have created or changed

- Adding a deploy facility for when you have created some HTML or REXX that you want to deploy to the HFS.

First of all, you have to write the functions that you will want to call from the UOW member list. The first thing to note is the parameters that are passed through to any execs or programs that you write. SCLM will pass through the following parameters:

- Project name
- Alternate project name
- Group of member selected
- Type of member selected
- Member name

The first of the example options above is to provide a facility to look at the listing that is associated with a source member. This facility is not provided by base SCLM processing, as it is in other SCM products, so to look at a listing you have to select the listing type in option 3.1 and then view the listing. However, the process of getting this information and creating an option in UOW to perform the action is quite straightforward. The REXX code to do this is show in Example 9-1.

Example 9-1 Sample REXX to view listing associated with a source member

```
/* REXX */

   Arg PARM

   Parse var PARM Proj "," Alt "," Group "," Type "," Member

/*--------------------------------------------------------------------*/
/* This exit will get the appropriate listing for the member selected */
/*--------------------------------------------------------------------*/

   address ispexec
   'CONTROL ERRORS RETURN'
   'TBEND   BMAPTABL'
   'CONTROL ERRORS CANCEL'
   tbnames = 'NAMES(ZSBKWRD ZSBMEM ZSBTYPE ZSBDATE ZSBTIME ZSBVER
   ZSBLINE)'
   'TBCREATE BMAPTABL' tbnames 'NOWRITE'
   'TBVCLEAR BMAPTABL'

   'SELECT CMD(FLMCMD GETBLDMP,'||PARM||',BMAPTABL)'
   If RC = 0 Then
   Do
      'TBTOP  BMAPTABL'
      'TBSKIP BMAPTABL'
      Do While (RC = 0)
         /*-----------------------------------------------------*/
         /* If the buildmap keyword is LMAP then the listing is a */
         /* link-edit listing, otherwise it is a compile listing  */
         /*-----------------------------------------------------*/
         If ZSBKWRD = 'LMAP' | ZSBKWRD = 'LIST' Then
         Do
            Acctparm = Proj","Alt","Group","ZSBTYPE","ZSBMEM
            Address TSO 'FLMCMD ACCTINFO,'Acctparm
            If RC = 0 Then
            Do
```

```
               Address ispexec 'vget (ZSADSN)'
               REALDSN = Strip(ZSADSN)
               Address ISPEXEC 'VIEW DATASET('REALDSN'('Strip(ZSBMEM)'))'
            End
            Else
               Say 'ACCTINFO called failed. RC='RC
            Leave
         End
         Else
         'TBSKIP BMAPTABL'
      End
      'TBEND BMAPTABL'
   End

Exit
```

The second example above is to provide a means to test the code you have created or changed. For example, if you are creating ISPF panels or REXX execs to test them, you have to split panels and go to option 7 (Dialog Test). It would be easier to edit the member, such as an ISPF panel, and then just run it with a different line command so that it displays so you can verify your changes. Additionally, if the panel is a stand-alone panel where all the processing is done with panel REXX, you can fully test the panel directly from within SCLM. Some sample REXX code to do this is shown in Example 9-2.

Example 9-2 Sample REXX to test the member being worked on

```
/* REXX */

   Arg PARM

   Parse var PARM Proj "," Alt "," Group "," Type "," Member

/*-------------------------------------------------------------------*/
/* This exit will run the required member in the appropriate way     */
/*-------------------------------------------------------------------*/

   Select
/*-------------------------------------------------------------------*/
/* If an ARCHLEC is selected then we need to get the name of the     */
/* load module that we will run and run that                         */
/*-------------------------------------------------------------------*/
      When (Type = 'ARCHLEC') Then
      Do
         address ispexec
         'CONTROL ERRORS RETURN'
         'TBEND   BMAPTABL'
         'CONTROL ERRORS CANCEL'
         tbnames = 'NAMES(ZSBKWRD ZSBMEM ZSBTYPE ZSBDATE ZSBTIME ZSBVER
         ZSBLINE)'
         'TBCREATE BMAPTABL' tbnames 'NOWRITE'
         'TBVCLEAR BMAPTABL'

         'SELECT CMD(FLMCMD GETBLDMP,'||PARM||',BMAPTABL)'
         If RC = 0 Then
         Do
            'TBTOP  BMAPTABL'
```

```
            'TBSKIP BMAPTABL'
            Do While (RC = 0)
               If ZSBKWRD = 'LOAD' Then
               Do
                  Acctparm = Proj","Alt","Group","ZSBTYPE","ZSBMEM
                  Address TSO 'FLMCMD ACCTINFO,'Acctparm
                  If RC = 0 Then
                  Do
                     Address ispexec 'vget (ZSADSN)'
                     REALDSN = Strip(ZSADSN)
                     Address TSO 'CALL 'REALDSN'('Strip(ZSBMEM)')'
                  End
                  Else
                     Say 'ACCTINFO called failed. RC='RC
                  Leave
               End
               Else
               'TBSKIP BMAPTABL'
            End
            'TBEND BMAPTABL'
         End
      End
/*-------------------------------------------------------------------*/
/* Display the panel. Saves going to Dialog test                     */
/*-------------------------------------------------------------------*/
      When (Type = 'PANELS') Then
      Do
         Acctparm = PARM
         Address TSO 'FLMCMD ACCTINFO,'Acctparm
         If RC = 0 Then
         Do
            Address ispexec 'vget (ZSADSN)'
            REALDSN = Strip(ZSADSN)
            Address ISPEXEC "LIBDEF ISPPLIB DATASET ID('"REALDSN"')"
            Address ISPEXEC 'DISPLAY PANEL('Member')'
            Do While (RC < 8)
               Address ISPEXEC 'DISPLAY PANEL('Member')'
            End
            If RC > 8 then
            Do
               Say ZERRSM
               Say ZERRLM
            End
            Address ISPEXEC "LIBDEF ISPPLIB"
         End
         Else
            Say 'ACCTINFO called failed. RC='RC
      End
/*-------------------------------------------------------------------*/
/* Run a REXX exec                                                   */
/*-------------------------------------------------------------------*/
      When (Type = 'REXX') then
      Do
         Acctparm = PARM
         Address TSO 'FLMCMD ACCTINFO,'Acctparm
```

Archived

```
         If RC = 0 Then
         Do
            Address ispexec 'vget (ZSADSN)'
            REALDSN = Strip(ZSADSN)
            "ALTLIB ACTIVATE APPLICATION(CLIST) DATASET('"REALDSN"')"
            Address ISPEXEC 'SELECT CMD('Member')'
            "ALTLIB DEACTIVATE APPLICATION(CLIST)"
         End
         Else
            Say 'ACCTINFO called failed. RC='RC
      End
      Otherwise
         Say 'No run processing for type 'Type' defined'
   End

Exit
```

The foregoing example only gives the coding for PANELS, REXX, and for running a load module using the CALL statement with no parameters. An example of how this can be expanded would be to provide, for the load module processing, a means to allocate data sets, pass parameters, and submit the test as a batch job.

Once we have our processes written, we now add them to our unit of work options list. From the primary UOW entry panel, there is a menu bar pull-down option as shown in Figure 9-6.

```
 Menu  SCLM  Utilities  Options  Help
---------------------- .-------------------------------. ------------------
                 SCLM U | 3  1. Set UOW Data Set Prefix  |
Option ===>             |    2. Modify SCLM Job Card     |
                        |    3. Define UOW List Commands |
                        '--------------------------------'
SCLM Library:
  Project  . : SCLM07
  Group  . . . DEV1
  Type . . . . ARCHDEF     (Must contain Architecture Definitions only)
  Member . . .             (Blank or pattern for member selection list)


Enter "/" to select option             Processing mode for build and promote
/  Hierarchy view                       1   1. Execute
/  Confirm delete                           2. Submit
   Show Member Description                  3. View options
/  View processing options for Edit
/  List include members
```

Figure 9-6 UOW menu bar option to add new commands

Archived

Select option **3** and you are presented with the panel to enter the commands. You enter the single character command, plus the long name for the command. These will be displayed on the panel. You also enter the name of the function to be called and what type of function it is, such as CMD, PANEL, or PGM. The reason for this is that SCLM issues an ISPF SELECT service call to invoke the function. So the required function must be allocated to the ISPF datasets so that it is found. For example, a CMD must be allocated to SYSPROC or SYSEXEC. The last column you enter is the Status. This is the literal that will be displayed on the panel once the command has been executed. The completed panel for both of our new options is shown in Figure 9-7.

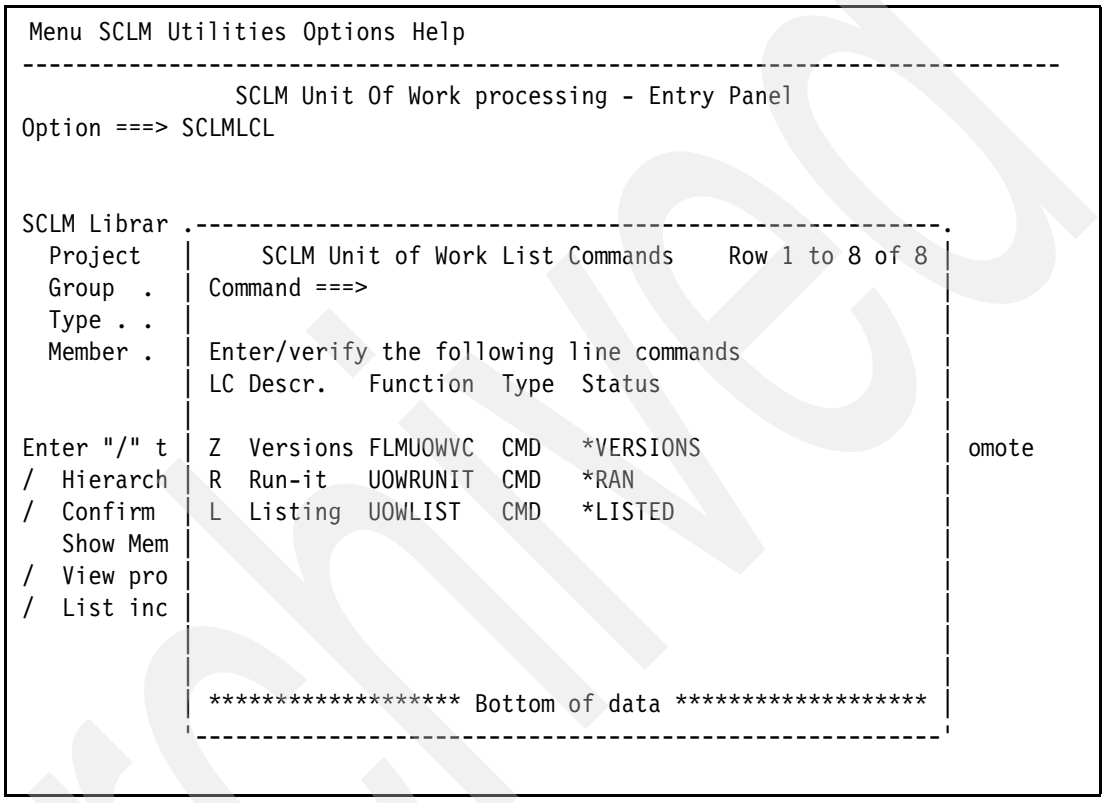


Figure 9-7 Adding UOW Work List Commands

We now go to the member list panel for an architecture definition member to display the contents of it as shown in Figure 9-8. You now see the two new options in the list of available commands at the top of the panel. These commands will be available for any member selected in the list.

Note: In Figure 9-8, the message, `Archdef not current`, can be seen in the top right part of the panel. In this case SCLM is telling us that the ARCHDEF member has been saved but not built. UOW processing uses the build maps whenever possible, but when a valid build map is not available, UOW will read and parse the ARCHDEF source. This could have performance implications. Also, you have to build the ARCHDEF before you can promote it.

Archived

```
 Menu  SCLM  Functions  Utilities  Options  Help
------------------------------------------------------------------------------
Work Element List for UOW  ALLAPPL  in SCLM07              Archdef not current
Command ===>                                                Scroll ===> CSR

S=Sel/edit E=Edit V=View P=Prom C=Build U=Upd A=Acct M=Map D=Del B=Brws
Z=Versions R=Run-it L=Listing
    Member   Type     Status      Account  Language Chg Date         User
    ALLAPPL  ARCHDEF              (Current UOW ARCHDEF)
    BILLING  ARCHLEC              TEST     ARCHDEF  2006/11/20 08:07 DOHERTL
    RATIO    PANELS               DEV1     TEXT     2006/11/22 04:58 DOHERTL
    REXXTEST REXX                 DEV1     TEXT     2006/11/22 06:11 DOHERTL
    STARTAPP ARCHLEC              TEST     ARCHDEF  2006/11/20 08:07 DOHERTL
****************************** Bottom of data ********************************
```

Figure 9-8 Work Element list panel showing new commands

So now if we edit the panel RATIO and make some changes we can test these changes by entering an **R** next to the member as shown in Figure 9-9.

```
------------------------------------------------------------------------------
Work Element List for UOW  ALLAPPL  in SCLM07                  Member 1 of 4
Command ===>                                                 Scroll ===> CSR

S=Sel/edit E=Edit V=View P=Prom C=Build U=Upd A=Acct M=Map D=Del B=Brws
Z=Versions R=Run-it L=Listing
    Member   Type     Status      Account  Language Chg Date         User
    ALLAPPL  ARCHDEF              (Current UOW ARCHDEF)
    BILLING  ARCHLEC              TEST     ARCHDEF  2006/11/20 08:07 DOHERTL
r   RATIO    PANELS   *EDITED     DEV1     TEXT     2006/11/22 08:31 DOHERTL
    REXXTEST REXX                 DEV1     TEXT     2006/11/22 06:11 DOHERTL
    STARTAPP ARCHLEC              TEST     ARCHDEF  2006/11/20 08:07 DOHERTL
****************************** Bottom of data ********************************
```

Figure 9-9 Entering the R(Run-it) command next to a panel member

Based on the code in the option we have created ISPF now displays the panel as shown in Figure 9-10. In the case of this example all of the panel processing is contained in the panel as panel REXX, so we can actually perform unit testing of the panel directly from SCLM.

Archive

```
--------------------------------------------------------------------------------
                          Gear Ratio Calculator
--------------------------------------------------------------------------------

Calculates the "inch" gear of a combination of cranks/sprockets

Wheel size    27

                                Sprocket Size
-------------------------------------------------------------------------------.
     |  12  |  13  |  14  |  15  |  16  |  17  |  18  |  19  |  21  |  23  |
----- ------ ------ ------ ------ ------ ------ ------ ------ ------ ------|
Crank|      |      |      |      |      |      |      |      |      |      |
 48  |108.0 | 99.7 | 92.6 | 86.4 | 81.0 | 76.2 | 72.0 | 68.2 | 61.7 | 56.3 |
 34  | 76.5 | 70.6 | 65.6 | 61.2 | 57.4 | 54.0 | 51.0 | 48.3 | 43.7 | 39.9 |
-------------------------------------------------------------------------------'
```

Figure 9-10 Running the Gear Ration calculator directly from SCLM UOW

Or if we drill own to the next level ARCHDEF, STARTAPP and enter an **L** next to the source member, as shown in Figure 9-11, we are put into view on the compiler listing for that source member as shown in Figure 9-12.

```
--------------------------------------------------------------------------------
Work Element List for UOW  STARTAPP in SCLM07                    Member 1 of 3
Command ===>                                                  Scroll ===> CSR

S=Sel/edit E=Edit V=View P=Prom C=Build U=Upd A=Acct M=Map D=Del B=Brws
Z=Versions R=Run-it L=Listing
    Member   Type     Status      Account  Language Chg Date         User
    STARTAPP ARCHLEC              (Current UOW ARCHDEF)
l   PRINTAPP COBOL                DEV1     COBDT    2006/11/21 07:45 DOHERTL
    STARTAPP COBOL                TEST     COBDT    2006/11/18 03:44 DOHERTL
    STARTCPY COPYBOOK             TEST     TEXT     2006/11/18 07:28 DOHERTL
****************************** Bottom of data ********************************
```

Figure 9-11 Entering the L(List) command next to a source member

Archived

```
------------------------------------------------------------------------------
VIEW       SCLM07.DEV1.LIST(PRINTAPP) - 01.00             Columns 00001 00072
Command ===>                                                 Scroll ===> CSR
****** **************************** Top of Data ******************************
000001 1PP 5655-G53 IBM Enterprise COBOL for z/OS  3.3.1
000002 0Invocation parameters:
000003  NONUM,LIB,XREF(FULL),MAP,OFFSET,NOOPTIMIZE,TEST(ALL,SYM,SEPARATE)
000004 0Options in effect:
000005       NOADATA
000006         ADV
000007         QUOTE
000008         ARITH(COMPAT)
000009       NOAWO
000010         BUFSIZE(4096)
000011       NOCICS
000012         CODEPAGE(1140)
000013       NOCOMPILE(S)
000014       NOCURRENCY
```

Figure 9-12 L(Listing) command places you into view on the listing for the selected source

So unit of work processing can be extended very easily to provide additional processes other than the standard SCLM provided functions.

9.2 SCLM audit and version utility processing

The audit and version utility enables you to view the SCLM audit and version records that SCLM creates when SCLM controlled members are changed and are being audited and versioned by SCLM. Using the audit and version utility, you can view the audit information for a member, compare versions, view versions and retrieve a version to a sequential data set not controlled by SCLM, to a partitioned data set not controlled by SCLM or to an SCLM controlled development group. This utility also enables you to delete audit and version records.

The SCLM project administrator controls the audit and version capabilities through the use of the FLMATVER macros within the project definition. Through the use of this macro the project administrator can decide when members in designated groups and types are to be audited and versioned when they are modified (created, changed, deleted, promoted, and so on). Members in a particular group and type can be audited without being versioned, but not vice versa.

Audit information is stored in a VSAM data set, and versions of the SCLM members are stored in one or more partitioned data sets allocated for this use. The data kept in audit VSAM data sets and the versioning partitioned data sets is for the exclusive use of the audit and version utility. *Do not edit or alter these data sets without using the audit and version utility or the data may be lost.*

Archived

The Audit and Version Utility is Option **8**, from the SCLM Utilities panel. Figure 9-13 shows the panel that appears when you select this option.

```
                SCLM Audit and Version Utility - Entry Panel
Command ===>

  Option . . 1  1. Versioning and Audit Tracking
                2. Versioning only
SCLM Library:
  Project  . : SCLM07
  Group  . . . DEV1
  Type . . . . COBOL
  Member . . .             (Blank or pattern for member selection list)

Selection date range:
  Date from  . .             (Blank or start date for member list)
  Date to  . . .             (Blank or end date for member list)

  Enter "/" to select option
  /  Hierarchy view
```

Figure 9-13 SCLM Audit and Version Utility - Entry Panel

The fields on the SCLM Audit and Version Utility - Entry panel are:

Option "Versioning and Audit Tracking" shows all audited actions for the selected members and date range.

"Versioning only:" shows only those entries that have version data associated with them.

Project The project that you specified on the SCLM Main Menu. An Alternate field also appears if you specified an alternate project definition. These fields can not be changed on this panel.

Group The group for which you want audit and versioning information. The specified group must be defined for auditing.

Type Specify up to four types of member for which you want the version and audit information displayed. The types must be defined for auditing.

Member The member for which you are requesting information. If you leave this field blank all members are displayed. A trailing * may be entered in this field to request a selection list according to a pattern match.

Date from The starting date of the range of dates to search for the specified member. The date must be in the form YYYY/MM/DD. If you leave this field blank, SCLM searches from the beginning of the file to the to date.

Date to The ending date of the range of dates to search for the specified member. The date must be in the form YYYY/MM/DD. If you leave this field blank, SCLM uses the current date as the end date for the search.

Hierarchy view When this option is selected, SCLM searches for audit/versioning records for the current group and for all groups above it in the hierarchy. The current group is determined by the value in the Group field on this panel.

Archived

9.2.1 SCLM version selection panel

Once you enter your audit selection criteria on the entry panel and press Enter, the list of members with audit information matching the selection criteria will show up on the SCLM Version Selection panel. On this panel you can view the audit information and associated accounting information for that version of the member, compare versions of a member or compare the member version with an external data set, delete a version of a member, view the editing history of a version, retrieve a version of a member or view the current contents of a version.

Figure 9-14 shows the version selection panel with the members found.

```
               SCLM Audit and Version Utility - Selection Panel Member 1 of 11
Option ===>                                                    Scroll ===> CSR

Project  . . . : SCLM07

Line Commands:   A Audit Info  C Compare     D Delete      X External Compare
                 H History     R Retrieve    V View

                              Action   Action     Action
S Member   Group    Type      Reason   Date       Time     Userid   V Status
- -------- -------- -------- -------- ---------- -------- -------- - --------
  BILLINGP TEST     COBOL     PROMOTE  2006/12/16 02:47:00 DOHERTL
  CBLDBGEX TEST     COBOL     PROMOTE  2007/01/05 12:18:28 DOHERTL  #
  PRINTAPP TEST     COBOL     PROMOTE  2006/12/16 02:47:00 DOHERTL
  PRINTAPP TEST     COBOL     PROMOTE  2006/12/16 02:46:36 DOHERTL  #
  RDBKC01  TEST     COBOL     PROMOTE  2006/12/23 02:43:53 DOHERTL
  STARTAPP PROD     COBOL     PROMOTE  2007/01/10 05:47:33 PCECK    #
  STARTAPP TEST     COBOL     PROMOTE  2007/01/10 05:46:18 PCECK    #
  STARTAPP TEST     COBOL     PROMOTE  2007/01/10 05:36:23 PCECK    #
  STARTAPP TEST     COBOL     DELETE   2006/12/23 02:38:46 DOHERTL
  STARTAPP TEST     COBOL     PROMOTE  2006/12/19 06:01:35 DOHERTL  #
  STARTAPP TEST     COBOL     PROMOTE  2006/12/16 02:46:37 DOHERTL  #
```

Figure 9-14 SCLM Audit and Version Utility - Selection Panel

The fields for the Version Selection panel, shown in Figure 9-14, are:

Member The names of the members matching the selection criteria on the entry panel that have audit and version information.

Group The group the members matching the selection criteria are stored in.

Type The type the members matching the selection criteria are stored in.

Action Reason The SCLM action that was performed against the specified member that resulted in the audit record. Valid values include:

BUILD, BLDDEL, DELETE, EXT LIB, FREE, IMPORT, LOCK, PROMOTE, STORE, UNLOCK, UPTATHCD (update authorization code), UPTCHGCD (update change code), UPTUENTY (update user entry)

Action Date The date the action listed in the Action Reason field occurred.

Action Time The time the action listed in the Action Reason field occurred.

Userid The user ID of the person who performed the action.

V Indicates, using a hash symbol (#), whether a version of the member exists.

Status Indicates the status of the line command. Otherwise it is blank. Possible values are:

*SELECT, *DELETED, *FAILED, *ERROR, RETRVOLD, and RETRVNEW

To the left of each member listed is a space for entering a line command. You can enter multiple commands on the panel as long as the commands do not conflict.

The available line commands are as follows:

A Displays the audit information for the member. The audit information is displayed on a panel. On this panel you can call an additional panel to display the member's accounting information.

C Displays the Compare panel where another version of the member can be selected to compare the selected member to. The **C** command can only be entered for member versions (not audit records).

D Deletes the audit record in the VSAM audit data set and deletes the versioned member in the partitioned data set.

X Displays the External Compare panel in which you can specify a member in an external data to compare the selected member to. The **X** command can only be entered for member versions (not audit records).

H Displays the history of editing changes made between the selected version and the current version. The **H** command can only be entered for member versions (not audit records).

R Displays the SCLM Audit Retrieve panel in which you can specify a data set into which the version will be retrieved. The **R** command can only be entered for member versions (not audit records).

V Displays the current contents of the selected member version. The **V** command can only be entered for member versions (not audit records).

Archive

9.2.2 SCLM audit and version record

If you enter **A** to display the audit information for a member, the Audit and Version Record panel in Figure 9-15 is shown.

```
                          SCLM - Audit/Version Record
Command ===>

Project  . : SCLM07
Audit data:
  Group  . . . . . . . : TEST            Calling service . . : PROMOTE
  Type . . . . . . . . : COBOL           Action Taken  . . . : PUT
  Member . . . . . . . : STARTAPP        Action Result . . . : COMPLETE
  Audit Date . . . . . : 2006/12/19      Fail Message  . . . :
  Audit Time . . . . . : 06:01:35.82
  Userid . . . . . . . : DOHERTL
  SCLM Change Date . . : 2006/12/19
  SCLM Change Time . . : 06:00:46
Version data:
  Data Set . . . . . . : SCLM07.TEST.COBOL.VERSION
  Member . . . . . . . : STARTAPP        Request format  . . : DELTA
  Change Date  . . . . : 2006/12/19      Current format  . . : DELTA
  Change Time  . . . . : 06:01:36

Enter "/" to select option;
  Display Accounting Information
```

Figure 9-15 SCLM - Audit / Version Record

Archived

If you enter a / in the “Display Accounting Information” field, another panel will be displayed to show the accounting information for the member as shown in Figure 9-16.

```
          SCLM07.DEV1.COBOL(STARTAPP): Accounting Record
Command ===>

   Physical Data Set . : SCLM07.DEV1.COBOL
   Accounting Status . : EDITABLE        Change Group . . . . : DEV1
   Change User ID  . . : DOHERTL         Authorization Code . : P
   Member Version  . . : 8               Auth. Code Change  . :
   Language  . . . . . : COBEDT1L        Translator Version . :
   Creation Date . . . : 2006/11/15      Change Date  . . . . : 2006/12/19
   Creation Time . . . : 08:23:38        Change Time  . . . . : 06:00:46
   Promote User ID . . : DOHERTL         Access Key . . . . . :
   Promote Date  . . . : 2006/12/19      Build Map Name . . . :
   Promote Time  . . . : 06:01:35        Build Map Type . . . :
   Predecessor Date  . : 2006/12/07      Build Map Date . . . : 2006/12/19
   Predecessor Time  . : 16:05:58        Build Map Time . . . : 06:00:46

  Enter "/" to select option
    Display Statistics
    Number of Change Codes     : 0
    Number of Includes         : 1
    Number of Compilation Units : 0
    Number of User Entries     : 0
```

Figure 9-16 Accounting Record from audit panel

9.2.3 SCLM version compare

If you enter **C** to select a version to be compared with other member versions, the SCLM Audit and Version Utility - Compare Panel, shown in Figure 9-17, is displayed. Information about the selected version is shown in the top section of the panel.

Archive

```
               SCLM Audit and Version Utility - Compare Panel  Member 6 of 11
Command ===>                                                  Scroll ===> CSR

SCLM Library:
  Version  . . . : SCLM07.TEST.COBOL(STARTAPP)
  Version Date . : 2006/12/16
  Version Time . : 02:46:37


                                                             Sequence
  Compare Type               Listing Type                    Numbers
     1. File                    1. Delta                       1. BLANK
     2. Line                    2. CHNG                        2. SEQ
     3. Word                    3. Long                        3. NOSEQ
     4. Byte                    4. OVSUM                       4. COBOL

Listing DS Name

                              Action   Action     Action
S Member   Group    Type      Reason   Date       Time     Userid   V
- -------- -------- -------- -------- ---------- -------- -------- -
  STARTAPP PROD     COBOL    PROMOTE  2007/01/10 05:47:33 PCECK    #
  STARTAPP TEST     COBOL    PROMOTE  2007/01/10 05:46:18 PCECK    #
```

Figure 9-17 SCLM Audit and Version Utility - Compare Panel

The bottom section of the panel lists all the matching versions of the member that were included in the initial version selection results. If the Hierarchy View option was selected on the SCLM Audit and Version Utility - Entry Panel, member versions in different groups appear on this list and can be compared. To the left of each version listed is a space for entering the **S** line command. This command selects the version against which you want to compare the currently selected member version. You can only select one of the listed versions.

You can control the compare processing using the following fields:

Compare Type Specifies the granularity of the comparison, ranging from entire member to member (File) comparison down to single Byte differences. Line compare is useful for source data. Word compare is most useful for text data.

Listing Type Specifies the context scope of the listing report. You can get a listing with summary information only (OVSUM), single line differences between files (Delta), differences plus or minus the five unchanged lines before and after changed lines (CHNG), or a listing that includes all of the lines in both files (Long).

Sequence numbers Specifies whether sequence numbers in the compared files are to be ignored or treated as data. Choose SEQ to ignore differences in standard sequence number columns 72 through 80 for FB LRECL 80 members. Choose NOSEQ to treat all columns in the files as data. Choose COBOL to ignore differences in columns 1 through 8 of the data. Choose Blank to cause SuperC to ignore standard sequence number columns if the data set is FB 80 or VB 255. Otherwise, the comparison processes these columns as data.

Listing DS Name The data set into which the compare listing is written. You can preallocate this data set, or let ISPF create one for you. If this data set is partitioned, you must specify a member name.

9.2.4 SCLM external compare

If you enter **X** to select a version to be compared with an external data set, the SCLM Audit and Version Utility - External Compare Panel, shown in Figure 9-18, is displayed. Information about the selected version is shown in the top section of the panel. Fill in the fields that follow as needed to start the compare:

SCLM Group To compare a version against a version of the member stored within SCLM, place a / against the SCLM Group and specify the group. This will search for the first occurrence of the selected member in the hierarchy starting at this group. The group can be in another leg of the hierarchy than the one you specified on the initial Audit and Version Utility panel.

ISPF Data Set To compare a version against an ISPF data set place a / against the ISPF Data Set and specify the data set name. This will search for the member in the data set.

Member If you have chosen to compare a version against an ISPF data set, you can specify the member to be used in the comparison.

Note: The SCLM Group and ISPF Data Set options are mutually exclusive, that is, you can only choose one of these options. If a / is placed in both options, a message will state that the ISPF Data Set option is invalid.

```
          SCLM Audit and Version Utility - External Compare
Command ===>

SCLM Library:
  Version . . . . : SCLM07.TEST.COBOL(STARTAPP)
  Version Date  . : 2006/12/16
  Version Time  . : 02:46:37

Compare Version with:
     SCLM Group . .
     ISPF Data Set
          Member
                                                          Sequence
  Compare Type               Listing Type                 Numbers
     1. File                    1. Delta                    1. BLANK
     2. Line                    2. CHNG                     2. SEQ
     3. Word                    3. Long                     3. NOSEQ
     4. Byte                    4. OVSUM                    4. COBOL

Listing DS Name . .
```

Figure 9-18 SCLM Audit and Version Utility - External Compare

Archive

You can control the compare processing using the following fields:

Compare Type Specifies the granularity of the comparison, ranging from entire member to member (File) comparison down to single Byte differences. Line compare is useful for source data. Word compare is most useful for text data.

Listing Type Specifies the context scope of the listing report. You can get a listing with summary information only (OVSUM), single line differences between files (Delta), differences plus or minus the five unchanged lines before and after changed lines (CHNG), or a listing that includes all of the lines in both files (Long).

Sequence numbers Specifies whether sequence numbers in the compared files are to be ignored or treated as data. Choose SEQ to ignore differences in standard sequence number columns 72 through 80 for FB LRECL 80 members. Choose NOSEQ to treat all columns in the files as data. Choose COBOL to ignore differences in columns 1 through 8 of the data. Choose Blank to cause SuperC to ignore standard sequence number columns if the data set is FB 80 or VB 255. Otherwise, the comparison processes these columns as data.

Listing DS Name The data set into which the compare listing is written. You can preallocate this data set, or let ISPF create one for you. If this data set is partitioned, you must specify a member name

9.2.5 SCLM version retrieve

If you enter **R** to select a version to be retrieved, the SCLM Audit and Version Utility - Retrieve Panel, shown in Figure 9-19, is displayed. Information about the selected version is shown in the top section of the panel. Enter the following fields to specify where to retrieve the member:

To Group The SCLM Group into which the version is to be retrieved.

To Type The SCLM Type into which the version is to be retrieved.

Auth Code The Authorization code used in the retrieval process. If left blank, this defaults to the Authorization code from the selected member version.

Data Set Name The ISPF data set into which the version is to be retrieved. If the data set is a PDS, a member must be specified. (Note that if a data set name is specified in this field, the To Group and To Name fields are ignored.)

Archive

```
              SCLM Audit and Version Utility - Retrieve Panel
Command ===>

SCLM Library:
  Version  . . . : SCLM07.TEST.COBOL(STARTAPP)
  Version Date . : 2006/12/16
  Version Time . : 02:46:37

SCLM retrieve group and type:
  To Group . . . .                Authorization code . .
  To Type  . . . .                (Defaults to auth code from audited member)

Other Data set:
  Data Set Name
```

Figure 9-19 SCLM Audit and Version Utility - Retrieve

9.2.6 Viewing a version

If you enter **V**, you can view the selected version. Here is an example of the first panel of a member that is being viewed as shown in Figure 9-20.

```
VIEW       PCECK.VERBROWS.SLFSE54A                          Columns 00001 00072
Command ===>                                                  Scroll ===> CSR
****** ***************************** Top of Data ******************************
010004 000100* -------------------------------------------------
020004 000200*    IDE Sample Program
030004 000300* -------------------------------------------------
040004 000400 Identification Division.
050004 000500 Program-ID.  StartApp.
060004 000600
070004 000700 Data Division.
080004 000800 Working-Storage Section.
090004 000900
100004 001000 COPY STARTCPY.
110004 001100
120004 001200 Procedure Division.
130004 001300
140004 001400     Initialize Program-pass-fields
150004 001500                Program-other-fields
160004 001600                Program-flags.
170004 001700
```

Figure 9-20 Sample View session

Archived

9.2.7 Viewing version history

If you enter **H**, you can display the history of changes made between the selected version and the current version. The Key column indicates whether each line has changed and if so, in which version. Here is an example of using this function, as shown in Figure 9-21.

```
VIEW       SYS07005.T065916.RA000.PCECK.VHIST.H01         Columns 00001 00072
Command ===>                                               Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001  Version History for Member: STARTAPP DSN: SCLM07.TEST.COBOL.VERSION
000002           Changes since, but not including Version 1
000003
000004 CURRENT 0      06/12/19  06:01:35.82  DOHERTL
000005 VERSION 1      06/12/16  02:46:37.18  DOHERTL
000006
000007 |------Key-----||----------Description----------|
000008 Ixxxxxx         Inserted into Version xxxxxxx
000009         Dxxxxxx Deleted from Version xxxxxxx
000010  (blank)        Unchanged since current version
000011
000012 |-------Key-----|-----------------------------------Source-----------
000013                 000100* ------------------------------------------------
000014                 000200*    IDE Sample Program
000015                 000300* ------------------------------------------------
000016                 000400 Identification Division.
000017                 000500 Program-ID.  StartApp.
000018                 000600
000019                 000700 Data Division.
.
.
.
000024                 001200 Procedure Division.
000025                 001300
000026                 001400     Initialize Program-pass-fields
000027                 001500                Program-other-fields
000028                 001600                Program-flags.
000029                 001700
000030 I0              001800*    Move "ITSO Redbooks team" to Input-name
000031         D0      001800     Move "ITSO Redbooks team" to Input-name
000032                 001900     Perform until Loop-done
000033 I0              002000         Display " "
000034 I0              002100         Display "Enter a name or Q to quit:"
000035 I0              002200         Move Spaces to Input-name
000036 I0              002300         Accept Input-name
000037         D0      002000*        Display " "
000038         D0      002100*        Display "Enter a name or Q to quit:"
000039         D0      002200*        Move Spaces to Input-name
000040         D0      002300*        Accept Input-name
000041                 002400         IF Input-name = Spaces
000042                 002500           Move "E" to Input-name
000043                 002600         End-IF
```

Figure 9-21 Sample Version History session

Archived

9.2.8 Audit reports

SCLM does not provide an easy way to create reports of audit data. The SCLM audit and version utility is the only panel driven method to retrieve audit information. However, SCLM does provide a program API that can be called from REXX or other programming languages to retrieve individual audit records. You can use this API to write your own audit reporting program.

Example 9-3 is a sample REXX that does this. It calls the SCLM VERINFO service in a loop that will retrieve all audit records for a SCLM project for a specific group and type. The type can be entered as a ***** for all types. The output is written to ddname SYSOUT, which you can set to a dataset or to SYSOUT, which the sample JCL shows. The output is unsorted and without headers, so you may want to modify this REXX to do some sorting and to add report headers. Or you could feed the output into a program you write or to a sort program to sort the output according to your needs and to add headers. You can also modify the **inputline** stem variable to add to or to modify the information that is written out for each record. The chapter "Invoking the SCLM Services" in the *SCLM Guide and Reference* has the variables that are set by VERINFO and therefore can be used in this REXX and can be added to the output line.

Example 9-3 Sample REXX to write out audit data

```
/* REXX */
/*********************************************************************/
/*  This exec will extract audit records based upon a project, group  */
/*  and type specified and will write them to SYSOUT                  */
/*********************************************************************/
trace o

PARSE UPPER ARG proj grp type

x = time('R')
prj = STRIP(proj)
grp = STRIP(grp)
type = STRIP(type)

say 'Project: ' proj
say 'Group:   ' grp
say 'Type:    ' type

IF type = '*' THEN
  tp = ''
ELSE
  tp = type

ADDRESS "TSO"

l = 0
w = 0
z = 0

"FLMCMD VERINFO,"prj",,"grp","tp",,,,,,,FORWARD,MSGS"
saverc = rc

/* variable l is used to exit if a infinite loop occurs; the method */
/* used to increment time and thus to move to the next record may   */
/* not be fool proof.  A time of 60.00 (59.99 incremented by one)   */
```

Archives

```
/* may not move the record forward.  It may move it to a previous   */
/* 00.00 record and thus create a infinite loop situation.  You     */
/* will need to adjust the constant 30000 when the number of        */
/* your audit records exceed 30000.                                 */
DO WHILE ((saverc = 0) & (l < 30000))
  /* if reporting on specific type and the next record read is not */
  /* that type then LEAVE because all records for the type have    */
  /* been read                                                     */
  IF type <> '*' THEN
    IF type <> zsvtype THEN
      LEAVE

  l = l + 1
  w = w + 1

  if w = 100 THEN
    do
      say 'records processed is: ' l
      say 'Elapsed time: ' time('E')
      w = 0
    end

  /* if audit record is a good audit record then save  */
  IF STRIP(zsvreslt) = 'COMPLETE' THEN DO
    z = z + 1
/* write out audit member, audit type, audit date, audit time, audit userid,*/
/* change group, change date, change time, change userid and audit event. */
    inputline.z = zsvambr zsvtype zsvdate zsvtime zsvuser zsalgrp
    inputline.z = inputline.z zsaldate zsaltime zsaluser zsvserv
  END

  /* set variables up to read in next record                       */
  mbr = zsvambr
  dt = zsvdate
  tp = zsvtype
  newtime = RIGHT((RIGHT(zsvtime,5) + 0.01),5,0)

  tm = LEFT(zsvtime,6)||newtime
  "FLMCMD VERINFO,"prj",,"grp","tp","mbr","dt","tm",,,,,FORWARD,MSGS"
  saverc = rc
END

inputline.0 = z

/* records read */
say 'l is: ' l

/* write out audit records */
"EXECIO * DISKW "sysout" (STEM inputline. FINIS)"

say 'Elapsed time after verinfo: ' time('E')

EXIT rc
```

Example 9-4 is JCL that can be used to execute the VERINFO program. You will need to add a job card and adjust the ISPF and SYSPROC datasets according to your environment. You will also need to change the call to VERINFO in the last line of the program to pass in your project and the group you want the audit info extracted for. Also adjust the last argument, which is for TYPE, if you only want a specific type reported.

Example 9-4 VERINFO JCL

```
//VERINFO  EXEC PGM=IKJEFT01,REGION=4096K
//ISPMLIB   DD DSN=ISPF.ZOSR8.SISPMENU,DISP=SHR
//ISPSLIB   DD DSN=ISPF.ZOSR8.SISPSENU,DISP=SHR
//ISPPLIB   DD DSN=ISPF.ZOSR8.SISPPENU,DISP=SHR
//ISPLLIB   DD DSN=ISPFLPA.ZOSR8.SISPLPA,DISP=SHR
//          DD DSN=ISPFLNK.ZOSR8.SISPLOAD,DISP=SHR
//ISPTLIB   DD DSN=ISPF.ZOSR8.SISPTENU,DISP=SHR
//SYSPROC   DD DSN=SCLM.PROJDEFS.REXX,DISP=SHR
//ISPTABL    DD UNIT=SYSDA,DISP=(NEW,PASS),SPACE=(CYL,(1,1,5)),
//            DCB=(LRECL=80,BLKSIZE=19040,DSORG=PO,RECFM=FB),
//            DSN=&TABLE1                    TEMPORARY TABLE LIBRARY
//ISPPROF  DD UNIT=SYSDA,DISP=(NEW,PASS),SPACE=(CYL,(1,1,5)),
//            DCB=(LRECL=80,BLKSIZE=19040,DSORG=PO,RECFM=FB),
//            DSN=&TABLE1                    TEMPORARY TABLE LIBRARY
//ISPLOG   DD SYSOUT=*,
//            DCB=(LRECL=120,BLKSIZE=2400,DSORG=PS,RECFM=FB)
//FLMMSGS  DD  DUMMY
//MSGS     DD SYSOUT=*
//LISTING  DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//*YSOUT   DD  DSN=PCECK.VERINFO.OUTPUT,
//*            UNIT=SYSDA,SPACE=(TRK,(20,20)),DISP=(NEW,KEEP,KEEP),
//*            DCB=(LRECL=80,RECFM=FB,BLKSIZE=14960)
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//* THE PROJECT, GROUP AND TYPE ARE PASSED TO VERINFO
//SYSTSIN  DD *
ISPSTART CMD(%VERINFO SCLM01 PROD *)
/*
//*
```

Example 9-5 shows some sample output from the job. The first set of dates / times is when the audit event (PROMOTE) occurred followed by the userid who initiated the PROMOTE. The second set of dates / times is the userid who last changed the member and in which development group it was changed in. The type of audit event comes last.

Example 9-5 Sample Output from the VERINO job

```
SCLMTST1 SOURCE   07/01/07 14:53:00.35 PCECK    DEV1     07/01/07 14:15:07 DOHERTL  PROMOTE
SCLMTST2 SOURCE   07/01/07 14:53:00.41 PCECK    DEV1     07/01/07 14:15:27 DOHERTL  PROMOTE
```

Archive

Archived

Part 3

SCLM: Beyond the basics

In this part of the book, we describe some areas of SCLM that are beyond basic usage — for example, defining user exits to perform certain tasks that might be specific to your environment, or defining language translators to prepare modules for debugging with the IBM Debug Tool.

Archived

© Copyright IBM Corp. 2007. All rights reserved.

Archived

10

Member level locking

In this chapter we describe a feature in the Software Configuration Library Manager (SCLM), called member locking, which was introduced in z/OS 1.8. This functionality, along with all of the SCLM z/OS 1.8 line items, was retrofitted to z/OS 1.6 via APAR OA18509 and z/OS 1.7 via APAR OA17599. This function provides the ability to lock a member so that it cannot be updated by another SCLM user.

SCLM was designed in such a way that each developer would have their own development group in SCLM so that they could work on code in isolation to others. Many customers converting to SCLM from other vendors' products, or from Library Management Facility (LMF) were used to working in a single development group. This posed a problem, as when SCLM stored the member in the development group, any developer who had access to the development group could then edit that member — even if the original developer had not finished making their changes.

The member level locking feature only allows the developer that stored the member in the development group to be able to edit the member while it is in the development group. Once the developer has completed their changes and promoted the member up to the next level in the hierarchy, then it can be edited by a different developer.

Additionally, the member level locking provides the provision for a developer to transfer ownership to another user, for example, if they are about to go on vacation. Also, the concept of an administrator userid is introduced such that an administrator can perform the transfer, for example, if the original developer leaves the company while still owning members in the development group.

In this chapter we look at how member level locking is activated by the SCLM administrator and how it works to protect members in the development group.

© Copyright IBM Corp. 2007. All rights reserved.

10.1 Activating member level locking

Member level locking is activated through three new macro parameters on the FLMCNTRL macro. Your SCLM administrator will have to perform the following steps:

1. Allocate a new SCLM control VSAM file.

   This is a VSAM file much the same as the VSAM account file. It is currently designed to just hold the SCLM administrator ids, therefore, it is not necessary to have one file per level of the hierarchy. In fact, there can be only one control file. Sample JCL to allocate this file is provided in ISP.SISPSAMP(FLM02CNT) or whatever your ISPF libraries are called.

2. Set the new MEMLOCK parameter on the FLMCNTRL macro to Y.

   Setting MEMLOCK=Y will turn on member level locking for the SCLM project.

3. Set the new CONTROL parameter on the FLMCNTRL macro to the name of the SCLM control VSAM file.

4. Set the new ADMINID parameter on the FLMCNTRL macro to the name of an initial SCLM administrator.

   The initial SCLM administrator defined by the ADMINID parameter can then use a new menu item to add additional SCLM administrator ids if required.

5. Assemble the project definition.

Figure 10-1 shows the FLMCNTRL macro in the project definition for SCLM07 showing the three new macro parameters.

```
 TITLE '*** PROJECT DEFINITION FOR SCLM07       ***'
SCLM07   FLMABEG
*
*********************************************************************
*               PROJECT CONTROLS
*********************************************************************
*
*
        FLMCNTRL ACCT=SCLM07.ACCOUNT.FILE,                          C
              VERS=SCLM07.VERSION.FILE,                             C
              VERCOUNT=5,                                           C
              MEMLOCK=Y,                                            C
              CONTROL=SCLM07.CONTROL.FILE,                          C
              ADMINID=DOHERTL,                                      C
              MAXVIO=999999,                                        C
              VIOUNIT=VIO
```

Figure 10-1 Sample FLMCNTRL macro showing new member level locking parameters

Archived

10.2 Adding new SCLM administrator ids

A new SCLM primary option menu has been added to facilitate adding additional SCLM administrator ids if required. Each of these administrator userid will have the ability to edit members owned by different users in the development group in addition to being able to transfer ownership to a different user. Figure 10-2 shows the new SCLM main menu with the **A** (SCLM Admin) option.

```
------------------------------------------------------------------------------
                                SCLM Main Menu
Option ===>

Enter one of the following options:

   1  View         ISPF View or Browse data
   2  Edit         Create or change source data in SCLM databases
   3  Utilities    Perform SCLM database utility/reporting functions
   4  Build        Construct SCLM-controlled components
   5  Promote      Move components into SCLM hierarchy
   6  Command      Enter TSO or SCLM commands
   6A Easy Cmds    Easy SCLM commands via prompts
   7  Sample       Create or delete sample SCLM project
   8  Admin        SCLM Administrator Toolkit
   A  SCLM Admin   Maintaining SCLM administrators
   X  Exit         Terminate SCLM


SCLM Project Control Information:
   Project . . . . SCLM07     (Project high-level qualifier)
   Alternate . . .            (Project definition: defaults to project)
   Group . . . . . DEV1       (Defaults to TSO prefix)
```

Figure 10-2 New SCLM Admin option on SCLM main menu

When you select the **A** option for the first time, the list of administrator ids is empty. The default userid specified in the project definition is not listed, only those added to the control file via the SCLM Admin option. Figure 10.3 shows the SCLM Admin panel when it is first invoked.

```
------------------------------------------------------------------------------
                          Maintain Administrators
Command ===>                                                Scroll ===> CSR

A=Add       D=Delete

    Userid   Name                              Created by
****************************** Bottom of data ********************************
```

Figure 10-3 Maintain Administrators panel

To add a user, enter an **A** command on the command line, and the Add Administrator Userids panel shown in Figure 10-4 is displayed. Enter the userid of the SCLM administrator and their name and press Enter to add the userid.

```
------------------------------------------------------------------------------
                          Add Administrator Userids
 Command ===>

 USER DETAILS
   User ID  . . PCECK
   User Name    Peter Eck
```

Figure 10-4 Add Administrator Userids panel

If you wish to add any additional userids, you can use the **A** command either on the command line or as a line command. The **D** command only functions as a line command and deletes the selected userid. Once you have added all the required administrators for the SCLM project, the Maintain Administrator panel will be populated with all the userids, as shown in Figure 10-5.

```
------------------------------------------------------------------------------
                          Maintain Administrators            Row 1 to 2 of 2
Command ===>                                                Scroll ===> CSR

A=Add       D=Delete

    Userid   Name                          Created by
    PCECK    Peter Eck                     DOHERTL
    JMEHTA   Jyotindra Mehta               DOHERTL
****************************** Bottom of data ********************************
```

Figure 10-5 List of administrators

Note: The Name field can be overtyped at any time.

10.3 Editing members when member level locking is activated

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

- You are the default SCLM administrator (ADMINID parameter on the FLMCNTRL macro).
- You are defined as an SCLM administrator (option **A** from SCLM main menu).
- The accounting record for the member does not exist at the development level.
- The accounting record exists at the development level and your user ID matches the change user ID on the accounting record.

Archived

If one of the above conditions is not met and you try to edit a member that is already in the development group, you receive a message from SCLM telling you that you are not allowed to edit the specified member. Figure 10-6 shows the SCLM Edit panel when the error message is issued.

```
------------------------------------------------------------------------------
                            SCLM Edit - Entry Panel
 Command ===>

 SCLM Library:
    Project . . : SCLM07
    Group . . . : DEV1     . . . TEST     . . . PROD     . . .
    Type  . . . . COBOL
    Member  . . . RDBKC01          (Blank or pattern for member selection list)

 Initial Macro  . .
 Profile Name . . .                (If blank, defaults to data set type)

 Options
 /  Confirm Cancel/Move/Replace
    Mixed Mode
    Edit on Workstation
    Preserve VB record length

.--------------------------------------------------------------------------.
| An error has occurred. Enter HELP for a detailed description of the error. |
'--------------------------------------------------------------------------'
```

Figure 10-6 SCLM Edit panel when edit is denied

Pressing PF1 for additional help shows the more detailed message that SCLM has issued, as shown in Figure 10-7.

```
TUTORIAL -------------------- Message Display ---------------------- TUTORIAL
Command ===>

 FLM20003 - MEMBER LEVEL LOCKING IS IN FORCE AND USER: DOHERTL HAS
 THE MEMBER LOCKED. PLEASE CONTACT THE SCLM ADMINISTRATOR TO
 UNLOCK THE MEMBER: RDBKC01
```

Figure 10-7 Member level locking error message

You are not allowed to edit this member until the user, in this case DOHERTL, has finished editing the member and either promotes it to the next level of the hierarchy, or deletes the version they are editing in the development group.

If you are the developer who has your userid assigned to the member in the development group, or you are an administrator, then you will be able to edit the member, and SCLM behavior will be as it normally is when editing a member.

Archived

10.4 Transferring ownership

You are allowed to transfer ownership of the member in the development group if one of the following conditions is true:

- You are the default SCLM administrator (ADMINID parameter on the FLMCNTRL macro).
- You are defined as an SCLM administrator (option A from SCLM main menu).
- Your user ID matches the change user ID on the accounting record for the member in the development group.

To transfer ownership, there is a new **T** option on the 3.1 member list panel as shown in Figure 10-8. Enter a **T** next to the member you wish to transfer and press Enter.

```
------------------------------------------------------------------------------
Member List : SCLM07.DEV1.COBOL - HIERARCHY VIEW -              Member 1 of 12
Command ===>                                                  Scroll ===> CSR

A=Account      M=Map        B=Browse      D=Delete      E=Edit
V=View         C=Build      P=Promote     U=Update      T=Transfer

    Member   Status      Account  Language   Text       Chg Date   Chg Time
_   BILLINGP            PROD     COBE       PROD       2007/01/13 16:17:33
_   CBLDBGEX            PROD     COBE       PROD       2007/01/13 16:25:38
_   ENTCOB1             DEV1     COBE       DEV1       2007/01/14 20:08:18
_   PETER1              DEV1     COBE       DEV1       2007/01/11 21:49:40
_   PETER2              DEV1     CBCID2DT   DEV1       2007/01/03 01:20:39
_   PRINTAPP            PROD     COBEDT     PROD       2007/01/13 16:17:19
_   RDBKC01             DEV1     CBCID2DT   DEV1       2007/01/15 13:04:04
t   RDBKC02             DEV1     COBCICD2   DEV1       2006/11/23 01:38:53
_   SCLMTST1            DEV1     CBCID2DT   DEV1       2007/01/11 05:08:30
_   SCLMTST2            DEV1     CBCID2DT   DEV1       2007/01/11 05:08:46
_   STARTAPP            DEV1     COBEDT     DEV1       2007/01/20 13:30:20
_   VCMSCLM             DEV1     COBE       DEV1       2006/12/23 06:01:59
****************************** Bottom of data ********************************
```

Figure 10-8 Entering the T (Transfer) command next to a member

You are presented with the transfer ownership panel as shown in Figure 10-9, where you enter the userid of the person you wish to transfer ownership to.

```
------------------------------------------------------------------------
                Transfer Of Member Locking Ownership
Command ===>

   Member to be updated . . . . : SCLM07.DEV1.COBOL(RDBKC02)

   Old Member Userid  . . . . . : DOHERTL
   New Member Userid  . . . . . . pceck
```

Figure 10-9 Transfer ownership panel

The member list panel will be updated with a message to reflect that the member has been transferred as shown in Figure 10-10.

```
------------------------------------------------------------------------------
Member List : SCLM07.DEV1.COBOL - HIERARCHY VIEW -              Member 1 of 12
Command ===>                                                  Scroll ===> CSR

A=Account      M=Map        B=Browse       D=Delete      E=Edit
V=View         C=Build      P=Promote      U=Update      T=Transfer

    Member   Status      Account  Language   Text       Chg Date   Chg Time
_   BILLINGP             PROD     COBE       PROD       2007/01/13 16:17:33
_   CBLDBGEX             PROD     COBE       PROD       2007/01/13 16:25:38
_   ENTCOB1              DEV1     COBE       DEV1       2007/01/14 20:08:18
_   PETER1               DEV1     COBE       DEV1       2007/01/11 21:49:40
_   PETER2               DEV1     CBCID2DT   DEV1       2007/01/03 01:20:39
_   PRINTAPP             PROD     COBEDT     PROD       2007/01/13 16:17:19
_   RDBKC01              DEV1     CBCID2DT   DEV1       2007/01/15 13:04:04
_   RDBKC02  *TRANSFRD   DEV1     COBCICD2   DEV1       2006/11/23 01:38:53
_   SCLMTST1             DEV1     CBCID2DT   DEV1       2007/01/11 05:08:30
_   SCLMTST2             DEV1     CBCID2DT   DEV1       2007/01/11 05:08:46
_   STARTAPP             DEV1     COBEDT     DEV1       2007/01/20 13:30:20
_   VCMSCLM              DEV1     COBE       DEV1       2006/12/23 06:01:59
****************************** Bottom of data ********************************
```

Figure 10-10 Ownership transferred message

If you are not the owner of the member or are not an administrator, then you will receive the message shown below in Figure 10-11 if you try to transfer ownership of the member.

```
 ------------------------------------------------------------------------------
Member List : SCLM07.DEV1.COBOL - HIERARCHY VIEW -       Option not available
Command ===>                                                 Scroll ===> PAGE

A=Account      M=Map        B=Browse       D=Delete      E=Edit
V=View         C=Build      P=Promote      U=Update      T=Transfer

    Member   Status      Text     Chg Date   Chg Time   Account    Bld Map
_   BILLINGP             PROD     2007/01/13 16:17:33   PROD       PROD
_   CBLDBGEX             PROD     2007/01/13 16:25:38   PROD       PROD
_   ENTCOB1              DEV1     2007/01/14 20:08:18   DEV1
_   PETER1               DEV1     2007/01/11 21:49:40   DEV1       DEV1
_   PETER2               DEV1     2007/01/03 01:20:39   DEV1       DEV1
_   PRINTAPP             PROD     2007/01/13 16:17:19   PROD       PROD
_   RDBKC01              DEV1     2007/01/15 13:04:04   DEV1       DEV1
_   RDBKC02              DEV1     2006/11/23 01:38:53   DEV1       DEV1
_   SCLMTST1             DEV1     2007/01/11 05:08:30   DEV1
_   SCLMTST2             DEV1     2007/01/11 05:08:46   DEV1
.------------------------------------------------------------------------------.
| The Transfer option is only available to SCLM administrators or the user who|
| has the member locked.                                                       |
'------------------------------------------------------------------------------'
```

Figure 10-11 User is not allowed to transfer ownership

Archived

11

NOPROM function

In this chapter we describe a new feature in Software Configuration Library manager (SCLM), called the NOPROM function, which was introduced via APAR OA18614. This function allows a member to be left behind during promotion. Prior to this, SCLM would copy all the required components to the next level.

Archived

© Copyright IBM Corp. 2007. All rights reserved.

11.1 Introduction

You can specify whether an EDITABLE member and its accounting record are promoted to the next level or not. During promotion, the outputs (load modules) built using the non-promoted EDITABLE member are either:

- Rebuilt at the next level (REBUILD)
- Not rebuilt at the next level

Next we describe sample scenarios for each of these possibilities:

- **Load modules rebuilt at the next level (REBUILD):**

  Scenario: There are DB2 table changes in the development group required for the updating of the DB2 DCLGEN copybooks, but these DB2 DCLGEN copybooks are not to be promoted until the DB2 changes are complete. At this stage, you create a fix for a program that uses one of the DB2 DCLGEN copybooks but does not require the other DB2 development changes. This fix needs to be promoted to production.

  When building at the development level, the DB2 copybook is used so the program can be tested. But when promoting the program fix, you want the DB2 copybook not to be promoted and the program to be rebuilt at the next level using the version of the DB2 DCLGEN copybook at that level or above.

- **Load modules not rebuilt at the next level:**

  Scenario: A copybook is being modified in development and a fix for a single program which uses the copybook needs to be promoted to production. However, promoting this copybook would cause problems when building other programs using the copybook after promotion.

  You want to build using the development version of the copybook but, when promoting the program fix, wants the copybook not to be promoted and the program source, load and so on, promoted as is.

  Note: By promoting everything except the copybook and its accounting record, the build map containing the non-promoted member is in a broken state. This is because the build map date and time for the non-promoted member does not match the member account record and member statistics date and time.

11.2 Setting a member as not being promotable

There are three way to specify that a member is not-promotable:

- Use of the `N` line command in Library Utilities (Option 3.1) or Unit of Work (Option 3.11)
- FLMCMD NOPROM service
- FLMLNK NOPROM service

By using one of these methods, you will be able to specify that an EDITABLE member is:

- **Not-promotable, but the build maps containing the not-promoted member will be rebuilt after promotion:**

  For the N line command, you specify “No promote (Rebuild)” on the SCLM non-promoted Member Update panel or, if using the FLMCMD/FLMLNK NOPROM service, you specify the REBUILD parameter. This sets the Account Status field in the accounting record to NOPROM-R.

The accounting status of NOPROM-R is used during build and promote to tell SCLM not to promote the member and ensure that the build maps containing the member are rebuilt.

- **Not-promotable, and the build maps containing the not-promoted member will not be rebuilt after promotion:**

  For the N line command, you specify “No promote (No Rebuild)” on the SCLM Not Promoted Member Update panel or, if using the FLMCMD/FLMLNK NOPROM service, you specify the NOREBUILD parameter. This sets the Account Status field in the accounting record to NOPROM-N.

  The accounting status of NOPROM-N is used during build and promote to tell SCLM not to promote the member and ensure the build maps containing the member are copied as is.

- **Promotable:**

  For the N line command, you specify “Remove no promote status” on the SCLM Not Promoted Member Update panel or, if using the FLMCMD/FLMLNK NOPROM service, you specify the REMOVE parameter. This sets the Account Status field in the accounting record to EDITABLE.

  You can only issue this option against a member with an Account Status of NOPROM-R or NOPROM-N.

11.2.1 Using the N line command in Library Utilities (Option 3.1) or Unit of Work (Option 3.11)

You can only issue the **N** line command in the Library Utilities (option 3.1) or Unit of Work (option 3.11) against a member with an accounting record that has an accounting status of EDITABLE, NOPROM-N, or NOPROM-R.

The **N** line command displays the SCLM Not Promoted Member Update panel as shown in Figure 11-1.

```
                SCLM Not Promoted Member Update
Command ===>

   SCLM Library:
   PROJECT  . . . . : SCLM07
   GROUP  . . . . . : DEV1
   TYPE . . . . . . : SOURCE
   MEMBER . . . . . : FLM01EQU

   Options
   NOPROM:     1. No promote (No Rebuild)
               2. No promote (Rebuild)
               3. Remove no promote status
```

Figure 11-1 SCLM Not Promoted Member Update panel

When you press Enter, SCLM changes the Account Status field to one of these possibilities:

- NOPROM-R, if you specify the “No promote (Rebuild)” option
- NOPROM-N, if you specify the “No promote (No Rebuild)” option
- EDITABLE, if you specify the “Remove no promote status” option and the account status is currently NOPROM-N or NOPROM-R

11.2.2 NOPROM service

The NOPROM service is used to modify the accounting status of a members accounting record to change the processing of a member when it is copied to the next level during promotion.

The format of the FLMCMD NOPROM service call is shown in Figure 11-2.

```
FLMCMD NOPROM,
       project,
       prj_def,
       group,
       type,
       member,
       accesskey,
       [REBUILD|REMOVE|NOREBUILD],
       dd_msgs
```

Figure 11-2 Format of the NOPROM service - FLMCMD invocation

The format of the FLMLNK NOPROM service call is shown in Figure 11-3.

```
lastrc := FLMLNK('NOPROM',
                 sclm_id,
                 group,
                 type,
                 member,
                 accesskey,
                 [REBUILD|REMOVE|NOREBUILD],
                 msg_line);
```

Figure 11-3 Format of the NOPROM service - FLMLNK invocation

Parameters

The parameters for the FLMLNK NOPROM service are as follows:

project The project name. The maximum parameter length is 8 characters.

prj_def The project definition name. It defaults to the project name. The maximum parameter length is 8 characters.

sclm_id An SCLM ID associated with a given project and project definition. The INIT service generates the SCLM ID. The maximum parameter length is 8 characters.

group The group at which the member accounting status is to be changed. The maximum parameter length is 8 characters.

type The type of member whose accounting status is to be changed. The maximum parameter length is 8 characters.

member Specifies the member whose accounting status is to be changed.

access_key The access key to be assigned to the member. It defaults to blank. The maximum parameter length is 16 characters.

Archived

NOREBUILD By specifying this option, SCLM will update the accounting record to specify an accounting status of 'NOPROM-N'. After building the appropriate member, SCLM while promoting will copy the build maps containing the not-promoted member, but the member itself will be left behind.

This option, if specified, causes the build map to copied during the promote, regardless of the FLMLRBLD macro, if the build map references the member not being promoted.

REBUILD By specifying this option, SCLM will update the accounting record to specify an accounting status of 'NOPROM-R'. After building the appropriate member, SCLM while promoting will not copy the build maps containing the not-promoted member and the member itself. Once the copy phase of the promote is complete, the build function is invoked at the level being promoted into. Because the build maps will be missing for members that reference the member not being promoted, those members, and any dependent members, will be rebuilt.

This option allows you to rebuild at the next level using the editable member found at that level or above it in the hierarchy.

REMOVE By specifying this option, SCLM will update the accounting record to specify an accounting status of 'EDITABLE'. SCLM will perform a normal promote of the member.

dd_msgs The ddname indicating the destination of the messages generated by the NOPROM service. If you specify a blank ddname, SCLM routes the NOPROM messages to the default output device, such as your terminal. Otherwise, before you call the NOPROM service, you must allocate the ddname. The following attributes should be used: RECFM=F, LRECL=80, BLKSIZE=80. The maximum parameter length is 8 characters. This parameter is used for FLMCMD only.

msg_line Services that only write one message have a msg_line parameter. Define a program variable that is 80 characters to hold the contents of this message line. This parameter only applies to services called through the FLMLNK interface.

FLMCMD NOPROM service example

Example 11-1 is an example of invoking the FLMCMD NOPROM service to set the member FLM01EQU as not-promotable. Using the parameter ‘REBUILD’ will cause all the build maps containing FLM01EQU to be rebuilt when promoted.

Example 11-1 Calling the NOPROM service with FLMCMD

```
/*REXX*/
/*--------------------------------------------------------------------*/
/*                                                                    */
/* FLMCMD NOPROM command                                              */
/*                                                                    */
/*--------------------------------------------------------------------*/
trace n
 /* allocate FLMMSG DSN */
 PARSE ARG PROJECT ALTPROJ GROUP TYPE MEMBER SVCPARM

ADDRESS TSO
msgdsn = "'"||USERID()||".TEMP.FLMMSG'"
X=MSG('OFF')
IF SYSDSN(msgdsn) <> "OK" THEN DO
```

```
   "ALLOC FI(AAAMSGS) DATASET("msgdsn") ",
        "UNIT(SYSDA) TRACKS SPACE(5 5) ",
        "LRECL(80) BLKSIZE(3120) RECFM(F B) DSORG(PS) ",
        "NEW CATALOG "
   "FREE FI(AAATEMP)"
END
"FREE FI(FLMMSGS) "
X=MSG('ON')
"ALLOC FI(FLMMSGS) DATASET("msgdsn") SHR REUSE"

PROJECT="SCLMPROJ"
ALTPROJ="SCLMPROJ"
GROUP  ="DEV1"
TYPE   ="SOURCE"
MEMBER ="FLM01EQU"
SVCPARM="REBUILD"

"FLMCMD NOPROM,"PROJECT","ALTPROJ","GROUP","TYPE","MEMBER",",
         ","SVCPARM",FLMMSGS"

 "ISPEXEC BROWSE DATASET("msgdsn")"
return
```

FLMLNK NOPROM service example

Example 11-2 is an example of invoking the FLMLNK NOPROM service to set the member FLM01EQU as not-promotable. Using the parameter 'NOREBUILD' will cause SCLM to promote everything except the FLM01EQU member and its accounting record.

Example 11-2 Calling the NOPROM service with FLMLNK

```
IDENTIFICATION DIVISION.
 PROGRAM-ID.   FLM02CBL.
*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
*@@.THIS PROGRAM ALLOWS YOU TO CALL SCLM SERVICES FROM A COBOL   @
*@@ PROGRAM.                                                     @
*@@                                                              @
*@@.THE FUNCTION OF THIS PROGRAM IS TO PERFORM AN SCLM NOPROM ON @
*@@ THE COPYBOOK FLM01EQU IN THE DEV1 GROUP.                     @
*@@                                                              @
*@@ 5647-A01 (C) COPYRIGHT IBM CORP. 1987, 2000                  @
*@@                                                              @
*@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  PROGRAM-NAME       PIC X(08) VALUE 'FLM02CBL'.
 01  SCLM-SERVICE       PIC X(08).
 01  SCLM-PROJECT       PIC X(08).
 01  SCLM-ALT-PROJ      PIC X(08).
 01  SCLM-GROUP         PIC X(08).
 01  SCLM-TYPE          PIC X(08).
 01  SCLM-MEMBER        PIC X(08).
 77  SCLM-SCLM-ID       PIC X(08) VALUE SPACE.
 77  SCLM-APPL-ID       PIC X(08) VALUE SPACE.
 77  SCLM-AUTHCODE      PIC X(08) VALUE SPACE.
```

```
 77  SCLM-NOPROM-FIELD  PIC X(16).
 77  SCLM-ACCESS-KEY    PIC X(16) VALUE SPACE.
 77  SCLM-ACCT-INFO     PIC S9(04) COMP VALUE ZERO.
 77  SCLM-LIST-INFO     PIC S9(04) COMP VALUE ZERO.
 77  SCLM-MSG-ARRAY     PIC S9(04) COMP VALUE ZERO.
 77  SCLM-DD-PRSLIST    PIC X(08) VALUE 'SYSOUT'.
 77  SCLM-DD-MSGS       PIC X(08) VALUE 'MDS3602M'.
 77  SCLM-DD-EXIT       PIC X(08) VALUE SPACE.
 77  SCLM-MSG-LINE      PIC X(80) VALUE SPACE.
*
 LINKAGE SECTION.
PROCEDURE DIVISION.
1-DRIVER.
*-------------------------------------------------------------
* FLMLNK NOPROM,JPHILP,OS2G,DEV1,SOURCE,FLM01EQU,NOREBUILD
*-------------------------------------------------------------
     MOVE ‘SCLMPROJ’      TO SCLM-PROJECT
     MOVE ‘SCLMPROJ’      TO SCLM-ALT-PROJ
     MOVE ‘DEV1’          TO SCLM-GROUP
     MOVE ‘SOURCE’        TO SCLM-TYPE
     MOVE ‘FLM01EQU’      TO SCLM-MEMBER
     MOVE ‘NOREBUILD’     TO SCLM-NOPROM-FIELD

     MOVE 'START'     TO SCLM-SERVICE
     CALL 'FLMLNK' USING SCLM-SERVICE
                         SCLM-APPL-ID.
     DISPLAY 'START RETURN CODE =  ' RETURN-CODE.
     MOVE 'INIT'      TO SCLM-SERVICE
     CALL 'FLMLNK' USING SCLM-SERVICE
                         SCLM-APPL-ID
                         SCLM-PROJECT
                         SCLM-ALT-PROJ
                         SCLM-SCLM-ID
                         SCLM-MSG-LINE.
     DISPLAY 'INIT  RETURN CODE =  ' RETURN-CODE.

     MOVE 'NOPROM'    TO SCLM-SERVICE
     CALL 'FLMLNK' USING SCLM-SERVICE
                         SCLM-SCLM-ID
                         SCLM-GROUP
                         SCLM-TYPE
                         SCLM-MEMBER
                         SCLM-NOPROM-FIELD
                         SCLM-ACCESS-KEY
                         SCLM-MSG-LINE.
     DISPLAY 'NOPROM RETURN CODE =  ' RETURN-CODE.
     IF RETURN-CODE > 0 THEN
        DISPLAY SCLM-MSG-LINE.
     MOVE 'FREE'      TO SCLM-SERVICE
     CALL 'FLMLNK' USING SCLM-SERVICE
                         SCLM-SCLM-ID
                         SCLM-MSG-LINE.
     DISPLAY 'FREE  RETURN CODE =  ' RETURN-CODE.

     MOVE 'END'       TO SCLM-SERVICE
```

Archived

```
     CALL 'FLMLNK' USING SCLM-SERVICE
                         SCLM-APPL-ID
                         SCLM-MSG-LINE.
     DISPLAY 'END   RETURN CODE =  ' RETURN-CODE.
     DISPLAY ' '.
     DISPLAY 'NOPROM COMPLETED'.
     DISPLAY 'LOAD FLMO2CBL ENDED'.

     GOBACK.
 1-EXIT. EXIT.
```

11.3 Process of not promoting a member; causes REBUILD

Next we cover the process of not promoting a member, which causes a REBUILD of the load modules that use it at the next level.

For the following description of this project, the project has the following structure as shown in Figure 11-4.

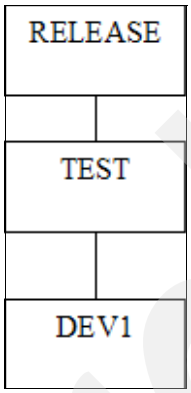


Figure 11-4 Example project structure

Given the situation where the DEV1 group contains a DB2 DCLGEN copybook SCLMTB01 for a DB2 table that was currently in the process of being modified. Now the user wants to be able to build and test in DEV1 using this copybook but does not want SCLMTB01 promoted until they are ready.

Using the new functionality, the member can be modified so it will not be copied to the next level during promotion. Any load modules which use SCLMTB01 will be rebuilt after promotion to the next level (TEST) to use the version of the copybook at TEST or above.

So, to do this, we can use either the **N** line command in the Library Utility (option 3.1) or Unit of Work (option 3.11) or the FLMCMD/FLMLNK NOPROM service. For our example we will use the **N** line command in the Library Utility. By issuing the **N** line command, SCLM will invoke the following panel as shown in Figure 11-5.

Archived

```
                SCLM Not Promoted Member Update
Command ===>

   SCLM Library:
   PROJECT  . . . . : SCLM07
   GROUP  . . . . . : DEV1
   TYPE . . . . . . : COPYLIB
   MEMBER . . . . . : SCLMTB01

   Options
   NOPROM:  2  1. No promote (No Rebuild)
               2. No promote (Rebuild)
               3. Remove no promote status
```

Figure 11-5 SCLM Not Promoted Member Update panel

By specifying 'No promote (Rebuild)' SCLM will modify the accounting record for SCLMTB01 to have an account status of NOPROM-R as shown below in Figure 11-6.

```
          SCLM07.DEV1.COPYLIB(SCLMTB01): Accounting Record
Command ===>


  Physical Data Set . : SCLM07.DEV1.COPYLIB
  Accounting Status . : NOPROM-R        Change Group . . . . : DEV1
  Change User ID  . . : JPHILP          Authorization Code . : P
  Member Version  . . : 2               Auth. Code Change  . :
  Language  . . . . . : TEXT            Translator Version . :
  Creation Date . . . : 2007/01/08      Change Date  . . . . : 2007/01/08
  Creation Time . . . : 16:55:08        Change Time  . . . . : 17:19:15
  Promote User ID . . :                 Access Key . . . . . :
  Promote Date  . . . : 0000/00/00      Build Map Name . . . :
  Promote Time  . . . : 00:00:00        Build Map Type . . . :
  Predecessor Date  . : 2007/01/08      Build Map Date . . . : 2007/01/08
  Predecessor Time  . : 16:56:22        Build Map Time . . . : 17:19:15

 Enter "/" to select option
   Display Statistics
   Number of Change Codes      : 0
   Number of Includes          : 0
   Number of Compilation Units : 0
   Number of User Entries      : 0
```

Figure 11-6 SCLM Account record showing change in Accounting Status to NOPROM-R

While building, SCLM will analyze the components and determine that the member SCLMTB01 has an account status of NOPROM-R. Once the build has completed, SCLM will update the build maps containing the SCLMTB01 member to have a NOPROM build map record.

At the end of the build report, you are shown the build maps that were updated, shown here in Figure 11-7.

```
         ******* N O P R O M   M E M B E R S            *******  Page 5

BUILD     BMAP      NOPROM    NOPROM MBR
MAP       TYPE      MEMBER    TYPE
--------  --------  --------  ----------
SCLMDB2   SOURCE    SCLMTB01  COPYLIB

****************************** Bottom of Data ******************************
```

Figure 11-7 Build report showing NOPROM members

The build map for the SCLMDB2 CCBOL member is shown in Figure 11-8

```
         SCLM07.DEV1.SOURCE(SCLMDB2): Build Map Contents        Line
Command ===>                                                  Scroll ===> CSR
                               Build Map Contents
                               ------------------

  Keyword  Member                                Type     Last Time Modified  Ver
  -------- ------------------------------------ -------- ------------------- ---
  SINC     SCLMDB2                               SOURCE   2007/01/08 17:16:24 3
  OBJ      SCLMDB2                               OBJ      2007/01/08 17:21:17 2
  LIST     SCLMDB2                               LIST     2007/01/08 17:21:17 2
  OUT1     SCLMDB2                               DBRM     2007/01/08 17:21:17 2
  I1*      COBCOPY1                              COPYLIB  2007/01/08 16:57:59 1
  NOPROM   SCLMTB01                              COPYLIB
  I1*      SCLMTB01                              COPYLIB  2007/01/08 17:19:15 2
  I1*      SCLMTB02                              COPYLIB  2007/01/08 17:19:24 2

  * Internal Keywords
    I#     - Included member referenced by SINC member, # = Imbedded Group
    NOPROM - Member was/will be left behind on promotion.
             Use the S line command to view the not promoted member.
```

Figure 11-8 Build map for member SCLMDB2 showing new NOPROM entries

The NOPROM build map record will be used during promote to indicate that the promotion contains member(s) that will not to be copied to the next level.

When promoting SCLMDB2 from DEV1 to TEST:

- SCLM encounters the NOPROM map record on the build maps SCLMDB2.
- SCLM compares the group specified on the NOPROM map record (DEV1), and since it is same as the group we are promoting from (DEV1), the normal date/time validation of the not-promoted member in the build maps will occur.
- SCLM continues and verifies that the SCLM components are current by comparing the build maps, accounting records, and member statistics.

Archived

- While performing this verification, SCLM checks the accounting record for the accounting status for the SCLMTB01 member. Since it is set to NOPROM-R, the member SCLMTB01, its accounting record, and all the build maps containing the NOPROM member SCLMTB01 will not be copied to the next level.
- When copying components to the next level, the SCLMTB01 member, its accounting record and all the build maps (that is, SCLMDB2) containing the SCLMTB01 member will not be copied to the next level.
- Once the copy phase is complete, SCLM invokes a build at the TEST level. Since the build maps containing the SCLMTB01 member were not promoted, these will be rebuilt at the TEST level using the version of SCLMTB01 at the TEST level or above.

At the end of the promote report, you are shown the not-promoted members and the build maps that were affected by the not-promoted member as shown in Figure 11-9.

```
**************************************************************************
**                                                                      **
**               N O T   P R O M O T E D   M E M B E R S                **
**                                                                      **
**************************************************************************
                    BACKUP    NOT PROMOTED
MEMBER    TYPE      MEMBER    OPTION
________  ________  ________  ________

SCLMTB01  COPYLIB             REBUILD

                                                                 PAGE   10
**************************************************************************
**                                                                      **
**             BUILD MAPS AFFECTED BY NOT PROMOTED MEMBERS              **
**                                                                      **
**************************************************************************
BUILD     BMAP      BMAP      NOPROM    NOPROM MBR
MAP       TYPE      REBUILT   MEMBER    TYPE
________  ________  ________  ________  _________

SCLMDB2   SOURCE    YES       SCLMTB01  COPYLIB

***************************** Bottom of Data *****************************
```

Figure 11-9 Promote report showing the not-promoted members

Once the promote is complete, viewing the SCLMDB2 build map, as shown in Figure 11-10, you will see that the SCLMTB01 member used to build SCLMDB2 was dated 2007/01/08 16:56, not 2007/01/08 17:19. Therefore SCLMDB2 was built with an earlier version of the member at the TEST level or higher.

Archived

```
          SCLM07.TEST.SOURCE(SCLMDB2): Build Map Contents        Line
Command ===>                                                   Scroll ===> CSR
                                Build Map Contents
                                -----------------

  Keyword  Member                                Type     Last Time Modified  Ver
  -------- ------------------------------------ -------- ------------------- ---
  SINC     SCLMDB2                              SOURCE   2007/01/08 17:16:24 3
  OBJ      SCLMDB2                              OBJ      2007/01/08 17:35:19 3
  LIST     SCLMDB2                              LIST     2007/01/08 17:35:19 3
  OUT1     SCLMDB2                              DBRM     2007/01/08 17:35:19 3
  I1*      COBCOPY1                             COPYLIB  2007/01/08 16:57:59 1
  I1*      SCLMTBO1                             COPYLIB  2007/01/08 16:56:22 1
  I1*      SCLMTBO2                             COPYLIB  2007/01/08 17:19:24 2

  * Internal Keywords
    I#     - Included member referenced by SINC member, # = Imbedded Group
  ****************************** Bottom of Data ******************************
```

Figure 11-10 Build map for SCLMDB2 post promote

11.4 The process of not promoting a member; NOREBUILD

Next we cover the process of not promoting a member, but promoting the load modules and related members or NOREBUILD.

Prior to this new functionality, promotion of SCLM members would leave the SCLM environment in a complete state where the dates/times in the accounting records, member statistics, and the build maps match. Performing a build at this level will not result in any members being built.

The new functionality of promoting everything except a member and its accounting record (No Promote – No Rebuild) will result in the SCLM being in a broken state. The build maps containing the not-promoted member will have dates/times for the member which was not promoted. This means that SCLM needs to cater for several issues when promoting/building using the `N` line command and specifying 'No promote (No Rebuild)' on the SCLM Not Promoted Member Update panel or using the FLMCMD/FLMLNK NOPROM service and specifying the 'NOREBUILD' parameter. These issues are:

- **We need to back up the not-promoted member:** Since we are promoting the build maps that were built using the not-promoted member, we need to take a backup of this member. This ensures that even if the member has been modified/deleted after we promoted, we will still be able to recreate the outputs (that is, load modules).
- **SCLM needs to be able to promote build maps at a level where a member was left behind:** After promotion, where a member was left behind but the build maps were not rebuilt, the level you promoted into is in a broken state. Normally, when attempting to promote these changes further, SCLM would give you date/time inconsistencies between the build map and accounting record for the not-promoted members.

For the new functionality, SCLM was changed so that when promoting it will ignore the date/time inconsistencies between the build map, accounting record, and member statistics if:

- The build map contains a NOPROM build map record.
- The build level (not viewable) found on the NOPROM build map record does not match the current promote from level.

- **SCLM needs to be able to build at a level where a member was left behind:** After promotion where a member was left behind but the build maps were not rebuilt, the level you promoted into is in a broken state. Normally, when attempting to build at this level, it would cause the build maps with date/time inconsistencies (due to left-behind members) to be rebuilt.

  For the new functionality, SCLM was changed so that when building it will ignore the date/time inconsistencies between the build map, accounting record and member statistics if:

  - The build map contains a NOPROM build map record.
  - The build level (not viewable) found on the NOPROM build map record does not match the current promote from level.

11.4.1 SCLM project setup when not promoting with no rebuilding of build maps

To allow SCLM to back up the not-promoted members, you must perform the following tasks:

- Allocate a NOPROM backup data set. This data set is a fixed block data set with a LECL of 1024.
- Modify the SCLM project definition to add in a NPROMBK parameter on the FLMCNTRL macro to specify the NOPROM backup data set name. The format of the NPROMBK macro parameter is as follows:

  [,NPROMBK=not_promoted_backup_data_set] — The name of the partitioned data set to contain a backup of unpromoted members. This parameter is optional. If you do not specify a NPRMBK value, unpromoted members will not be backed up. No SCLM variables can be used for this parameter.

  The FLMNPROM macro must be used to specify which SCLM editable elements can be marked as not-promotable using the NOPROM service.

- Allocate a CONTROL VSAM data set (if not already defined).

  The sample JCL for defining this member can be located in 'ISP.SISPSAMP(FLM02CNT)'

- Modify the SCLM project definition to add in a CONTROL parameter on the FLMCNTRL macro to specify the CONTROL VSAM data set name. If the FLMNPROM macro is coded the CONTROL parameter on FLMCNTRL must be specified. The format of the CONTROL macro parameter is as follows:

  [,CONTROL=control_data_set] — Specifies the VSAM data set where the additional SCLM control information is stored.

If these actions have not been performed, SCLM will still complete a promotion containing the not-promoted member, however, no backups will be taken. If for any reason you need to recreate the outputs (that is, the Load module) based on the left-behind member, you will need to use the left-behind member. If for any reason this member was modified or deleted after promotion, the SCLM outputs might not be able to be recreated.

Archived

11.4.2 Build containing a not-promotable member (NOREBUILD)

The following description is based on the setup of the SCLM sample project.

We have a situation where a copybook FLM01EQU is being modified in development and a fix for a load module FLM01LD4 which uses the copybook needs to be promoted to production. However, promoting the copybook FLM01EQU would cause problems when building other load modules (that is, FLM01LD3) using the copybook after promotion.

So the user wants to build using the development version of the copybook FLM01EQU. But when promoting the load module FLM01LD4, the user does not want the copybook copied to the next level (TEST).

To set up FLM01EQU so that it is not promoted and that load module FLM01LD4 is not rebuilt, we can use either the **N** line command in the Library Utility (option 3.1) or Unit of Work (option 3.11) or the FLMCMD/FLMLNK NOPROM service. For our example we use the **N** line command in the Library Utility. By issuing the **N** line command, SCLM will invoke the following panel, shown in Figure 11-11.

```
                SCLM Not Promoted Member Update
Command ===>

   SCLM Library:
   PROJECT  . . . . : SCLM07
   GROUP  . . . . . : DEV1
   TYPE . . . . . . : SOURCE
   MEMBER . . . . . : FLM01EQU

   Options
   NOPROM:  1  1. No promote (No Rebuild)
               2. No promote (Rebuild)
               3. Remove no promote status
```

Figure 11-11 SCLM Not Promoted Member Update panel

By specifying 'No promote (No Rebuild)' SCLM will modify the accounting record for FLM01EQU to have an account status of NOPROM-N as shown in Figure 11-12.

Archived

```
         SCLM07.DEV1.SOURCE(FLM01EQU): Accounting Record
Command ===>


  Physical Data Set . : SCLM07.DEV1.SOURCE
  Accounting Status . : NOPROM-N        Change Group . . . . : DEV1
  Change User ID  . . : JPHILP          Authorization Code . : P
  Member Version  . . : 4               Auth. Code Change  . :
  Language  . . . . . : HLAS            Translator Version . :
  Creation Date . . . : 2005/06/02      Change Date  . . . . : 2007/01/09
  Creation Time . . . : 08:02:16        Change Time  . . . . : 12:00:37
  Promote User ID . . :                 Access Key . . . . . :
  Promote Date  . . . : 0000/00/00      Build Map Name . . . :
  Promote Time  . . . : 00:00:00        Build Map Type . . . :
  Predecessor Date  . : 2006/10/11      Build Map Date . . . : 2007/01/09
  Predecessor Time  . : 10:56:51        Build Map Time . . . : 12:00:37

 Enter "/" to select option
   Display Statistics
   Number of Change Codes      : 2
   Number of Includes          : 0
   Number of Compilation Units : 0
   Number of User Entries      : 0
```

Figure 11-12 SCLM Account record showing change in Accounting Status to NOPROM-N

While building, SCLM will analyze the components to be built and determine that the member FLM01EQU has an account status of NOPROM-N. Once the build has completed for FLM01LD4, SCLM will update the build maps containing the FLM01EQU member to have a NOPROM build map record.

At the end of the build report you are shown the build maps that were updated as shown in Figure 11-13.

```
         ******* N O P R O M   M E M B E R S            *******  Page 5

BUILD     BMAP      NOPROM    NOPROM MBR
MAP       TYPE      MEMBER    TYPE
--------  --------  --------  ----------
FLM01MD4  SOURCE    FLM01EQU  SOURCE
FLM01MD5  SOURCE    FLM01EQU  SOURCE
FLM01MD6  SOURCE    FLM01EQU  SOURCE

******************************** Bottom of Data ************************
```

Figure 11-13 Build report showing NOPROM members

The build map for the FLM01MD4 member is shown in Figure 11-14.

```
                              Build Map Contents
                              ------------------

Keyword  Member                                Type     Last Time Modified  Ver
-------- ------------------------------------ -------- ------------------- ---
SINC     FLM01MD4                             SOURCE   2006/02/17 12:05:59 4
OBJ      FLM01MD4                             OBJ      2007/01/09 12:02:53 57
LIST     FLM01MD4                             SOURCLST 2007/01/09 12:02:53 57
NOPROM   FLM01EQU                             SOURCE
I1*      FLM01EQU                             SOURCE   2007/01/09 12:00:37 4

* Internal Keywords
  I#     - Included member referenced by SINC member, # = Imbedded Group
  NOPROM - Member was/will be left behind on promotion.
           Use the S line command to view the not promoted member.
****************************** Bottom of Data ******************************
```

Figure 11-14 Build map for member FLM01MD4 showing new NOPROM entries

The NOPROM build map record in the build maps for FLM01MD4, FLM01MD5, and FLM01MD6 will be used during promote to indicate that the promotion contains member(s) that will not to be promoted.

11.4.3 Promote containing a not-promotable member (NOREBUILD) from the same level containing the NOPROM member

After the member has been specified as being not-promotable (NOREBUILD) and SCLM has performed a build, these changes can then be promoted to the next level (TEST).

When promoting FLM01LD4 from DEV1 to TEST:

- SCLM encounters the NOPROM map record on the build maps FLM01MD4, FLM01MD5, and FML01MD6.
- SCLM compares the group specified on the NOPROM map record (DEV1) and since it is same as the group we are promoting from (DEV1), the normal date/time validation of the not-promoted member in the build maps will occur.
- SCLM continues and verifies that the SCLM components are current by comparing the build maps, accounting records and member statistics.
- While performing this verification, SCLM checks the accounting record for the accounting status for the FLM01EQU member. Since it is set to NOPROM-N all the build maps containing the NOPROM member FLM01EQU will be copied to the next level.
- SCLM reads the CONTROL file to determine the name of the backup member. To do this, it reads the CONTROL file to determine the NOPROM backup number, this is used to generate the backup member name. If the backup number is 00000001 then the backup member will be A0000001.
- SCLM backs up the not-promoted member FLM01EQU into the NOPROM PDS specified by the NPROMBK parameter on the FLMCNTRL macro in the project definition.
- SCLM updates the build maps for FLM01MD4, FLM01MD5, and FML01MD6 at the DEV1 level to add in the backup member name generated above (A0000001) into the NOPROM build map record.

- When copying components to the next level, the FLM01EQU member and its accounting record will not be promoted, but build maps containing the FLM01EQU member will.

11.4.4 Viewing the not-promoted backup member

Figure 11-15 on page 237 shows a build map containing NOPROM build map entries for which a backup has been taken.

```
          SCLM07.DEV1.SOURCE(FLM01MD3): Build Map Contents      Line
Command ===>                                                  Scroll ===> CSR
                               Build Map Contents
                               ------------------

  Keyword  Member                                Type     Last Time Modified  Ver
  -------- ------------------------------------ -------- ------------------- ---
  SINC     FLM01MD3                             SOURCE   2006/10/11 10:52:10 5
  OBJ      FLM01MD3                             OBJ      2007/01/09 17:09:17 79
  LIST     FLM01MD3                             SOURCLST 2007/01/09 17:09:17 79
  I1*      FLM01CON                             SOURCE   2006/10/05 10:05:41 3
_ NOPROM   FLM01EQU                             SOURCE
  I1*      FLM01EQU                             SOURCE   2007/01/09 17:09:00 4

  * Internal Keywords
    I#     - Included member referenced by SINC member, # = Imbedded Group
    NOPROM - Member was/will be left behind on promotion.
             Use the S line command to view the not promoted member.
  ****************************** Bottom of Data ********************************
```

Figure 11-15 Build map showing how to view a backed up source member

By typing an S alongside the NOPROM member you will be able to view the backup of the member that was taken as a part of the promotion of this build map.

11.4.5 Promote containing a not-promotable member (NOREBUILD) from a level not containing the NOPROM member

Now the user has tested the changes at TEST and wants to promote the changes to the RELEASE level. However the SCLM components at TEST are in a broken state as build maps FLM01MD4, FLM01MD5, and FLM01MD6 at TEST will contain the member FLM01EQU but this member was not promoted from DEV1.

So when promoting from TEST to RELEASE:

- SCLM encounters the NOPROM map record in the build maps build maps FLM01MD4, FLM01MD5, and FLM01MD6. The FLM01EQU member will added to the list of members that are not being promoted.
- SCLM compares the group specified on NOPROM map record (DEV1), and since it is not equal to the group we are promoting from (TEST):
  - SCLM skips the date/time validation of the not-promoted member FLM01EQU.
  - SCLM does not back up the not-promoted member FLM01EQU.

- SCLM continues and verifies that the SCLM components are current by comparing the build maps, accounting records and member statistics. If it encounters the member FLM01EQU it will be flagged as being up to date.
- When copying components to the next level the FLM01EQU member and its accounting record will not be promoted but build maps containing the FLM01EQU member will.

11.4.6 Build containing a not-promotable member (NOREBUILD) at a level which does not contain the NOPROM member

After the changes have been promoted to RELEASE the SCLM components will be in a broken state as build maps FLM01MD4, FLM01MD5, and FLM01MD6 will contain the member FLM01EQU but this member was not promoted from DEV1.

Prior to this new functionality, a build performed against FLM01LD4 would rebuild the load module due to the broken state of the build maps FLM01MD4, FLM01MD5, and FLM01MD6. This is not acceptable, as it would override the changes you have just promoted.

So now when performing a build containing a NOPROM build map record where the level specified on the build maps record does match the build level: SCLM performs these tasks:

- SCLM, when analyzing the SCLM components, will encounter the NOPROM map record on the build maps FLM01MD4, FLM01MD5, and FML01MD6.

  For each of these build maps:

  – SCLM compares the group specified on the NOPROM map record (DEV1) to see if it is different to the group you are building at. If it is and the date/time of the accounting record does not match the date/time in the build maps then SCLM knows that the member FLM01EQU has not been promoted.

  – Since the FLM01EQU member has not been promoted, it will be set as ‘up to date’ in the build map and message `FLM44523` issued.

  – If other changes are encountered which would cause the build map to be rebuilt then SCLM issues message `FLM44522` and completes with a return code of 8.

    The above situation occurs if a member used by the build map was changed and promoted, but the build map was not rebuilt. To resolve this problem, you would either need to:

    - Delete the build map causing the problem and rebuild. This will cause the build map to be rebuilt using the version of not-promoted member at the build level or above (that is, RELEASE).
    - Rebuild the build map at the group containing the not-promoted member (DEV1) and re-promote your changes. This will rebuild the build map using the members at the higher level as well the not-promoted member.

11.4.7 Build after promotion of the not-promotable member (NOREBUILD)

If the member FLM01EQU is ready to be promoted into RELEASE, the user can use the **`N`** line command in Library Utilities (option 3.1) or Unit of Work (Option 3.11) or run the FLMCMD/FLMLNK NOPROM service to remove the not-promotable status of the member FLM01EQU. Once this is complete, the accounting record for the member FLM01EQU will look as shown in Figure 11-16.

```
          SCLM07.DEV1.SOURCE(FLM01EQU): Accounting Record
Command ===>


  Physical Data Set . : SCLM07.DEV1.SOURCE
  Accounting Status . : EDITABLE        Change Group . . . . : DEV1
  Change User ID  . . : JPHILP          Authorization Code . : P
  Member Version  . . : 4               Auth. Code Change  . :
  Language  . . . . . : HLAS            Translator Version . :
  Creation Date . . . : 2005/06/02      Change Date  . . . . : 2007/01/09
  Creation Time . . . : 08:02:16        Change Time  . . . . : 12:00:37
  Promote User ID . . :                 Access Key . . . . . :
  Promote Date  . . . : 0000/00/00      Build Map Name . . . :
  Promote Time  . . . : 00:00:00        Build Map Type . . . :
  Predecessor Date  . : 2006/10/11      Build Map Date . . . : 2007/01/09
  Predecessor Time  . : 10:56:51        Build Map Time . . . : 12:00:37

 Enter "/" to select option
   Display Statistics
   Number of Change Codes      : 2
   Number of Includes          : 0
   Number of Compilation Units : 0
   Number of User Entries      : 0
```

Figure 11-16 Account record for FLM01EQU

Promoting the member FLM01EQU or a build map containing FLM01EQU will cause SCLM to invoke a normal promotion process. Once the member FLM01EQU has been promoted to the RELEASE level, if a rebuild of the build maps FLM01MD4, FLM01MD5, and FLM01MD6 has not occurred, then these build maps will still contain the NOPROM build map records.

When building at the RELEASE level after promotion of FLM01EQU:

- SCLM encounters the NOPROM map record on the build maps FLM01MD4, FLM01MD5, and FLM01MD6.
- SCLM compares the group specified on the NOPROM map record (DEV1) to see if it is different to the level you are building at. If it is and the date/time of the accounting record matches the date/time in the build maps, then SCLM knows that the member FLM01EQU has been promoted.
- Hence when building at RELEASE, SCLM removes the NOPROM build map record entries from the build maps FLM01MD4, FLM01MD5, and FLM01MD6, even if they were not re-built.

11.4.8 Restricting the setting of not-promotable

You can restrict the use of the **N** line command in the Library Utility (option 3.1) or Unit of Work (option 3.11) or the FLMCMD/FLMLNK NOPROM service by the use of the FLMNPROM macro within the SCLM project definition.

If FLMNPROM is not specified in the SCLM project definition, there are no restrictions on the use of the **N** line command in Library Utility (option 3.1) or Unit of Work (Option 3.11) or the NOPROM service within the SCLM project.

By specifying the FLMNPROM macro, you will be able to specify which SCLM editable elements can or cannot be marked as not-promotable using the NOPROM service. This macro is required if you wish to use the NOPROM service, and can be specified multiple times. Only members that match the groups, types, and languages associated with the NPROM=YES parameter can be marked as not-promotable using the NOPROM service.

The format of the FLMNPROM macro is shown in Figure 11-17.

```
FLMNPROM
     GROUP=(group1,group2,...)|*
    ,TYPE=(type1,type2,...)|*
    ,LANG=(lang1,lang2,...)|*
    ,NPROM=YES|NO
```

Figure 11-17 FLMNPROM macro format

Parameters

The parameters of the FLMNPROM macro are as follows:

GROUP=(group1,group2,...)|* If NPROM=YES is specified, the names of the “from” groups whose members can be marked as not-promotable. If NPROM=NNO is specified, the names of the “from” groups whose members may not be marked as not-promotable. Use an asterisk (*) to indicate all groups.

,TYPE=(type1,type2,...)|* If NPROM=YES is specified, the names of the “from” types whose members can be marked as not-promotable. If NPROM=NO is specified, the names of the “from” types whose members may not be marked as not-promotable. Use an asterisk (*) to indicate all types.

,LANG=(lang1,lang2,...)|* If NPROM=YES is specified, the languages of those members that can be marked as not-promotable. If NPROM=NO is specified, the names of the languages of those members that may not be marked as not-promotable. Use an asterisk (*) to indicate all languages.

,NPROM=YES|NO Specifies whether members in the specified group(s) and type(s) and with the specified languages may or may not be marked as not-promotable. If you specify YES, the NOPROM service can be used to mark identified members as not-promotable. If you specify NO, the NOPROM cannot be used to mark identified members as not-promotable.

Examples

Example 11-3 shows how to specify that editable members with a language of COBCOPY in the SOURCE type, but in any group, can be marked as not-promotable.

Example 11-3 Specifying FLMNPROM example 1

```
FLMNPROM GROUP=*,TYPE=SOURCE,LANG=COBCOPY,NPROM=YES
```

Example 11-4 shows how to specify that all editable members with a language of COBCOPY in the SOURCE type, in all groups except EMERFIX can be marked as not-promotable.

Example 11-4 Specifying FLMNPROM example 2

```
FLMNPROM GROUP=*,TYPE=SOURCE,LANG=COBCOPY,NPROM=YES
FLMNPROM GROUP=EMERFIX,TYPE=SOURCE,LANG=COBCOPY,NPROM=NO
```

12

SCLM user exit processing

Software Configuration Library Manager (SCLM) provides exits that are break points within the functions and services of SCLM, and they allow the customization of SCLM. By doing so, they allow a customer to incorporate their unique requirements into SCLM. For example, an exit can be used to stop the SCLM processing if the user is not authorized in some way. They can be used to allow external processing, for example, copying load modules to external locations, performing a CICS new copy, or a DB2 bind. Exits can be written in any programming language that can create a load module or through interpretive languages such as CLIST or REXX execs. All of the examples in this chapter are written in REXX.

The exit break points occur at different places within each of the SCLM functions and services.

- Migrate, Save, Import, and Store services and the Edit function have two exits: one during the verification, and one at the end of the process.
- Build has two exit points: initial and notify.
- Promote has four exit points: initial, verify, copy, and purge.
- Delete and delete group functions have three exit points: initial, verify, and notify.
- The Audit/Version delete process has two exit points: verify and notify.

Exits are defined in SCLM using the FLMCNTRL macro within the SLCM project definition. Each exit is defined separately within the FLMCNTRL macro.

© Copyright IBM Corp. 2007. All rights reserved.

12.1 Examples of user exits

Here are some examples of various types of SCLM user exits:

- Authorization checking:
  - Check if the user is allowed to perform certain functions
- Change Code verification:
  - In-house or through products like Tivoli® INFOMAN
- Updating of non-SCLM controlled resources to reflect change in SCLM controlled members:
  - Copy of load modules to “live” load libraries so JCL changes are not needed
  - CICS new copy
  - Deploy code to remote machines
  - Delete from non-SCLM libraries to reflect delete in SCLM
  - Bind processing
- Notification processing
- Interface to other products or internal systems:
  - Package approval products (such as Breeze or INFOMAN)

12.2 SCLM exit call methods

The FLMCNTRL macro defines to SCLM how the exit should be called. This is referred to as the *call method*. Each exit has its own parameter to define the call method. This allows for a variety of programs, REXX execs, CLISTs, etc. for the different exit points within SCLM.

The four call methods for SCLM user exits are:

LINK For load modules that use ISPF variables or services.

ATTACH For load modules not using ISPF services.

TSOLNK For REXX execs and CLISTs.

ISPLNK For execs and CLISTs that use ISPF variables and services.

Archived

12.3 Sample FLMCNTRL MACRO with calls to user exits

Example 12-1 is an example of a FLMCNTRL macro with several SCLM User Exits defined. This example is using: CC Save, Build Notify, Promote Verify, Promote Copy and Promote Purge Exits.

Example 12-1 FLMCNTRL with user exit calls

```
         FLMCNTRL ACCT=SCLM.ACCOUNT.FILE,    * PRIMARY ACCOUNTING      C
               ACCT2=SCLM.ACCOUNT2.FILE,     * SECONDARY ACCOUNTING    C
               VERS=SCLM.AUDIT.FILE,         * PRIMARY AUDIT VSAM      C
               VERS2=SCLM.AUDIT2.FILE,       * SECONDARY AUDIT VSAM    C
               VERPDS=@@FLMDSN.VERSION,      * VERSION PDS NAME        C
               VERCOUNT=10,                  * # OF VERSIONS TO KEEP   C
               DASDUNIT=SYSDA,               * UNIT ALLOC FOR DASD     C
               MAXVIO=500000,                * LIMIT TO ALLOC ON VIO   C
               CCSAVE=CCSAVE,                * CCSAVE USER EXIT        C
               CCSAVCM=TSOLNK,               * METHOD TO CALL EXIT     C
               CCSAVDS=SCLM.PROJDEFS.REXX,   * REXX DATASET            C
               BLDNTF=BLDNTF,                * BUILD NOTIFY USER EXIT  C
               BLDNTFCM=TSOLNK,              * METHOD TO CALL EXIT     C
               BLDNTFDS=SCLM.PROJDEFS.REXX,  * REXX DATASET            C
               PRMVFY=PRMVFY,                * PROMOTE VERIFY USER EXITC
               PRMVFYCM=TSOLNK,              * METHOD TO CALL EXIT     C
               PRMVFYDS=SCLM.PROJDEFS.REXX,  * REXX DATASET            C
               PRMCOPY=PRMCOPY,              * PROMOTE COPY USER EXIT  C
               PRMCPYCM=TSOLNK,              * METHOD TO CALL EXIT     C
               PRMCPYDS=SCLM.PROJDEFS.REXX,  * REXX DATASET            C
               PRMPURGE=PRMPURGE,            * PROMOTE PURGE USER EXIT C
               PRMPRGCM=TSOLNK,              * METHOD TO CALL EXIT     C
               PRMPRGDS=SCLM.PROJDEFS.REXX   * REXX DATASET
```

In this example, a method of calling a common REXX exec for each exit is used. Then from the common REXX exec the individual exits are called. This will allow multiple exits to be called from a single exit point. **We recommend that this method be used for all exits written in REXX**. This allows you to add a new exit at any time without reassembling the project definition and allows for easy analysis by the SCLM admin as to which exits are being used. If no exits are needed for a given exit point then you can code a exit 0 statement in the called common REXX.

Sample common REXX calling multiple exits

Example 12-2 is an example of calling multiple build user exits from a common exit file.

Example 12-2 Common Build Exit

```
/* REXX */
/*********************************************************************/
/*  This exec called from the Build Notify exit, will setup an       */
/*  ISPF environment for the execution of REXX execs.                */
/*********************************************************************/
TRACE o
ARG parm                 /* Parse arguments into variable parm      */
bldexrc = 0
ADDRESS ISPEXEC
/* Perform any necessary external binds */
```

```
"SELECT CMD(EX 'SCLM.PROJDEFS.REXX(BINDEXTR)' '"parm"')"
bldexrc = MAX(bldexrc,rc)

/* Perform any necessary copies and new copies */
IF bldexrc = 0 THEN DO
   "SELECT CMD(EX 'SCLM.PROJDEFS.REXX(COPYEXTR)' '"parm"')"
   bldexrc = MAX(bldexrc,rc)
   END

/* Call Breeze Build Exit */
IF bldexrc = 0 THEN DO
    "SELECT CMD(EX 'SYS1.BRZE.SBZZCLIB(BZZSME01)' '"parm"')"
    bldexrc = MAX(bldexrc,rc)
    END
exit bldexrc
```

The above REXX exec is called from the FLMCNTRL macro on the previous page for the build notify user exit. It shows how multiple exits can be called from a common exit REXX. *The SELECT CMD will establish an ISPF environment for the execution of the individual execs.* This is required for any ISPF variables that might be used in the called exec or if ISPF services are going to be utilized. Notice that the common REXX was called with the easier to code call methodology TSOLNK. The TSOLNK methodology allows the common REXX to call the individual REXX exits using the SELECT command and thus allows the called programs to use ISPF services.

12.4 Components of an exit – parameters and exit files

Data is passed from SCLM to exits in *two* ways:

- *First* it is sent from SCLM through parameters that are passed in a string to the program or REXX exec. These PARMs are divided by *commas*.
- *Second* it is passed through exit files

12.4.1 SCLM exit parameter passing

The SCLM exit parameters are split into two sections. An option list of up to 255 bytes, which is defined in the project definition followed by a number of parameters that SCLM passes through. Usually an option list is not passed.

Depending on the exit point that is called, SCLM sends different PARMs. Different information is needed based on what SCLM functions are being performed. For example, the change code is needed for the change code exits but not for build, promote, or delete exits. Also, the member's language is passed in the PARMs for change code and audit/version delete exits. These exits are processed per member only so that the member name can be passed through the PARMs.

For build, promote, and delete exits many members can be affected by one architecture definition. Therefore, the member name that is passed in the PARM could be for an ARCHDEF and not the source name.

Table 12-1 shows the exact parameters for the various exit types.

Table 12-1 Parameters for various exit types

Change Code Exits	Build and Promote Exits	Audit Delete Exits	Delete Exits
OPTION LIST	OPTION LIST	OPTION LIST	OPTION LIST
GROUP	EXIT TYPE	EXIT TYPE	EXIT TYPE
TYPE	PROJECT	PROJECT	PROJECT
MEMBER	LIBDEF	LIBDEF	LIBDEF
LANGUAGE	USERID	USERID	USERID
USERID	FROM GROUP	GROUP	GROUP
AUTHCODE	TYPE	TYPE	TYPE
CHANGE CODE	MEMBER	MEMBER	MEMBER
	SCOPE	DATE	FLAG
	MODE	TIME	MODE
	TO GROUP	VERSION MEMBER NAME	

12.4.2 Components of exits – parameter parsing

Example 12-3 contains some sample REXX code that shows the PARMs passed by SCLM to a build or promote exit and how they can be parsed into individual REXX variables. A different parameter list is passed for change control, audit/version delete and delete exits.

Example 12-3 Parsing the parameters

```
/* Break out the parms passed from the SCLM user exit */

PARSE UPPER VAR parm extyp ',' proj ',' prjdf ',' tsouid ',',
          fromgrp ',' type ',' member ',' scope ',' mode ',' togrp

extyp    = STRIP(extyp,'T')    /* Exit Type - where in SCLM?     */
proj     = STRIP(proj,'T')     /* Project - high level qualifier */
prjdf    = STRIP(prjdf,'T')    /* Alternate Project Definition   */
tsouid   = STRIP(tsouid,'T')   /* User executing SCLM function   */
fromgrp  = STRIP(fromgrp,'T')  /* From grp(Promote) or build grp */
type     = STRIP(type,'T')     /* Type from the SCLM panel       */
member   = STRIP(member,'T')   /* Member from the SCLM panel     */
scope    = STRIP(scope,'T')    /* Scope entered on SCLM panel    */
mode     = STRIP(mode,'T')     /* Mode entered on SCLM panel     */
```

The exit type is used to determine what process is being used and where you are within SCLM. For example, if the exit type is *BINITIAL*, then this would mean that the exit was called during the initial build exit. The same REXX exec can be called from multiple exit points in SCLM. The exec must examine the exit type variable passed from SCLM to determine who the caller is. Note: in this example no option list has been passed through to the exit. So the first parameter in the variable PARM is the first SCLM passed parameter which is the exit type.

12.4.3 Components of exits – exit file

The *second* way exits receive information from SCLM is through exit files. These files contain *all* the members that were affected by the SCLM function whether it be build, promote or delete. This file is created in SCLM to be used by the *build notify*, *promote verify*, *promote copy*, and *promote purge* exits. The file contains a record for each member with group, type, member name, and status. This file is also available when a Delete Group is performed and a delete notify user exit is defined in the FLMCNTRL macro.

Exit files are not available for: change code exits, audit/version delete exits, build initial exit, promote initial exit, or delete initial exit. For the build exit file, the only members written to the file are those outputs produced due to translator calls. For the promote exit file the members written to the exit file are those members who will be verified, copied or purged.

The *ddname* allocated by SCLM for the exit file is **BLDEXIT** for *build*, **PROMEXIT** for *promote*, and **DGEXIT** for *delete*. This ddname can be used in the exit program to access the exit file.

Exit file layout

```
Group  column 1       (8 bytes)
Type   column 10      (8 bytes)
Member column 19      (8 bytes)
Status column 28      (16 bytes)
```

Table 12-2 shows the exit file status messages.

Table 12-2 Exit file status

Exit name	Status message
BLDNTF	BUILT / DELETED
PRMVFY	PROMOTABLE / NOT PROMOTABLE
PRMCOPY	COPY SUCCESSFUL / COPY FAILED / COPY NOT ATTEMPTED
PRMPURGE	PURGE SUCCESSFUL / PURGE FAILED
DELETE	DELETE SUCCESSFUL / DELETE FAILED / DELETE WARNING
DELGRP	NOT ATTEMPTED

Reading a line in the exit file

Example 12-4 shows sample REXX code that shows the PROMEXIT file being read into stem variables. For each line in the PROMEXIT file, the line is then parsed into four stem variables, which represent the four exit file variables that make up each line of the exit file. The individual stem variables can then be processed in a follow-on do loop.

Example 12-4 Reading the exit file

```
/* Break out the parms passed from the SCLM user exit */
"execio * diskr PROMEXIT (stem exitdata. finis"

Do i=1 to exitdata.0
   Parse var exitdata.i Group.i Type.i Member.i Status.i
End
```

12.5 SCLM exits during an edit

Figure 12-1 shows the SCLM edit user exits.

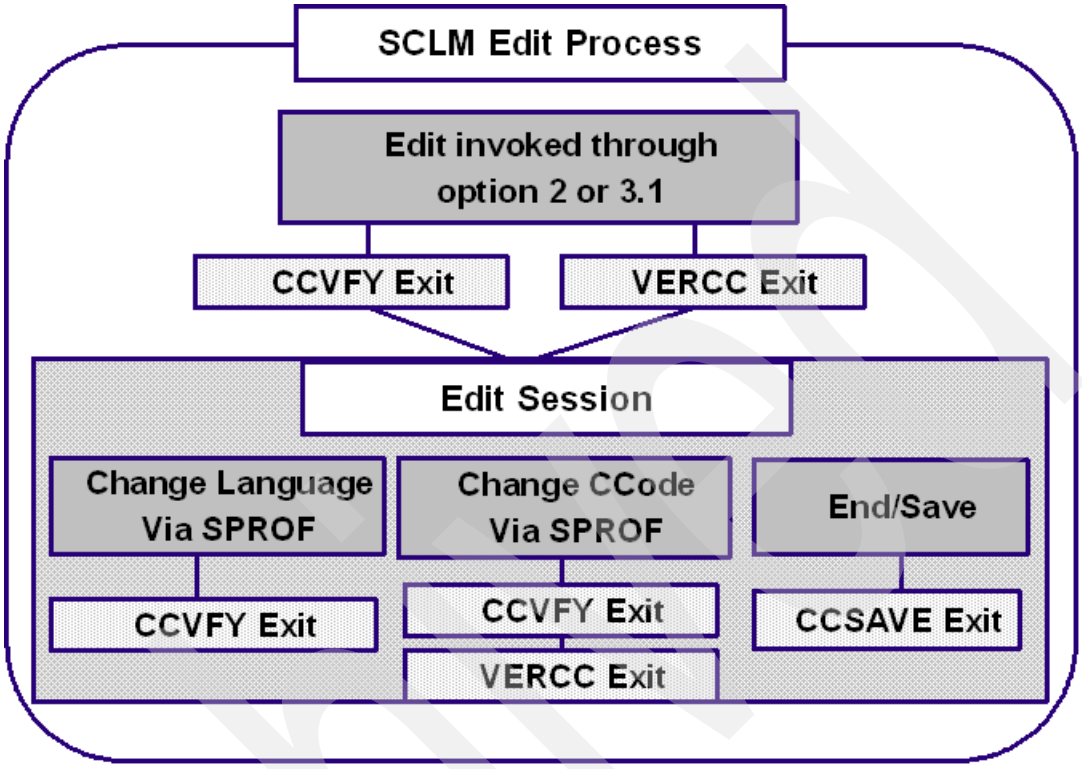


Figure 12-1 SCLM Edit User Exits

12.5.1 SCLM exits during an edit – verify change code exit

This exit is invoked at edit verification which includes panel processing via SCLM option 2 and option 3.1, Edit service and also **SPROF** processing. **SPROF** processing is used if either the language or change code has been modified. It is used to verify the change code that was entered. A non-zero return code from this exit will *stop processing*.

FLMCNTRL parameters

These are the parameters:

- CCVFY='name of program'
- CCVFYDS='dataset where program resides'
- CCVFYCM='program call method'
- CCVFYOP='exit options list'

This exit routine can be used to do the same functions as the VERCC exit. CCVFY was introduced in release 2.8 and replaces the first half of VERCC as shown in Example 12-5.

Example 12-5 Sample CCVFY exit

```
/* REXX */
   ARG parm
   /* Parse parms passed by SCLM into variables  */
   Parse var PARM GROUP ',' TYPE ',' MEMBER ',' LANG ',' ,
```

```
               USERID ',' AUTHCODE ',' CCODE
ADDRESS TSO "ALLOC DA('RDBK.SCLM.CCIDVAL') F(CCODEDS) SHR"
"EXECIO * DISKR "CCODEDS" (STEM ccline. FINIS)"
IF rc <> 0 THEN DO
   SAY 'ERROR READING CHANGE CODE FILE'
   EXIT (16)
End
Address TSO "FREE FI(CCODEDS)"

Do I = 1 To ccline.0
   Select
      When SUBSTR(ccline.I,1,8) = ccode  Then
         Exit (0)
      Otherwise
         Nop
   End
End
Say "Invalid change code"
exit 12
```

This sample REXX code is a CCVFY exit that verifies the change code that the user enters on the SCLM Edit - Entry panel or the SPROF panel. This REXX code allocates a change code file that is maintained outside of SCLM. Valid change codes are stored in column 1-8, one on each line. If the allocation is successful, then the REXX exec loops through the change code file to find a match on the SCLM change code. If a match is not found, then the exit will send a return code of 12 back to SCLM which will stop the edit session or will force the user to enter a valid change code on the SPROF panel.

12.5.2 SCLM exits during an edit – save change code exit

Once a member is saved, but prior to the accounting information update, the CCSAVE exit is available. This exit will be called when a save is done in *Edit* or *Migrate* and from the edit, store, save, and migrate services. A non-zero return code sent back to SCLM will stop processing.

This exit might be used to update text data like JCL, Procs, and PARMs in the development stage. If the testing for the development stage takes place on a separate LPAR or if test data needs to be placed in a file external to SCLM, then an exit could be written to transport the data to the external dataset. Only build outputs are listed in the *Build Notify Exit* file so “text” members do not show up; therefore, the *Build Notify Exit* could not be used to update text member in development stages to external sources.

FLMCNTRL parameters

These are the parameters:

- CCSAVE=‘name of program’
- CCSAVDS=‘dataset where program resides’
- CCSAVCM=‘program call method’
- CCSAVOP=‘exit options list’

12.5.3 SCLM exits during an edit – change code verification exit

This exit is called twice during the edit function in SCLM. The *first* time is before the edit is started. If a non-zero return code goes back to SCLM during this exit call, then it will be ignored by SCLM and processing will continue.

This exit is called the second time if the user changes the Change Code via the SPROF panel. This exit is also invoked by the migrate and import utilities, and edit, migrate, import, save and store services. It has the following characteristics:

- Present in SCLM just for backward compatibility; replaced by CCVFY
- Invoked at the Start of Edit/Change of Change Code/Migrate/Import
- Also invoked on Edit/Migrate/Import/Save and Store services
- Can be used to verify change code against an external file
- On End/Save, if VERCC exit point is present in the project definition, then the Change Code cannot be blank

FLMCNTRL parameters

These are the parameters:

- VERCC='name of program'
- VERCCDS='dataset where program resides'
- VERCCCM='program call method'
- VERCCOP='exit options list'

If the exit sends a non-zero return code to SCLM, then the user can do one of two things:

- Use the **CREATE** command in the edit session to place the member in an external SCLM library
- Use the **SPROF** command to modify the change code so that the exit will give a 0 return code

Note: The member cannot save in SCLM until the problem is corrected.

Given the problems with VERCC (that is, a non-zero return code is being ignored from the call to the exit at the start of the edit), IBM now recommends the use of the newer CCVFY and CCSAVE exits.

12.5.4 Edit exits warning

Both the CCVFY and VERCC exits are invoked prior to the edit session being started. If you use SCLM Edit, option 2, and do not put a member name in, then the exit is invoked. In this case a blank member name is passed through to the exit.

The reason for this is that SCLM is passing control to the ISPF editor, so the exit point is invoked at the last possible time that SCLM still has control.

If your exit processing requires the member name, then you may need some code that checks if the member is blank and if so issue a message and then deny the edit. By using SCLM option 3.1, a member name will generally always be passed through to the exit.

Archived

12.6 SCLM build exits

Figure 12-2 shows the SCLM build user exits.

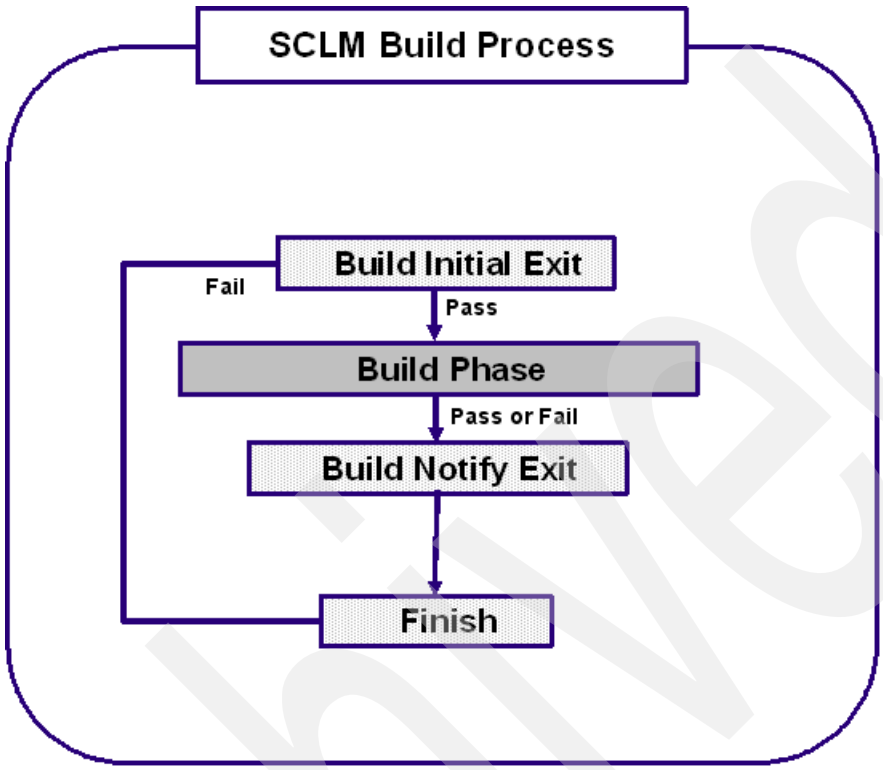


Figure 12-2 SCLM build user exits

As you can see, there are two build exits, a build initial exit that occurs before the build occurs, and a build notify exit that occurs after the build has completed.

12.6.1 SCLM build exits – build initial exit

This exit is invoked at the beginning of the build process, prior to the build verification. This exit should be used for any function that might have to be performed prior to the build process, or that would stop the build process. For example, if the hierarchy has many development stages, but these stages are utilized by specific user or departments, then an exit could be written that would allow only a certain user to build in that development stage. The EXITTYPE is BINITIAL.

FLMCNTRL parameters

These are the parameters:

- BLDINIT='name of program'
- BLDINIDS='dataset where program resides'
- BLDINICM='program call method'
- BLDINIOP='exit options list'

A build exit file does not exist when this exit is invoked. This means that specific build member validation cannot be completed. Any non-zero return code from this exit which is sent back to SCLM will *stop the build processing*. This exit is new for OS/390 version 2.10.

12.6.2 SCLM build exits – build notify exit

After all outputs from the build process have been produced, the BLDNTF exit is called.

This exit might be used to copy the outputs generated from SCLM to external files, or an exit could also be used to perform a CICS new copy. The EXITTYPE is BUILD.

FLMCNTRL parameters

These are the parameters:

- BLDNTF='name of program'
- BLDNTFDS='dataset where program resides'
- BLDNTFCM='program call method'
- BLDNTFOP='exit options list'

A non-zero return code from this exit which is sent back to SCLM will cause SCLM to finish with a return code of 4. At this point *all* processing has taken place, so SCLM will continue its processing. This exit has been given a new name for the OS/390 release 2.10. In the 2.10 release and thereafter the exit can be referred to as either **BLDNTF** or **BLDEXT1**.

12.6.3 SCLM build exits – build notify exit - example of the common REXX

Example 12-6 is an example of a common build notify exit. It calls two exits, one to copy members to external datasets where they are to be executed and a second similar exit for Java / J2EE™ development that copies jar files to their execution environment either on a local or remote USS directory. Both of these exits have been coded to run during the build notify and promote purge exits because the function is needed at each time. Whenever a new output is generated during a build or a output is promoted the external datasets have to be updated. The COPYEXTR exit will be documented in this section while the DEPLOY exit will be documented in the promote exit section.

Example 12-6 Common build exit

```
/* REXX */
/*********************************************************************/
/*  This exec will set up an ISPF environment for the execution of   */
/*  the build exits                                                  */
/*********************************************************************/
TRACE o
ARG parm                /* Parse arguments into variable parm */
bldntfrc = 0

ADDRESS ISPEXEC
"SELECT CMD(EX 'SCLM07.PROJDEFS.REXX(COPYEXTR)' '"parm"')"
bldntfrc = MAX(bldntfrc,rc)

"SELECT CMD(EX 'SCLM07.PROJDEFS.REXX(DEPLOY)' '"parm"')"
bldntfrc = MAX(bldntfrc,rc)

exit bldntfrc
```

12.6.4 SCLM build exits – build notify exit - example of a copy exit

In the following example of a copy exit, you can look at the comments in the code to see how the exit has been coded and how you can use this code in your exits. In particular, take a close look at how the parameters and exit file have been parsed and processed. This code can be common to any of your build exits. You can also see how certain SCLM types have been tested for in the exit file because only members in the exit file of these types have to be copied to external files. In this example the SELECT statement only processes members in types LOAD and CICSLOAD and sets up the external datasets for them. If you wanted to use this copy exit you would need to modify the SELECT for your types and external datasets accordingly. You could also extend the exit to perform other actions that may be necessary on a copy such as invoking a new copy in the CICS region or sending an e-mail notifying the administrator or user of the success or failure of the copy.

Example 12-7 is REXX code that shows how ISPF services can be incorporated into the REXX code to allocate and copy datasets. This program uses the ISPF **LMINIT** command to allocate the from and to datasets. Notice that variables from the PARM for project, to group, and type were used to determine the source dataset name. **LMCOPY** is used to perform the copy.

Example 12-7 Copy exit example

```
/* REXX */
/*********************************************************************/
/* This exec copies the executable members to their execution dataset.*/
/*********************************************************************/
trace o
ARG PARM
ret_code = 0
/* Parse arguments passed into the exit */
PARSE UPPER VAR parm extyp ',' proj ',' prjdf ',' tsouid ',',
      fromgrp ',' type ',' member ',' scope ',' mode ',' togrp
proj   = strip(proj)
type   = strip(type)
member = strip(member)

/* setup up the exit ddname and the 'good' status message to check for. */
/* for a build exit the from group indicates where the build is occurring */
/* for a promote exit we want to use the destination group for the source of*/
/* the copy since the promote will have occurred by the time the exit runs. */
if strip(extyp) = 'BUILD' then do
   extdd = 'BLDEXIT'
   goodmsg = 'BUILT'
   copygrp = strip(fromgrp)
   end
else do
   extdd = 'PROMEXIT'
   goodmsg = 'PURGE SUCCESSFUL'
   copygrp = strip(togrp)
   end
"execio * diskr " extdd "(stem extline. finis)"

do i = 1 to extline.0   /* For all lines in exit file                */
   /* Extract variables from a line out of the exit file */
   parse upper var extline.i eogroup 10 eotype 19 eomember 28 eostatus
   eogroup = STRIP(eogroup)
```

```
   eotype  = STRIP(eotype)
   eomember= STRIP(eomember)
   eostatus= STRIP(eostatus)

   /* If members status is OK then continue to process */
   If eostatus <> goodmsg then
      iterate

   /* To dataset for the copy is a pre-allocated library      */
   /* Modify the Select statement for your types and datasets */
   Select
      When eotype = 'LOAD' then do
         outdsn = "'RDBK."copygrp".LOADLIB'"
         Copy = 'Yes'
         end
      When eotype = 'CICSLOAD' then do
         outdsn = "'RDBK."copygrp".CICSLOAD'"
         Copy = 'Yes'
         end
      Otherwise
         Copy = 'No'
   End
   If Copy = 'Yes' then call DoCopy
End

exit ret_Code

DoCopy:
/* Initialize the dataset to copy from */
indsn  = "'"proj"."copygrp"."eotype"'"
Address ISPEXEC "LMINIT DATAID(FROMDSN) DATASET("indsn")"
ret_code= rc
if ret_code <> 0 then do
   Say 'Error on LMINIT for dataset 'indsn' return code' rc
   exit 32
   end

Address ISPEXEC "LMINIT DATAID(TODSN) DATASET("outdsn")"
ret_code= rc
if ret_code <> 0 then do
   Say 'Error on LMINIT for dataset 'outdsn' return code' rc
   exit 32
   end

/* Copy member from SCLM prod to external dataset */
Address ISPEXEC "LMCOPY FROMID("fromdsn") FROMMEM("eomember")
                 TODATAID("todsn") TOMEM("eomember") REPLACE"
If rc <> 0 then do /* If error on the Copy */
   Say 'Error on LMCOPY for member 'eomember' return code' rc
   exit 32
   end
else       /* Member was copied successfully */
   Say eomember' has been copied to 'outdsn
Return
```

Archived

12.7 SCLM promote exits

Figure 12-3 shows the SCLM promote exits.

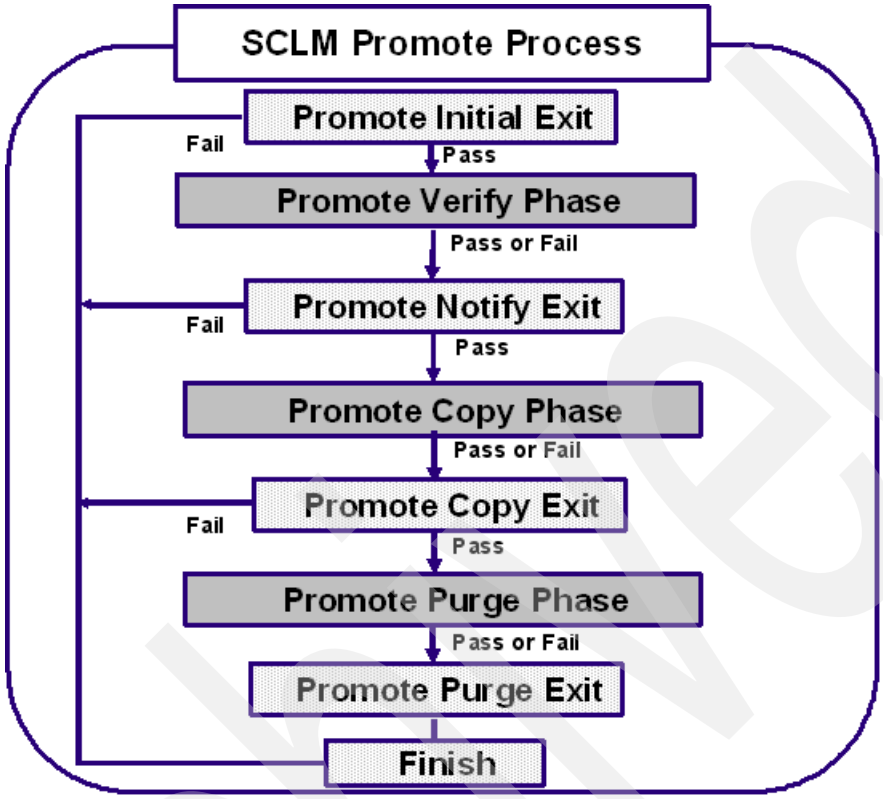


Figure 12-3 SCLM promote exits

As you can see, SCLM has four promote exits, the Promote Initial exit, which occurs before the verify phase; the promote notify exit, which occurs after the verification is complete; the promote copy exit, which occurs after the copies have occurred; and the promote purge exit, which occurs after the purge phase.

12.7.1 SCLM promote exits – promote initial exit

This exit is accessed at the beginning of the promote process prior to verification. This exit might be used for any function that might have to be performed prior to the promote process or any verification that would need the promote process to stop. For example, the exit might interface with a change control system to check for an authorization of the promote. The EXITTYPE is PINITIAL.

FLMCNTRL parameters

These are the parameters:

- PRMINIT=‘name of program’
- PRMINIDS=‘dataset where program resides’
- PRMINICM=‘program call method’
- PRMINIOP=‘exit options list’

The promote exit file does not exist when this exit is invoked. This means that a promote of an ARCHDEF would not have specific member information during this exit process. Any non-zero return code from this exit which is sent back to SCLM will *stop the promote process*. This exit is new for OS/390 version 2.10.

12.7.2 SCLM promote exits – promote verify exit

PRMVFY is invoked after the verification phase of the promote process (prior to *promote copy* and *promote purge*). PRMVFY might be used to backup the outputs in the group being promoted prior to the *promote copy* phase. You could also use this exit to check the promote TO stage to see if this member is already present at the TO stage and if the member already exists, then stop the promote. The EXITTYPE is PVERIFY.

FLMCNTRL parameters

These are the parameters:

- PRMVFY='name of program'
- PRMVFYDS='dataset where program resides'
- PRMVFYCM='program call method'
- PRMVFYOP='exit options list'

Depending on the mode specified on the promote, the return code could affect the promote processing. If conditional mode is used, then any return code greater than 4 will stop all the promote processing. If *unconditional* mode is used, then any return code other than 20 will allow processing to continue. This exit has been given a new name for the OS/390 release 2.10. In the 2.10 release and thereafter the exit can be referred to as either **PRMVFY** or **PRMEXT1**.

12.7.3 SCLM promote exits – promote verify exit - example of common REXX

Example 12-8 is an example of a common promote verify exit. It calls one exit that backs up members before they are overlaid during a promote. It also verifies that the promote is being performed only using a PACKAGE type. This exit is used on a system that is using Breeze to control the promote and all Breeze promotes must be performed using a high level ARCHDEF and on this system the high level ARCHDEFs are stored in the PACKAGE type. This is an example of coding a small amount of logic directly into the common REXX. The logic could be performed in a called REXX program, but it does not make sense to do so since it consists of only a few lines of REXX. Notice though that the passed parameters must be parsed to find the member type that is being promoted.

Example 12-8 Promote Verify Common exit

```
/* REXX */
/*********************************************************************/
/*  This exec will set up an ISPF environment for the execution of   */
/*  the promote verify exit.                                         */
/*********************************************************************/
TRACE o
ARG parm                  /* Parse arguments into variable parm */
CALL Init
prmntfrc = 0

if type <> 'PACKAGE' then do
   say 'All promotes must use the PACKAGE type'
   exit 20
   end
```

```
ADDRESS ISPEXEC
"SELECT CMD(EX 'SCLM07.PROJDEFS.REXX(BKUPPROD)' '"parm"')"
prmntfrc = MAX(prmntfrc,rc)

exit prmntfrc
/*********************************************************************/
/* Init:     Subroutine to initialize variables, examine the parms  */
/*           passed from SCLM, and set additional variables.        */
/*********************************************************************/
Init:
   /* Break out the parms passed from the SCLM user exit */
   PARSE UPPER VAR parm extyp ',' proj ',' prjdf ',' tsouid ',',
         fromgrp ',' type ',' member ',' scope ',' mode ',' togrp
   extyp    = STRIP(extyp,'T')    /* Exit Type - where in SCLM?     */
   proj     = STRIP(proj,'T')     /* Project - high level qualifier */
   prjdf    = STRIP(prjdf,'T')    /* Alternate Project Definition   */
   tsouid   = STRIP(tsouid,'T')   /* User executing SCLM function   */
   fromgrp  = STRIP(fromgrp,'T')  /* From grp(Promote) or build grp */
   type     = STRIP(type,'T')     /* Type from the SCLM panel       */
   member   = STRIP(member,'T')   /* Member from the SCLM panel     */
   scope    = STRIP(scope,'T')    /* Scope entered on SCLM panel    */
   mode     = STRIP(mode,'T')     /* Mode entered on SCLM panel     */
   togrp    = STRIP(togrp,'T')    /* To grp (Promote), N/A (Build)  */
return
```

12.7.4 SCLM promote exits – promote verify exit - example of a backup exit

Example 12-9 is an example of a promote verify exit. It backs up the current members in production prior to them being overlaid during the promote. The SCLM backout function can perform a similar function, but it only handles outputs. This exit handles all types.

Example 12-9 Promote verify backup exit

```
/* REXX */
/*********************************************************************/
/*                                                                   */
/* This rexx exec copies production members into a backup library    */
/* before the SCLM Promote overlays them with the new members being  */
/* promoted into the PROD group.                                     */
/*                                                                   */
/*********************************************************************/
trace o                        /* o - off r - result i - intermediate */
arg parm

call init

/* If the promote is done in report mode then don't execute          */
if mode = 'REPORT' then
   exit

/* If not invoked from Promote Verify exit, then exit                */
if extyp <> 'PVERIFY' then
   exit
```

```
/* If not promoting into PROD group, then exit                        */
if togrp <> prdgrp then
   exit
/* If promoting from the PACKAGE BACKOUT group (BACK) then exit */
if fromgrp = 'BACK' then
   exit

/* Read all lines from the exit file into a stem variable */
"execio * diskr "ddname" (stem extline. finis)"

if extline.0 <> 0 then  /* if file is not empty                       */
   do i = 1 to extline.0   /* For all lines in stem variable */
      call Process
   end

exit

/*********************************************************************/
/* Init:     Subroutine to initialize variables, examine the parms  */
/*           passed from SCLM, and set additional variables.        */
/*********************************************************************/
Init:
   /* setup the group to do the backup for */
   prdgrp   = 'PROD'
   /* setup the type of members to backup */
   copytype = 'ARCHDEF CCDEF LECDEF PACKAGE COPYBOOK CICSLOAD DCLGEN',
   'DBRMLIB CNTLLIB PARMLIB PROCLIB JCLLIB LOAD ASM COBOL'

  /* Break out the parms passed from the SCLM user exit */
    PARSE UPPER VAR parm extyp ',' proj ',' prjdf ',' tsouid ',',
          fromgrp ',' type ',' member ',' scope ',' mode ',' togrp
    extyp    = STRIP(extyp,'T')     /* Exit Type - where in SCLM? */
    proj     = STRIP(proj,'T')      /* Project - high level qualifier */
    prjdf    = STRIP(prjdf,'T')     /* Alternate Project Definition */
    tsouid   = STRIP(tsouid,'T')    /* User executing SCLM function */
    fromgrp  = STRIP(fromgrp,'T')   /* From grp(Promote) or build grp */
    type     = STRIP(type,'T')      /* Type from the SCLM panel */
    member   = STRIP(member,'T')    /* Member from the SCLM panel */
    scope    = STRIP(scope,'T')     /* Scope entered on SCLM panel */
    mode     = STRIP(mode,'T')      /* Mode entered on SCLM panel */
    togrp    = STRIP(togrp,'T')     /* To grp (Promote), N/A (Build) */

    goodmsg  = 'PROMOTABLE'
    badmsg   = 'NOT PROMOTABLE'
    ddname   = 'PROMEXIT'
Return

/*********************************************************************/
/* Process:  Subroutine to initialize the from and to datasets,     */
/*           perform the copy of certain types of members to backup */
/*           libraries, and free the files for the next call.       */
/*********************************************************************/
Process:
/* Extract variables from a line out of the exit file */
parse upper var extline.i eogroup 10 eotype 19 eomember 28 eostatus
```

Archived

```
eogroup = STRIP(eogroup)
eotype  = STRIP(eotype)
eomember= STRIP(eomember)
eostatus= STRIP(eostatus)

/* Only backup members that SCLM has marked as Promotable */
if eostatus <> goodmsg then return

/* Only backup members with the types we want */
if wordpos(eotype, copytype) = 0 then return

/* From dataset for the backup is an SCLM PROD library */
indsn  = "'"proj"."prdgrp"."eotype"'"

address ispexec "lminit dataid(fromdsn) dataset("indsn")"
if rc <> 0 then do
   say 'Error on LMINIT for dataset 'indsn' return code' rc
   exit 32
   end

/* To dataset for the backup is a pre-allocated library */
outdsn = "'"proj"."prdgrp"."eotype".BACKUP'"

/* Initialize the dataset to copy into */
address ispexec "lminit dataid(todsn) dataset("outdsn")"
if rc <> 0 then do
   say 'Error on LMINIT for dataset 'outdsn' return code' rc
   exit 32
   end

/* Copy member from SCLM prod into backup dataset */
address ispexec "lmcopy fromid("fromdsn") frommem("eomember")
     todataid("todsn") tomem("eomember") replace"

if rc = 8 then     /* If member is new, and not in SCLM Prod */
   nop             /* No copy was done, not an error */
else
   if rc <> 0 then do /* If error on the Copy */
      say left('* Error Copying member 'eomember, 77) '*' rc
      say left('* from 'indsn, 77) '*'
      say left('* to   'outdsn, 77) '*'
      exit 32
      end
   else do    /* Member was copied successfilly */
      say ' '
      say eomember' has been backed up to 'outdsn
      end

address ispexec "lmfree dataid("todsn")"
address ispexec "lmfree dataid("fromdsn")"
return
```

12.7.5 SCLM promote exits – promote copy exit

When the promote members have been copied to the next group, but prior to them being removed from the current group, the PRMCOPY exit will be called. This exit might be used to stage a batch promote process. An *intermediate* stage would be placed in the hierarchy before the production stage. When the promote to the intermediate stage occurs this exit could write a record to a file. A batch job could then be scheduled to read the file and process the promotes to production. This exit could also be used to copy the outputs from the SCLM output datasets to external datasets or to notify a user of the promotion. The EXITTYPE is PCOPY.

FLMCNTRL parameters

These are the parameters:

- PRMCOPY='name of program'
- PRMCPYDS='dataset where program resides'
- PRMCPYCM='program call method'
- PRMCPYOP='exit options list'

If a return code of 20 is sent back to SCLM, then *all promote processing will stop*.

This exit has been given a new name for the OS/390 release 2.10. In the 2.10 release and thereafter the exit can be referred to as either **PRMCOPY** or **PRMEXT2**.

12.7.6 SCLM promote exits – promote purge exit

This exit is invoked after **promote verification**, **promote copy**, and **promote purge** have occurred.

This exit might be used to clear out the *emergency* stage of a hierarchy by using the Delete Group SCLM service. This would delete all associated inputs and outputs for a same named member in the emergency group when the member is promoted to the production stage. This deletion will allow the new load module in production to run without an override from an emergency library which is concatenated on top. This exit might also be used to copy members to external libraries where they are to be executed.The EXITTYPE is PPURGE.

FLMCNTRL parameters

These are the parameters:

- PRMPURGE='name of program'
- PRMPRGDS='dataset where program resides'
- PRMPRGCM='program call method'
- PRMPRGOP='exit options list'

A non-zero return code sent back to SCLM will cause SCLM to finish with a return code of 8. At this point all processing has taken place, so SCLM will continue its processing. This exit has been given a new name for the OS/390 release 2.10. In the 2.10 release the exit can be referred to as either **PRMPURGE** or **PRMEXT3**.

12.7.7 SCLM promote exits – promote purge exit - example of common REXX

Example 12-10 is an example of a common promote purge exit. It calls two exits, one to copy members to external datasets where they are to be executed and a second similar exit for Java / J2EE development that copies jar files to their execution environment either on a local or remote USS directory. Both of these exits have been coded to run during the build notify

Archived

and promote purge exits because the function is needed at each time. Whenever a new output is generated during a build or a output is promoted the external datasets have to be updated. The COPYEXTR exit was documented in the build exit section while the DEPLOY exit will be documented below.

Example 12-10 Common Purge Exit

```
/* REXX */
/*********************************************************************/
/*  This exec will set up an ISPF environment for the execution of   */
/*  the build exits                                                  */
/*********************************************************************/
TRACE o
ARG parm                 /* Parse arguments into variable parm */
CALL Init
prmprgrc = 0
ADDRESS ISPEXEC
"SELECT CMD(EX 'SCLM07.PROJDEFS.REXX(COPYEXTR)' '"parm"')"
prmprgrc = MAX(prmprgrc,rc)

"SELECT CMD(EX 'SCLM07.PROJDEFS.REXX(DEPLOY)' '"parm"')"
prmprgrc = MAX(prmprgrc,rc)

exit prmprgrc
```

12.7.8 SCLM promote exits – promote purge exit - example of a deploy exit

Example 12-11 is an exit that copies jar files to a USS directory where they are to be executed from. This exit would be used for a SCLM project that is using the SCLM Developer Toolkit to handle java code and create class and jar files as output. The output will be stored in SCLM but will have to be copied to a USS directory for execution. The copy in this example is performed with a OPUT command when the directory is mounted on the same LPAR where SCLM is executing and uses FTP for a remote deploy. The destination directories and the userid and password for the FTP process must be setup in a file. Look in the read_ip routine in the exit below for some information on how to set this file up.

Example 12-11 Promote purge exit example for deployment

```
/* REXX */
/*********************************************************************/
/*                                                                   */
/* This rexx exec copies jar files to a USS directory via a OPUT     */
/* copy below PROD or via FTP to a remote LPAR for PROD.  It looks   */
/* up the long name of the member to find the correct USS name to    */
/* copy to.                                                          */
/*                                                                   */
/*********************************************************************/
trace o
Address TSO
  Arg PARM

  parse var parm  vexttype ","project","libdef","tsouid","fromgrp","
                  type","prommbr","scope","mode","togrp

  Pmemlist. = 0
  deplloc.  = ''
```

```
  longname. = ''

  fromgrp   = Strip(fromgrp)
  togrp     = Strip(togrp)
  Exit_rc = 0

  if vexttype = 'BUILD' then
     srchgrp = fromgrp
  else
     srchgrp = togrp

  /* read exit file to find members being promoted              */
  call get_mem_being_promoted

  /* read destination information for each group                */
  call read_ip

  /* if a match is found in the destination info then do the    */
  /* type of copy specified for the destination (local or FTP)  */
  do k = 1 to ipaddrs.0
     if groupto.k = srchgrp then do
        count = 0
        if ftphfs.k = 'FTP' then
           call Do_FTP
        else
           call Do_Copy
        if count > 0 then
           /* if any members have been copied send a email       */
           call Send_Email
        end
  end

Exit (Exit_rc);

/*--------------------------------------------------------------------*/
/* This reads the exit output dataset for members being built      */
/*--------------------------------------------------------------------*/
Get_mem_being_promoted:

  If VEXTTYPE = 'BUILD' Then
     "execio * diskr BLDEXIT (stem exitfile. finis)"
  else
     "execio * diskr PROMEXIT (stem exitfile. finis)"

  if rc \= 0 Then return

  do i = 1 to exitfile.0
     This_line = exitfile.I
     This_line = TRANSLATE(this_line)
     Parse var This_line Group Type Member Status
     Group.i  = Strip(Group)
     Type.i   = Strip(Type)
     Member.i = Strip(Member)
     Status.i = Strip(Status)
  End
```

Archived

```
Return;

/*-------------------------------------------------------------------*/
/* Read info for destination                                         */
/*-------------------------------------------------------------------*/
read_ip:
  Address TSO
  /* alloc long name file used by the developer toolkit              */
  "FREE FI(LSTRANS)"
  "FREE FI(LSTRNPTH)"
  "ALLOC DD(LSTRANS)   DA('BWB.LSTRANS.FILE') SHR REUSE"
  "ALLOC DD(LSTRNPTH)  DA('BWB.LSTRANS.FILE.PATH') SHR REUSE"
  /* Read destination info into Stem variables.  To use this        */
  /* this exit you will need to create your own ip files.           */
  /* See the do loop below for the layout of the file.  It is       */
  /* space delimited.  This dataset will need to be protected       */
  /* since it contains passwords.  There can be multiple            */
  /* destinations for a group.                                      */
  Address TSO "ALLOC F(IPADDRS) ",
            "DA('SCLM07.PROJDEFS.SOURCE(IPADDRS)') SHR"
  "EXECIO * DISKR IPADDRS (FINIS STEM ipaddrs."
  /* Here is an example dataset                                     */
  /* group (FTP/local) (dest ip/local) (dest USS)  perm userid pw   */
  /* DEV1  local   local       /u/pceck/RDBK/dev1/ 777  na     na   */
  /* TEST  local   local       /u/pceck/RDBK/test/ 777  na     na   */
  /* PROD  FTP     abc.ibm.com /u/pceck/RDBK/prod/ 777  pceck  abcd */

  j = 0
  do i = 1 to ipaddrs.0
     Parse var ipaddrs.i groupto ftphfs ip_addr path mode userid password
     if groupto = srchgrp then do
        j = j + 1
        groupto.j  = Strip(groupto) /* SCLM group to process the copy */
        ftphfs.j   = Strip(ftphfs)  /* hard code 'FTP' or 'local'     */
        ip_addr.j  = Strip(ip_addr) /* ip address of dest or 'local'  */
        path.j     = Strip(path)    /* USS directory to copy to       */
        mode.j     = Strip(mode)    /* permission to set member to    */
        userid.j   = Strip(userid)  /* userid to use for the FTP      */
        password.j = Strip(password) /* password to use for the FTP   */
        end
  end
  ip_count = j /* count of the destinations for the group */
Return;

/*-------------------------------------------------------------------*/
/* Perform the FTP of the files                                      */
/*-------------------------------------------------------------------*/
Do_FTP:
  /* For each different IP Address FTP the files and CHMOD */
  do J = 1 to ip_count
     call Init_FTP
     Do I = 1 to exitfile.0
        if Type.i = 'J2EEJAR' Then do
           FromDSN = Strip(PROJECT)'.'Strip(srchgrp)'.'Type.i
           /* find longname for shortname */
```

```
            Say "Find longname for shortname"
            FLMLSSHR = member.i
            Address ISPEXEC "VPUT (FLMLSSHR) PROFILE"
            Address ISPEXEC "SELECT PGM(FLMLSTRN) PARM(FINDLONG)"
            LSRC=RC
            If LSRC > 0 Then do
               Address ISPEXEC "VGET (FLMLSERR,FLMLSER1) PROFILE"
               Say "LS ERROR LINE 1 ==>" FLMLSERR
               Say "LS ERROR LINE 2 ==>" FLMLSER1
               Return
               End
            Else do
               Address ISPEXEC "VGET (FLMLSSHR,FLMLSLNG) PROFILE"
               Longname.i = FLMLSLNG
               deplloc.i = path.j || longname.i
               F = F + 1
               ftpcmd.F = "Put '"FromDSN"("Member.i")'" deplloc.i
               F = F + 1
               ftpcmd.F = "Site chmod "mode.j" "deplloc.i
               count = count + 1
               End
            end

         /* these lines were put in for testing of one additional type*/
         /* it is not very useful but it does show you can extend this*/
         /* REXX for any type of member                                */
         if Type.i = 'LMAP' Then do
            FromDSN = Strip(PROJECT)'.'Strip(srchgrp)'.'Type.i
            longname.i = member.i
            deplloc.i = path.j || member.i || '.lmap'
            F = F + 1
            ftpcmd.F = "Put '"FromDSN"("Member.i")'" deplloc.i
            F = F + 1
            ftpcmd.F = "Site chmod "mode.j" "deplloc.i
            count = count + 1
            end
      end

      if F > 4 Then do
         /* if any lines have been added to the FTP file after the init*/
         F = F + 1
         ftpcmd.F  = 'quit'
         ftpcmd.0  = F
         Call Invoke_FTP
         End
   end

Return(0);

/*-------------------------------------------------------------------*/
/* Perform the local copy                                            */
/*-------------------------------------------------------------------*/
Do_Copy:
  /* For each different destination copy the files and CHMOD */
  do J = 1 to ip_count
```

```
    do I = 1 to exitfile.0
      if Type.i = 'J2EEJAR' Then do
        FromDSN = Strip(PROJECT)'.'Strip(srchgrp)'.'Type.i
        /* find longname for shortname */
        Say "Find longname for shortname"
        FLMLSSHR = member.i
        Address ISPEXEC "VPUT (FLMLSSHR) PROFILE"
        Address ISPEXEC "SELECT PGM(FLMLSTRN) PARM(FINDLONG)"
        LSRC=RC
        if LSRC > 0 Then do
          Address ISPEXEC "VGET (FLMLSERR,FLMLSER1) PROFILE"
          Say "LS ERROR LINE 1 ==>" FLMLSERR
          Say "LS ERROR LINE 2 ==>" FLMLSER1
          Return
          End
        else do
          Address ISPEXEC "VGET (FLMLSSHR,FLMLSLNG) PROFILE"
          Longname.i = FLMLSLNG
          deplloc.i = path.j || longname.i
          "OPUT '"FromDSN"("Member.i")' '"deplloc.i"'"
          say "OPUT "FromDSN"("Member.i")' to '"deplloc.i"' rc= " rc
          chmod    = deplloc.i mode.j
          Call syscalls 'ON'
          Address syscall 'chmod' chmod
          Say 'chmod' chmod 'rc=' rc
          count = count + 1
          end
        end

      if Type.i = 'LMAP' Then do
        /* these lines were put in for testing of one additional type*/
        /* it is not very useful but it does show you can extend this*/
        /* REXX for any type of member                                */
        FromDSN = Strip(PROJECT)'.'Strip(srchgrp)'.'Type.i
        longname.i = member.i
        deplloc.i = path.j || member.i || '.lmap'
        "OPUT '"FromDSN"("Member.i")' '"deplloc.i"'"
        say "OPUT "FromDSN"("Member.i")' to '"deplloc.i"' rc= " rc
        chmod    = deplloc.i mode.j
        Call syscalls 'ON'
        Address syscall 'chmod' chmod
        Say 'chmod' chmod 'rc=' rc
        count = count + 1
        end
     end
    end

Return(0);

/*-------------------------------------------------------------------*/
/* This reads initialises FTP variables and stores the DUMMYSMP    */
/* into memory for processing                                        */
/*-------------------------------------------------------------------*/
Init_FTP:
  remote_host = ip_addr.j
```

```
  login_id    = userid.j
  pass_word   = password.j
  /*-------------------------------------------------------------*/
  /* Setup the FTP subcommands to be issued...                   */
  /*-------------------------------------------------------------*/
  ftpcmd.1  = remote_host;
  ftpcmd.2  = login_id pass_word  /* USER and PASS responses  */
  ftpcmd.3  = "cd  "path.j
  ftpcmd.4  = "binary"

  F = 4          /* FTP Line count   */
  C = 0          /* CHMOD line count */

Return

/*-------------------------------------------------------------------*/
/* This allocates files and invokes FTP                              */
/*-------------------------------------------------------------------*/
Invoke_FTP :
  address TSO 'Allocate fi(INPUT) new space(30,30) tracks ',
              'dsorg(ps) lrecl(2080) recfm(v,b) blksize(0) reuse'

  address TSO 'Allocate fi(OUTPUT) new space(30,30) tracks ',
              'dsorg(ps) lrecl(160) recfm(f,b,a) blksize(0) reuse'

  "execio" ftpcmd.0 "diskw INPUT (stem ftpcmd. finis"

  "FTP (EXIT"

  FTP_rc = rc

  if (FTP_rc <> 0) then do
     Say 'FTP failed. RC='FTP_rc

     Address TSO "execio * diskr OUTPUT (stem sysprint. finis"

     If (rc <> 0) then
        Say 'Read failure on SYSPRINT dataset.  Return Code='rc
     else
        Do i = 1 to sysprint.0
           sysprint.i = strip(sysprint.i,'T',' ')
           say sysprint.i
        End
      Exit_rc = 12
      end
  else
      Say 'FTP worked. RC='FTP_rc

  Address TSO 'Free fi(INPUT)'
  Address TSO 'Free fi(OUTPUT)'
Return

Send_Email:
  sysid = MVSVAR('sysname')
```

Archived

```
/* Date routine from SMTPNOTE */
date = date("n")" "time()" AWST"
date = subword(date,1,2)                || ,   /* dd mon */
       " "                              || ,
       strip(substr(word(date,3),3,4)) || ,   /* yy */
       " "                              || ,
       subword(date,4)                        /* time and zone  */
'MAKEBUF'
atsign = '@'
queue 'helo PTHAPCO.AU.IBM.COM'
queue 'mail from:<'strip(tsouid)||atsign||'PTHAPCO>'
queue 'rcpt to:<user1'||atsign||'us.ibm.com>'
queue 'data'
queue 'Date:    'date
queue 'From:    'strip(tsouid)||atsign||'PTHAPCO'
queue 'To:      user1'||atsign||'us.ibm.com'
queue 'Cc:      user2'||atsign||'us.ibm.com'
queue 'Subject: SCLM Suite HFS Copy Report 'date' 'sysid' 'PROMMBR
queue ' '
queue 'The following objects have been copied to the HFS on system 'sysid
queue 'Project : 'PROJECT 'From group : 'srchgrp
queue 'Using work element : 'PROMMBR
queue 'By User : 'TSOUID
queue ' '
Do i = 1 to exitfile.0
   If deplloc.i <> '' Then
      queue Member.i '('longname.i') of type' Type.i 'to' deplloc.i
End
queue '.'
queue ''
"ALLOC F(SMTPMSG) SYSOUT(B) LRECL(255) RECFM(V B) DSORG(PS) "||,
      "DEST(PTHAPDO) WRITER(SMTP)"
If Rc = 0 Then
Do
   "execio * diskw smtpmsg (finis"
   "FREE F(SMTPMSG)"
End
'DROPBUF'

Return
```

Archived

12.8 SCLM delete exits

Figure 12-4 shows the SCLM delete user exits.

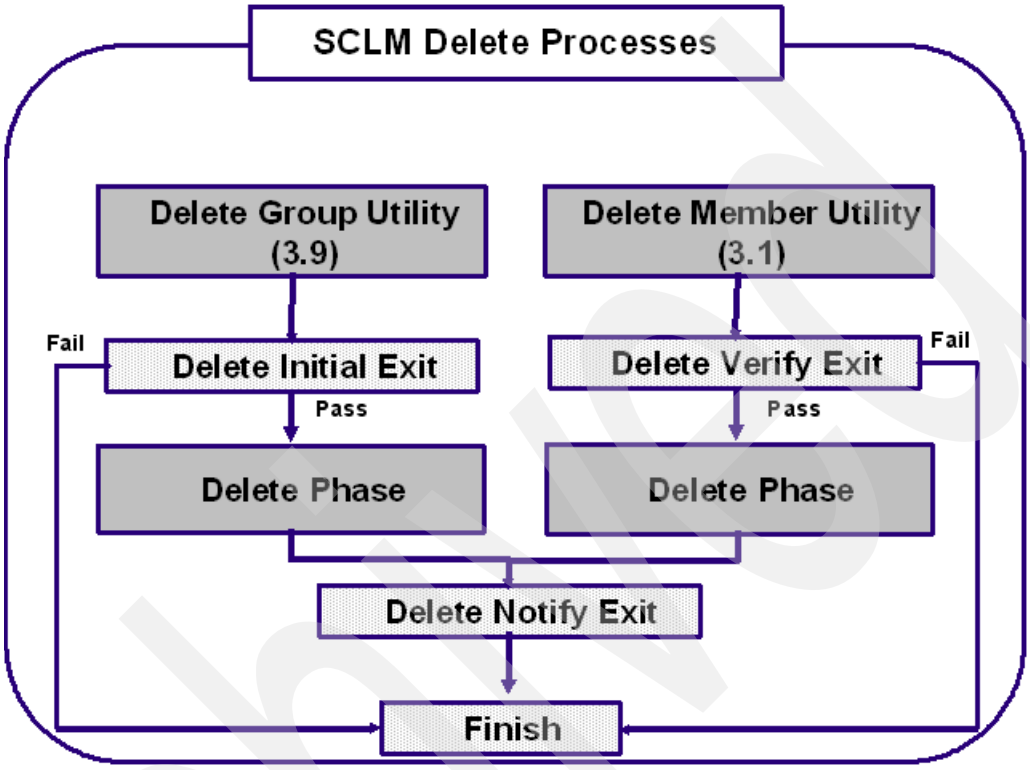


Figure 12-4 Delete user exits

12.8.1 SCLM delete exits – delete initial exit

This exit is called for the Delete Group dialog or service. The exit is invoked at the beginning of the deletion process (during initialization). When using the Delete Group dialog or service, a wild card can be used for *group*, *type*, and *member name*.

This exit might be used to check a userid against the group or type to ensure that the user has access to delete. Some companies do not allow deleting from production without special authorization. This exit could interface with the change control system to validate the authorization. The EXITTYPE is DGINIT.

FLMCNTRL parameters

These are the parameters:

- DELINIT='name of program'
- DELINIDS='dataset where program resides'
- DELINICM='program call method'
- DELINIOP='exit options list'

Any return code other than 0 will *stop the delete processing*. This exit is new for OS/390 version 2.10.

12.8.2 SCLM delete exits – delete initial exit - example

Example 12-12 is an example of a delete init exit that will be used to stop the deletion of members by other than those who changed it last. An admin can delete any member. The dbutil service is used to find all members matching delete group delete criteria in case wildcards are used for group or type. The dbutil must be used because the delete exit file is not available in the delete initial exit. The dbutil is also used to find the user id of the person who changed it last.

Example 12-12 Delete initial exit example

```
/* REXX */
/*********************************************************************/
/*                                                                   */
/*  This exec called from the Delete Init exit will stop the delete  */
/*  if the user is not the owner of the member or is not a           */
/*  administrator.                                                   */
/*                                                                   */
/*********************************************************************/
TRACE o
ARG parm                 /* Parse arguments into variable parm */

CALL Init

/* No need to execute if run in report mode                         */
If mode = 'REPORT' then exit 0

/* No need to execute if source member is not being deleted         */
If FLAG <> 'TEXT' then exit 0

/* edit with the ids that can delete all code */
sclmadm = 'admext1 admext2'

call get_members

do i = 1 to rptline.0
   parse upper var rptline.i dgrp dtype dmbr duser
   if pos(dtype,testtypes) = 0 then iterate
   else do
      if (duser = tsouid) | (pos(tsouid,sclmadm) > 0) then
         say 'Member 'dmbr' can be deleted from group 'dgrp' type 'dtype
      else do
         say 'Can not delete member 'dmbr' in group 'dgrp' type 'dtype
         say 'Member is owned by 'strip(duser)
         exit 0020
         end
      end
end

Exit
/*********************************************************************/
/* Init:     Subroutine to initialize variables, examine the parms  */
/*           passed from SCLM, and set additional variables.        */
/*********************************************************************/
Init:
   /* Break out the parms passed from the SCLM user exit */
```

```
   PARSE UPPER VAR parm extyp ',' proj ',' prjdf ',' tsouid ',',
         group ',' type ',' member ',' flag ',' mode

   extyp    = STRIP(extyp,'T')    /* Exit Type                      */
   proj     = STRIP(proj,'T')     /* Project Definition             */
   prjdf    = STRIP(prjdf,'T')    /* Alternate Project Definition   */
   tsouid   = STRIP(tsouid,'T')   /* User executing SCLM function   */
   group    = STRIP(group,'T')    /* Group member is in             */
   type     = STRIP(type,'T')     /* Type from the SCLM panel       */
   member   = STRIP(member,'T')   /* Member from the SCLM panel     */
   flag     = STRIP(flag,'T')     /* Flag                           */
   mode     = STRIP(mode,'T')     /* Mode                           */

   /* modify with list of types you want ownership checked for.     */
   /* Add in output types if you don't want outputs deleted by      */
   /* other than user who built them.                               */
   testtypes = 'CNTLLIB COPYLIB COBOL DCLGEN JCLLIB PROCLIB SOURCE'
 RETURN

/*********************************************************************/
/* Get the list of members to be deleted                             */
/*********************************************************************/
get_members:

"alloc file("dummydd") dummy reuse"
"alloc file("rptdd") recfm(v b) lrecl(133) blksize(15000) space(5 5)",
        "tracks reuse"
/*
"alloc file("flmmsgs") dummy reuse" */

/* Invoke the SCLM Database Contents Utility (SCLM 3.4) to report on  */
/* all accounting records for specified group, type and change user.  */
"FLMCMD DBUTIL,"proj","prjdf","group",,,,,,"type","member",*,"||,
"*,*,*,*,YES,ACCT,*,,,,NORMAL,N,N,,"dummydd","dummydd","rptdd"," ||,
"@@FLMGRP @@FLMTYP @@FLMMBR @@FLMCUS"
saverc = rc


if saverc > 0  then do
   say "DBUTIL return code = " rc
   exit 0030
   end

"EXECIO * DISKR "rptdd" (STEM rptline. FINIS)"

if rc <> 0 then do
   say "Error reading EXIT output, return code = " rc
   exit 0040
   end

"FREE FILE("dummydd rptdd ")"
return
```

Archived

12.8.3 SCLM delete exits – delete verify exit

This exit point is utilized for the Library Utility Delete (ISPF Option 10.3.1) and delete service. It is called after the delete verification but prior to the actual delete. This exit might be used to check a userid against the group or type to ensure that the user has access to delete. Some companies do not allow deletion from production without special authorization. This exit could interface with the change control system to validate the authorization. The **DELINIT** and **DELVFY** perform the same kind of functions. The **DELINIT** is called for the delete group and **DELVFY** is called for the delete (ISPF Option 10.3.1). If you want to make sure that the verification for deletions are performed for all deletes, then the same verification will have to be in the **DELINIT** and **DELVFY** exits. The EXITTYPE is DVERIFY.

FLMCNTRL parameters

These are the parameters:

- DELVFY=‘name of program’
- DELVFYDS=‘dataset where program resides’
- DELVFYCM=‘program call method’
- DELVFYOP=‘exit options list’

Any return code other than 0 will *stop the delete processing*. This exit is new for OS/390 version 2.10.

12.8.4 SCLM delete exits – delete verify exit - example

Example 12-13 is an example of a delete verify exit. It is similar to the delete initial exit above, but since it is called from the DELETE service it only needs to check a single member. The ACCTINFO service is used to find the last change user id, since it runs quicker than the DBUTIL service.

Example 12-13 Delete verify exit example

```
/* REXX */
/*********************************************************************/
/*                                                                   */
/* This exec allows only the person who last changed a member or a   */
/* administrator to delete the member.  This exit is invoked from    */
/* Library Utility Delete (ISPF Option 10.3.1) or the delete service. */
/*                                                                   */
/*********************************************************************/
trace o
ARG parm                        /* parse arguments into variable parm */

/* TSO user id of the person invoking this rexx routine.             */
tsouid = userid()

CALL Initialize                                    /* Initializations */

sclmadm = 'admid1 admid2'    /* modifiy with your SCLM administrators */

/* Use ACCTINFO to find out the change user for the member.          */
CALL find_delete_member_owner

/* If the change user is same as the user invoking the delete service */
/* or the user invoking is an admin, then delete the module.         */
```

Archived

```
if (zsaluser = tsouid) | (pos(tsouid,sclmadm) > 0) then
   say 'Member 'member' can be deleted from group 'group' type 'type
else do
   say 'Can not delete member 'member' in group 'group' type 'type
   say 'Member is owned by 'strip(zsaluser)
   exit 0010
   end
exit

/*********************************************************************/
/* Initialize all variables, parameters, ....                        */
/*********************************************************************/
Initialize:

/* Parse parms passed by SCLM into variables & change to uppercase    */
PARSE UPPER VAR parm exittype ',' proj ',' prjdf ',' tsouid ','         ,
      group ',' type ',' member ',' flag ',' mode

exittype = STRIP(exittype,'T')
proj     = STRIP(proj,'T')
prjdf    = STRIP(prjdf,'T')
tsouid   = STRIP(tsouid,'T')
group    = STRIP(group,'T')
type     = STRIP(type,'T')
member   = STRIP(member,'T')
flag     = STRIP(flag,'T')
mode     = STRIP(mode,'T')

return

/*********************************************************************/
/* Use ACCTINFO to find out the owner of the module from the SCLM     */
/* accounting record.                                                */
/*********************************************************************/

find_delete_member_owner:
ADDRESS TSO

"FLMCMD ACCTINFO,"proj","prjdf","group","type","member",,,,,MATCH"

if rc > 0 then do
   say "ACCTINFO return code = " rc
   say 'Please, contact SCLM administrator. '
   exit 0030
   end
say 'owner is ' zsaluser
return
```

Archived

12.8.5 SCLM delete exits – delete notify exit

This exit is called for the Library Utility Delete (ISPF Option 10.3.1), Delete service, Delete Group service, and Delete Group function. The exit is invoked after the actual delete process has been completed. SCLM will only generate a Delete Exit file when a Delete Group is performed with the Delete Notify User Exit. The exit file is not needed for individual member deletions.

This exit might be used to delete output from external libraries. This exit can also be used in conjunction with the Promote Copy Exit that places SCLM output members into external libraries. This would perform a deletion, instead of a copy of the member. This exit could also notify a user that a delete has taken place. The EXITTYPE is DGNOTIFY for Delete Group and DNOTIFY for member delete.

FLMCNTRL parameters

These are the parameters:

- DELNTF=‘name of program’
- DELNTFDS=‘dataset where program resides’
- DELNTFCM=‘program call method’
- DELNTFOP=‘exit options list’

Any return code other than 0 will *stop the delete processing*. The only problem with this is that the delete of the member has been completed so there is no net effect to this. This exit is new for OS/390 version 2.10.

12.8.6 SCLM delete exits – delete notify exit – example

Example 12-14 is an example of a delete notify exit. It is coded to work in conjunction with the copy external exit that was documented in the build exit section. It is used to delete members from the external library when members are deleted inside of SCLM. Notice that this exit has been coded to run from either the DELETE service which processes a single member or the Delete Group service which will process multiple members. It uses the delete exit file, which is available during the delete notify exit.

Example 12-14 Delete notify exit example

```
/* REXX */
/*********************************************************************/
/* This rexx exec will delete members from the external libraries    */
/* they were copied to if they are deleted from SCLM                 */
/*********************************************************************/
trace r                                 /* o - off, r - result      */
Address 'ISPEXEC'                       /* Set proper environment   */

Arg parms                               /* Receive parms from SCLM  */

Call Init                               /* Initialize variables     */

If mode   = 'REPORT'   Then Exit        /* Exit if REPORT mode      */
If extyp  = 'DGNOTIFY' |,               /* Exit if not DELGROUP     */
   extyp  = 'DNOTIFY' then nop          /* or if not DELETE SERVICES*/
else exit
If flag  <> 'TEXT' & flag <> 'OUTPUT' Then Exit  /* Exit not delt mem */
If WORDPOS(type,valid_types) = 0 Then Exit    /* Exit - type not valid*/
```

```
if extyp = 'DNOTIFY' then do
   call do_delete group type member
   end
else do
   /* Read all lines from exit file into stem variable               */
   Address 'TSO' "EXECIO * DISKR "ddname" (STEM EXTLINE. FINIS)"

   if extline.0 = 0 Then Exit            /* Exit if exit file empty  */

   do i = 1 To extline.0                 /* For all lines in stem var*/
                                         /* Get vars from exit file  */
      parse upper var extline.i eodata 9 eogroup 18 eotype 27 eomember,
               36 eostatus 56 eooutput
      eodata   = STRIP(eodata)              /* Remove trailing blanks*/
      eogroup  = STRIP(eogroup)             /* Remove trailing blanks*/
      eotype   = STRIP(eotype)              /* Remove trailing blanks*/
      eomember = STRIP(eomember)            /* Remove trailing blanks*/
      eostatus = STRIP(eostatus)            /* Remove trailing blanks*/
      eooutput = STRIP(eooutput)            /* Remove trailing blanks*/

      if eodata   = 'MEMBER' Then do     /* If member delete ok cont */
         if eostatus = goodmsg | eostatus = warnmsg Then NOP
         else Iterate                       /* Skip bad records     */
         end
      Else Iterate                       /*Skip non-mem(bmap) deletes*/

      /* process the member types that can be deleted                */
      if WORDPOS(eotype,valid_types) = 0 then iterate

      Call do_delete eogroup eotype eomember /* Delete the member from*/
                                         /* external dataset     */
   end

   end
Exit exitrc

/*********************************************************************/
/* Init:     Subroutine to initialize variables, examine the parms   */
/*           passed from SCLM, and set additional variables.         */
/*********************************************************************/
Init:
   valid_types = 'CICSLOAD DBRMLIB LOAD'
   exitrc      = 0                /* Exit return code              */
                                  /* Separate parms passed from SCLM*/
   PARSE UPPER VAR parms extyp ',' proj ',' prjdf ',' tsouid ',',
         group ',' type ',' member ',' flag ',' mode

   extyp    = STRIP(extyp,'T')      /* Exit Type - where in SCLM     */
   proj     = STRIP(proj,'T')       /* Project - high level qualifier */
   prjdf    = STRIP(prjdf,'T')      /* Alternate Project Definition  */
   tsouid   = STRIP(tsouid,'T')     /* User executing SCLM function  */
   group    = STRIP(group,'T')      /* From delete group             */
   type     = STRIP(type,'T')       /* Type from the SCLM panel      */
   member   = STRIP(member,'T')     /* Member from the SCLM panel    */
   flag     = STRIP(flag,'T')       /* Flag  entered on SCLM panel   */
```

```
   mode     = STRIP(mode,'T')       /* Mode entered on SCLM panel     */

   SELECT
     WHEN extyp = 'DGNOTIFY' THEN
       DO
         dgnotify = 'DGNOTIFY'
         goodmsg  = 'DELETE SUCCESSFUL'
         warnmsg  = 'DELETE WARNING'
         badmsg   = 'DELETE FAILED'
         ddname   = 'DGEXIT'
       END
     WHEN extyp = 'DNOTIFY' THEN
       DO
         dnotify  = 'DNOTIFY'
         goodmsg  = 'DELETE SUCCESSFUL'
         warnmsg  = 'DELETE WARNING'
         badmsg   = 'DELETE FAILED'
       END
     OTHERWISE
       SAY 'Invalid exit type for DELGRP user exit: 'extyp
   END
Return

do_delete:
parse arg dgroup dtype dmbr
   /* Setup dataset for the delete */
   Select
      When dtype = 'LOAD' then do
         outdsn = "RDBK."dgroup".LOADLIB"
         End
      When dtype = 'CICSLOAD' then do
         outdsn = "RDBK."dgroup".CICSLOAD"
         End
      Otherwise do
         say 'No delete necessary for type ' dtype
         return
         End
   End

   Member_To_Be_Deleted = "'"outdsn"("dmbr")'"

   if sysdsn(Member_To_Be_Deleted) = 'OK' then do
      Address 'TSO' "DELETE " Member_To_Be_Deleted
      exitrc = MAX(exitrc,rc)

      If exitrc <> 0 then do
         Say 'Could not delete ' Member_To_Be_Deleted
         end
      else do
         /* Member was copied successfully */
         Say dmbr' has been deleted from 'outdsn
         end
      end
return
```

Archived

12.9 Audit and version delete

Figure 12-5 shows the SCLM audit and version delete user exits.

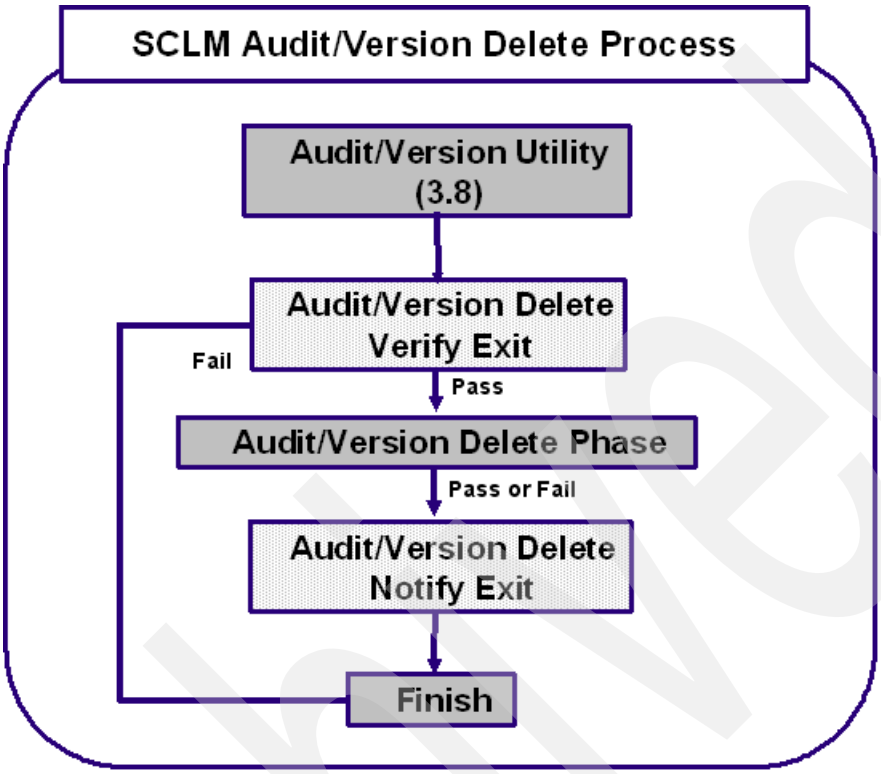


Figure 12-5 Audit / Version Delete user exits

12.9.1 SCLM audit and delete verify exit

This exit is invoked when an audit record or audit record and its associated version are deleted. It is invoked after verification but prior to delete. This exit will be called during the usage of the SCLM Audit Version Utility, Version Selection dialog (ISPF Option 10.3.8), or the VERDEL service interface. This exit might be used to check a userid against the group or type to ensure that the user has access to delete. Often, the SCLM administrator is the only one allowed to delete the audit and version files. The EXITTYPE is ADVERIFY.

FLMCNTRL parameters

These are the parameters:

- AVDVFY='name of program'
- AVDVFYDS='dataset where program resides'
- AVDVFYCM='program call method'
- AVDVFYOP='exit options list'

Any non-zero return code will *stop processing*. This exit is new for OS/390 version 2.10.

Archived

12.9.2 SCLM audit and delete notify exit

This exit is invoked when an audit record or audit record and its associated version are deleted. It is invoked after the delete takes place. This exit will be called during the usage of the SCLM Audit Version Utility, Version Selection dialog (ISPF Option 10.3.8), or the VERDEL service interface. This exit might be used to write the audit and version information to an external audit file. Some companies require obsolete program information to be retained for a specific number of years. This exit might also notify a user that a delete has taken place. The EXITTYPE is ADNOTIFY.

FLMCNTRL parameters

These are the parameters:

- AVDNTF=‘name of program’
- AVDNTFDS=‘dataset where program resides’
- AVDNTFCM=‘program call method’
- AVDNTFOP=‘exit options list’

Any return code other than 0 will *stop the delete processing*. The only problem with this is that the delete of the member has been completed. This exit is new for OS/390 version 2.10.

12.10 SCLM utility panel exit

The utility panel exit (SLMUEXIT) has the following characteristics:

- Optionally invoked from the SCLM Utility panel FLMUDU#P (SCLM 3.1)
- Used to add additional user options to the utility menu
- If variable SLMUEXIT is set, then the ISPF SELECT service is invoked using the value set in the SLMUEXIT variable as these three examples show:
  - &SLMUEXIT = 'CMD(USERUTIL)'
  - &SLMUEXIT = 'PANEL(SCLMUOPT)'
  - &SLMUEXIT = 'PGM(USERPGM) PARM(XYZ)'

Example 12-15 shows the exit invoked from the SCLM utility panel if the variable SLMUEXIT is set. The value assigned to this variable is the command, panel or program that you want an ISPF SELECT service run against.

Depending on panel space, you can add any number of local tools by setting the SLMUEXIT value.

Example 12-15 User exit called from the SCLM Utilities panel (FLMUDU#P)

```
/* example of adding a user report option */
/* IF (&ZCMD = 8) */
/*   &SLMUEXIT = 'CMD(USERUTIL)' */
   IF (&ZCMD = B)
     &SLMUEXIT = 'CMD(SCLMBMAI)'
   IF (&ZCMD = U)
     &SLMUEXIT = 'PANEL(SCLMUOPT)'
&OPT = &ZCMD
```

12.11 Additional SCLM user exit example

These are two additional examples of how user exits can be used

- **Example 1:** Edit user exit example

  This example shows how we interface with a vendor product, in this case, Breeze, to stop a member from being edited if it is part of a package that is still pending promotion.

- **Example 2:** Build or Promote user exit example

  This example is of a project that contains DB2 code and needs to use submitted jobs to control the binds.

12.11.1 Example 1: BZZXXCDT user exit

Example 12-16 is the CCVFY exit invoked when a user attempts to edit a member. The exit checks to see if this member is part of a Breeze package that is awaiting approval or promotion. If it is then edit is denied.

Processing in the user exit is performed as follows:

1. Retrieve SCLM variables and parse user exit parameters.
2. Ensure a member name has been selected.
3. Allocate Breeze datasets.
4. Set the status to “Pending” and Call procedure “Extract_Breeze” to build syntax of Breeze reporting request and then call the Breeze reporting module.
5. If some data is found, then parse the report output to see if the selected member is found in a Breeze package.
6. Repeat these two steps for “Approved” packages.
7. If member is found, put out a message and deny edit.

Example 12-16 Verify exit for breeze code

```
/* REXX */
/*********************************************************************/
/* Check if member being edited is part of a package that is being    */
/* promoted by Breeze. If so either put out warning or stop the edit  */
/*********************************************************************/
   tsoid = userid()

   Address ISPEXEC 'VGET (SPRJ1,LIBDEF)'
   If LIBDEF = '' Then
      LIBDEF = SPRJ1

   Arg PARM
   Parse var PARM GROUP ',' TYPE ',' MEMBER ',' LANG ',' ,
                  USERID ',' AUTHCODE ',' CCODE

   If MEMBER = '' Then Do
      Say 'Member name must be present. Either enter member '
      Say 'name or use SCLM option 3.1 to process member lists'
      Exit_RC = 8
   End
   Else Do
```

Archived

```
      "ALLOC ddname(CIGIN) recfm(f b),lrecl(80) blksize(8000) space(1 1),
         tracks unit(vio)"
      "ALLOC F(CIGLOG) NEW TRACKS UNIT(VIO)
             BLKSIZE(13300) LRECL(133) RECFM(F B A) SPACE(1 1)"
      "ALLOC F(CIGRPT) NEW TRACKS UNIT(VIO)
             BLKSIZE(13300) LRECL(133) RECFM(F B A) SPACE(1 30)"
      "ALLOC F(CIGOUT) DUMMY REUSE "

      Exit_RC = 0
      STATUS = 'PENDING'
      BZZ_rc = Extract_Breeze()
      If BZZ_rc = 0 Then Do
         If Member_in_package() = 4 then
            BZZ_rc = 8
         Else
            BZZ_rc = 4
      End
      If BZZ_rc = 4 Then
      Do
         STATUS = 'APPROVED'
         BZZ_rc = Extract_Breeze()
         If BZZ_rc = 0 Then
         Do
            If Member_in_package() = 4 then
               BZZ_rc = 8
            Else
               BZZ_rc = 4
         End
      End

      If BZZ_rc = 8 Then
      Do
         Say 'Member being edited is part of a PENDING/APPROVED'
         Say 'Breeze package. Edit not allowed until member has'
         Say 'been promoted from the 'Strip(GROUP)' group.'
         Say 'Breeze Package : 'package
         Say 'Package Details :'
         Say ' 'Package_details
         Exit_RC = 8
      End

      "FREE DDNAME(CIGLOG)"
      "FREE DDNAME(CIGRPT)"
      "FREE DDNAME(CIGOUT)"
      "FREE DDNAME(CIGIN)"
   End
Exit (Exit_RC)

Extract_Breeze :
   Address TSO
   Newstack
   QUEUE "    REPORT PACKAGES * "
   QUEUE "    WHERE STATUS   = "STATUS
   QUEUE "    WHERE PROJECT  = "SPRJ1
   QUEUE "    WHERE ALTPROJ  = *"
```

Archived

```
   QUEUE "    WHERE GROUP    = "GROUP
   QUEUE "    OPTIONS CONTENTS " " "
   QUEUE " . "
   QUEUE ""

   x = msg("off")
   Address TSO
   "execio * diskw CIGIN (finis)"
   DELSTACK

   "CALL 'BZZ.SBZZLOAD(BZZS0000)'     "
   BZZ_rc = rc
Return BZZ_rc

Member_in_package :
   "EXECIO * DISKR CIGRPT (FINIS STEM INPUT."

   EOF = 'N'
   j = 0
   I = 1
   If I > INPUT.0 Then EOF = 'Y'
   Do While (EOF = 'N')
      If I > INPUT.0 Then EOF = 'Y'

      Do While (pos('FOR PACKAGE:',Strip(INPUT.I)) = 0 & EOF = 'N')
         I = I + 1
         If I > INPUT.0 Then EOF = 'Y'
      End
      If pos('FOR PACKAGE:',Strip(INPUT.I)) > 0 then
      Do
         Parse var INPUT.I 'FOR PACKAGE: ' Package .
         I = I + 1
         If I > INPUT.0 Then EOF = 'Y'
      End
      Do While (pos('PROJECT:',Strip(INPUT.I)) = 0 & EOF = 'N')
         I = I + 1
         If I > INPUT.0 Then EOF = 'Y'
      End
      If pos('PROJECT:',Strip(INPUT.I)) > 0 then
      Do
         Package_details = INPUT.I
         I = I + 1
         If I > INPUT.0 Then EOF = 'Y'
      End
      Do While (pos('MONITORED CONTENT',Strip(INPUT.I)) = 0 & EOF = 'N')
         I = I + 1
         If I > INPUT.0 Then EOF = 'Y'
      End
      If pos('MONITORED CONTENT',Strip(INPUT.I)) > 0 then
      Do
         Do While (pos('FOR PACKAGE:',Strip(INPUT.I)) = 0 & EOF = 'N')
            parse var INPUT.I BZZMEM BZZPRJ BZZALT BZZGRP BZZTYP BZZLAN
            If BZZMEM = MEMBER &,
               BZZGRP = GROUP &,
               BZZTYP = TYPE Then
```

Archived

```
            Do
               BZZ_rc = 4
               Return BZZ_rc
            End
            I = I + 1
            If I > INPUT.0 Then EOF = 'Y'
         End
      End
      Do While (pos('FOR PACKAGE:',Strip(INPUT.I)) = 0 & EOF = 'N')
         I = I + 1
         If I > INPUT.0 Then EOF = 'Y'
      End
   End

   BZZ_rc = 0

Return BZZ_rc
```

12.11.2 Example 2: bind external exit

In this example the build notify and promote purge exits are used to call a bind exit that creates an externally submitted job to perform the bind. Previously a bind process was documented that shows how the binds can be performed from within the SCLM build job. The bind external process documented here has several potential advantages versus the internal bind process:

1. The first potential advantage is that since the bind process runs from a submitted job, it can be submitted to run on other LPARs. This is necessary if the DB2 subsystem for a particular SCLM group exists on another LPAR. The job can be submitted on another LPAR in one of two ways. If the JES job queue is shared across LPARs, then a JOBPARM statement can be added to the job card to direct the execution of the job to the LPAR where the DB2 subsystem exists. This method is documented in the example below. If JES is not shared, then an FTP or perhaps NDM process can be created to ship the bind job to another LPAR.
2. The second advantage of the submitted bind process is that it is easier to set up versus internal bind processing, which requires individual bind CLIST and ARCHDEFs to be created. The external bind process requires no additional types or members. However it does have a limitation in that it is difficult to override bind parameters for individual members. It works best when all bind parameters can be set in the logic in the bind user exit.
3. The third advantage is that the submitted bind job can be submitted in a held job status to be executed later if you do not want the binds to occur immediately. In fact you could code the skeleton process to save the bind jobs to a dataset where they could be run on demand.

However, the advantages and disadvantages between the two bind processes are not clear-cut. The internal bind process can be coded to submit bind jobs and the user exit bind process can be coded to read bind control members to find bind parameters for individual members. These variations are not shown.

Example 12-17 is the common build notify exit code modified to call the bind external exit.

Example 12-17 Bind exit - common exit

```
/* REXX */
/*********************************************************************/
/*  This exec will set up an ISPF environment for the execution of   */
/*  the build exits                                                  */
/*********************************************************************/
TRACE o
ARG parm                 /* Parse arguments into variable parm */
CALL Init
bldntfrc = 0

ADDRESS ISPEXEC
"SELECT CMD(EX 'SCLM07.PROJDEFS.REXX(COPYEXTR)' '"parm"')"
bldntfrc = MAX(bldntfrc,rc)

"SELECT CMD(EX 'SCLM07.PROJDEFS.REXX(BINDEXTR)' '"parm"')"
bldntfrc = MAX(bldntfrc,rc)

"SELECT CMD(EX 'SCLM07.PROJDEFS.REXX(DEPLOY)' '"parm"')"
bldntfrc = MAX(bldntfrc,rc)
exit bldntfrc
```

Example 12-18 is the sample bind external exit. Notice that the exit uses file tailoring using ISPF skeletons. These skeletons will have to be created also as documented below. This exit will have to be customized with your DBRMLIB type and with the bind parameters you want to use for each group where the bind job needs to run.

Example 12-18 Bind exit example

```
/* REXX */
/*********************************************************************/
/*  This exec is called from the promote purge exit and build exits  */
/*  to perform all binds for all groups.                             */
/*                                                                   */
/*  After including the proper skeletons for the bind it submits a   */
/*  job to perform the binds.  The job is submitted in held job status*/
/*                                                                   */
/*********************************************************************/
trace o                                          /* trace facility */
ARG parm                /* original sclm parms will be passed here */

CALL Init                                       /* Initializations */

/* If the promote is done in the report mode this exec should not    */
/* get executed.                                                     */
if mode = 'REPORT' then
   exit
/* If not build exit 1 or promote purge exit return                  */
if extyp <> bldext1 & extyp <> prmext3 THEN
   exit

/*  Set the group for the bind                                       */
SELECT
  WHEN extyp = bldext1 THEN
      bndgrp = fromgrp
```

```
  WHEN extyp = prmext3 THEN
      bndgrp = togrp
  OTHERWISE
    exit
END  /* Select stmt */

/* Read all lines from exit file into a stem variable                */
"execio * diskr "ddname" (stem extline. finis)"
if extline.0 = 0 then   /* If file was empty                         */
   exit

do i = 1 to extline.0   /* For all lines in stem variable            */
   /* add members to be processed to a set of ISPF tables            */
   call chkdb2
end

/* If dbrmlib is being promoted then create jobs to process bind     */
if db2mbrs > 0 then do
   call process

   saverc = rc
   if saverc <> 0 then do
      say ' Unable to create and submit bind job, rc is = ' saverc
      say ' Contact SCLM administrator'
      maxrc = MAX(maxrc,saverc)
      end
  end
Exit maxrc

/*********************************************************************/
/* Init:     Subroutine to initialize variables, examine the parms  */
/*           passed from SCLM, and set additional variables.        */
/*********************************************************************/
Init:
   ftopenV  = 0
   maxrc    = 0
   db2mbrs  = 0    /* Bind Count */

   /* Set standard BIND options */
   flag      = 'I'
   iso       = 'CS'

   /* Break out the parms passed from the SCLM user exit */
   PARSE UPPER VAR parm extyp ',' proj ',' prjdf ',' tsouid ',',
         fromgrp ',' type ',' member ',' scope ',' mode ',' togrp

   extyp    = STRIP(extyp,'T')    /* Exit Type - where in SCLM?     */
   proj     = STRIP(proj,'T')     /* Project - high level qualifier */
   prjdf    = STRIP(prjdf,'T')    /* Alternate Project Definition   */
   tsouid   = STRIP(tsouid,'T')   /* User executing SCLM function   */
   fromgrp  = STRIP(fromgrp,'T')  /* From grp(Promote) or build grp */
   type     = STRIP(type,'T')     /* Type from the SCLM panel       */
   member   = STRIP(member,'T')   /* Member from the SCLM panel     */
   scope    = STRIP(scope,'T')    /* Scope entered on SCLM panel    */
   mode     = STRIP(mode,'T')     /* Mode entered on SCLM panel     */
```

Archived

```
   togrp    = STRIP(togrp,'T')    /* To grp (Promote), N/A (Build)  */

   bnduser  = tsouid

   SELECT
     WHEN extyp = 'BUILD' THEN
       DO
         Bldext1  = 'BUILD'
         goodmsg  = 'BUILT'
         badmsg   = ''
         ddname   = 'BLDEXIT'
       END

     WHEN extyp = 'PPURGE' THEN
       DO
         Prmext3  = 'PPURGE'
         goodmsg  = 'PURGE SUCCESSFUL'
         badmsg   = 'PURGE FAILED'
         ddname   = 'PROMEXIT'
       END

     OTHERWISE
       SAY 'Invalid exittype for BIND user exit: 'extyp
   END  /* Select stmt */

Return

/*********************************************************************/
/* chkdb2:   Subroutine to process each exit record to determine    */
/*           if any database programs are being processed           */
/*********************************************************************/

chkdb2:                       /* process each member from exit file */

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

eomember= STRIP(eomember)
eogroup = STRIP(eogroup)
eotype  = STRIP(eotype)
eostatus= STRIP(eostatus)
mbr = eomember
bindmbr = mbr

/* If member ok continue */
if eostatus = goodmsg then do
   if eotype = 'DBRM' then do
      call addBind
      end /* if eotype is dbrmlib                                    */
   end /* if eostatus is goodmsg                                     */
Return

/*********************************************************************/
/*                                                                   */
/* addBind: Add members needing bind to table                        */
```

Archived

```
/*                                                                   */
/*********************************************************************/
addBind:
Address ISPEXEC

call setBindParms

if db2mbrs = 0 then do
   /* create table to hold the OP members that need a BIND */
  "TBCREATE DB2MBRS NAMES(mbr pkg qual own act val exp) NOWRITE REPLACE"
   saverc = rc
   maxrc = MAX(maxrc,saverc)
   end

"TBADD DB2MBRS"                /* Add member to table    */
saverc = rc
maxrc = MAX(maxrc,saverc)
if saverc = 0 then
   db2mbrs = db2mbrs + 1
return

/*********************************************************************/
/*                                                                   */
/* Process:  Subroutine to include the bind job skeletons            */
/*                                                                   */
/*********************************************************************/
Process:
Address 'ISPEXEC'         /* Issue all commands to ISPF              */

if db2mbrs > 0 then do
   "FTOPEN TEMP"               /* File tailoring to temp dataset   */

   if bndgrp <> 'PROD' then
      "FTINCL BINDJOB"         /* Include job card                 */
   else
      "FTINCL BINDJOBP"        /* Include job card (production)    */

   /* Include DB2 Bind Statements                                   */
   "FTINCL BINDMBR"
   saverc = rc

   if saverc <> 0 then do
      say ' Unable to include the bind skeleton, rc is = ' saverc
      say ' Contact SCLM administrator'
      maxrc = MAX(maxrc,saverc)
      end

   "TBEND DB2MBRS"                /* Delete table; no longer needed*/

   if saverc = 0 then call submit_job
   saverc = rc

   if saverc <> 0 then do
      say ' Unable to submit the bind job, rc is = ' rc
      say ' Contact SCLM administrator'
```

```
      maxrc = MAX(maxrc,saverc)
      end
return

/*********************************************************************/
/*   Close file tailoring and submit bind job                        */
/*********************************************************************/
Submit_job:
subrc = 0

"FTINCL BINDEND"           /* Finished file tailoring              */
subrc = rc

if subrc <> 0 then do
   say ' Unable to include the bindend skeleton, rc is = ' rc
   say ' Contact SCLM administrator'
   maxrc = MAX(maxrc,subrc)
   end

"FTCLOSE"                          /* Finished file tailoring       */
subrc = rc

if subrc = 0 Then do
   "VGET ZTEMPF"
   address TSO
   Dummy = Outtrap("cmd_output_line.","*")
   Address 'TSO' "SUBMIT '"ztempf"'"
   subrc = rc
   Parse Var cmd_output_line.1 . jobname "(" jobnum ")" .
   if subrc = 0 then do
      say copies('*',79)
      say left('* A bind job for all members'||,
               ' has been submitted in held job status .',77) '*'
      say left('* Job name: 'jobname' , Job Id: '||,
                jobnum'.',77) '*'
      say left('* Please release the job when'||,
               ' you are ready to have the binds performed.',77) '*'
      say copies('*',79)
      end
   else do
      say copies('*',79)
      say left('* The submit of the bind job '||,
               'has failed.',77) '*'
      say copies('*',79)
      subrc = 100
      end
   end
maxrc = MAX(maxrc,subrc)
Return subrc

/*********************************************************************/
/*   Setup the Bind Parameters for the group and database            */
/*********************************************************************/
setBindParms:
```

Archived

```
select
  when bndgrp = 'DEV1' | bndgrp = 'DEV2' then do
    sys    = 'DI11'
    pkg    = bndgrp
    qual   = 'DEVDBA'
    own    = 'DEVDBA'
    exp    = 'NO'
    val    = 'RUN'
    act    = 'REP'
    end
  when bndgrp = 'TEST' then do
    sys    = 'DI11'
    pkg    = bndgrp
    qual   = 'TESTDBA'
    own    = 'TESTDBA'
    exp    = 'NO'
    val    = 'RUN'
    act    = 'REP'
    end
  when bndgrp = 'PROD' then do
    sys    = 'DI11'
    pkg    = bndgrp
    qual   = 'PRODDBA'
    own    = 'PRODDBA'
    exp    = 'NO'
    val    = 'RUN'
    act    = 'REP'
    end
  otherwise do
    say 'INVALID GROUP PASSED'
    say 'GROUP PASSED IS' bndgrp
    say 'CONTACT SCLM ADMINISTRATOR'
    EXIT 312
    end
end
return
```

Example 12-19 is the non-PROD bind job skeleton (BINDJOB) used by the bind exit. It needs to be put in the same skeleton library as your customized FLMLIBS skeleton. The job card and steplib will have to be coded to local standards. Remove the typrun=hold line if you want the bind to be immediate.

Example 12-19 BINDJOB skeleton

```
//SCLMBIND JOB (ACCOUNT),'SCLM BIND IN &BNDGRP.',
// TYPRUN=HOLD,
// MSGCLASS=X,MSGLEVEL=(1,1),
// NOTIFY=&&SYSUID
//*
//*****************************************************************
//* SCLM BIND JOB                                                 *
//*****************************************************************
//DB2BIND  EXEC  PGM=IKJEFT1A,REGION=2560K
//STEPLIB  DD  DSN=DB2.V810.SDSNLOAD,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSTSPRT DD  SYSOUT=*
```

Archived

```
//SYSOUT   DD  SYSOUT=*
//DBRMLIB  DD  DSN=&PROJ..&BNDGRP..DBRM,DISP=SHR
//SYSTSIN  DD  *
 DSN SYSTEM(&SYS.)
```

Example 12-20 is the production bind job skeleton (BINDJOBP) used by the bind exit. It needs to be put in the same skeleton library as your customized FLMLIBS skeleton. It needs to be customized for your local environment and the JOBPARM statement needs to be coded to point to your production LPAR if you wan to run the bind job there.

Example 12-20 BINDJOBP skeleton

```
//SCLMBNDP JOB (ACCOUNT),'SCLM BIND IN &BNDGRP.',
// TYPRUN=HOLD,
// MSGCLASS=X,MSGLEVEL=(1,1),
// NOTIFY=&&SYSUID
/*JOBPARM S=PROD
//*
//*******************************************************************
//* SCLM BIND JOB                                                   *
//*******************************************************************
//DB2BIND  EXEC  PGM=IKJEFT1A,REGION=2560K
//STEPLIB  DD  DSN=DB2.V810.SDSNLOAD,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSTSPRT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//DBRMLIB  DD  DSN=&PROJ..&BNDGRP..DBRM,DISP=SHR
//SYSTSIN  DD  *
 DSN SYSTEM(&SYS.)
```

Example 12-21 is the BINDMBR skeleton. You will need to customize this with the bind values that you want to vary from member to member and with any values that you want to code specifically for each program. Corresponding changes to the bind exit would have to be performed also.

Example 12-21 BINDMBR skeleton

```
)DOT DB2MBRS
 BIND PACKAGE (&PKG.) -
   MEMBER(&MBR.) -
   OWNER (&OWN.) -
   QUALIFIER (&QUAL.) -
   VALIDATE (&VAL.) -
   ISOLATION (&ISO.) -
   ACTION(&ACT.) -
   EXPLAIN (&EXP.)
)ENDDOT
```

Example 12-22 is the small BINDEND skeleton.

Example 12-22 BINDEND skeleton

```
/*
//*
```

Archived

Archived

13

Creating language definitions from JCL

Many sites use Job Control Language (JCL) to run preprocessors, compilers, linkage editors, and other tools used in the development process. SCLM supports developers and project administrators through the use of language definitions that tell SCLM how to parse, build, and promote members of an SCLM-controlled data set. Language definitions can also specify additional translators to execute for the COPY, PURGE, and VERIFY functions.

Because the SCLM language definitions provide an easier method of implementing processing control than JCL does, many sites have found it beneficial to convert their JCL to SCLM language definitions. To ease the conversion process, SCLM provides sample language definitions that you can tailor to the special needs of your site.

In this chapter we explain how to construct SCLM language definitions to replace existing JCL runstreams. Examples illustrate the basic principles underlying a successful migration from JCL to SCLM language definitions and also demonstrate methods for avoiding potential problems and conflicts.

Archived

© Copyright IBM Corp. 2007. All rights reserved.

13.1 Preparing to convert

Before you try to convert your existing JCL runstreams to SCLM language definitions, you must obtain and review “expanded” listings of the JCL. The expanded JCL listings allow you to determine the actual values of the symbolic parameters in the JCL; these values include data set names, options, and other information that is required for successful translation to an SCLM language definition. You will also need to know the order in which programs are executed in the JCL, and the condition codes that are expected from each program. Your system administrator should be able to help you locate this information.

You should also review the information provided about SCLM macros in the z/OS ISPF Software Configuration and Library Manager Reference, paying special attention to the following macros and their parameters:

- FLMTRNSL
- FLMTCOND
- FLMALLOC
- FLMCPYLB
- FLMINCLS
- FLMTOPTS

13.2 Capabilities and restrictions

There are two basic equivalencies that you will use to convert JCL statements to SCLM macro statements:

- Every JCL EXEC statement with PGM=abc will correspond to an FLMTRNSL macro with COMPILE=abc in your language definition. Conditional execution of BUILD translators may be addressed through use of the FLMTCOND macro.
- Every JCL DD statement will correspond to an FLMALLOC macro and/or an FLMSYSLB macro associated with an FLMALLOC macro in your language definition.

In the case of STEPLIB, the JCL DD statement will correspond to the DSNAME parameter in the FLMTRNSL macro. A STEPLIB concatenation of more than one data set would use the TASKLIB parameter. The TASKLIB parameter is set to the ddname associated with the data set concatenation. FLMCPYLBs are used to specify the data sets on an FLMALLOC macro with DDNAME set to the TASKLIB ddname. When both DSNAME and TASKLIB are specified, the DSNAME data set is searched first, followed by the TASKLIB data sets, followed by the system concatenation.

In the case of SYSLIB-type ddnames for a compiler, the data sets must be specified FLMSYSLBs. Then either ALCSYSLB=Y must be specified on the FLMLANGL macro and/or FLMCPYLBs must be specified for the appropriate FLMALLOC macros. For an example of this, refer to the Enterprise COBOL (FLM@COBE) or C/370™ (FLM@C370) language definitions supplied with SCLM.

Three areas of restrictions can prevent a simple, one-to-one translation of JCL statements to SCLM macro statements:

- Backward referencing of data definition names (DDs):

  If a JCL DD statement uses the “refer back” technique to reference a previous DD statement (other than the one in the preceding step), or if a DD statement refers to a data set using a ddname that differs from the data set's ddname in a prior step, conversion to an SCLM language definition can involve the use of an intermediate translator or a

Archived

ddname substitution list in order to allocate the correct data set name for the program. (An intermediate translator is not needed if the succeeding translator supports DDNAME substitution lists; in this case, the succeeding translator can “hard-code” the DDNAME and use IOTYPE=U on the FLMALLOC macro.)

- Complex conditional execution:

  A JCL runstream that specifies skipping all steps after a specified condition code from one or more previous steps is directly converted to appropriate FLMTRNSL macros with appropriate GOODRC values. Other conditional executions of BUILD translators can be addressed by using the FLMTCOND macro. For example, if the JCL is set up to run BUILD translator X if any previous return code is 4, but run Build translator Y if any previous return code is 8, you can use the FLMTCOND macro. FLMTCOND is only valid for use with BUILD translators. Conditional execution of non-BUILD translators can require modification of the translators or interface programs to handle the control of execution.

- TSO Address Space compatibility:

  Some programs that run from JCL will not run in the TSO Address Space in which SCLM resides without a special interface translator. IBM has provided interface programs for several common IBM programs with this characteristic. For example, the FLMTMSI (SCRIPT), FLMTMJI (JOVIAL), and FLMTMMI (DFSUNUB0) translators all use the TSO Service Facility IKJEFTSR.

If you have JCL that runs program XYZ without any errors, but fails when you try to run program XYZ from an FLMTRNSL macro, this may be the problem. You must write a translator to call the program using IKJEFTSR.

The following sections describe how to convert JCL statements and runstreams into functionally equivalent SCLM language definitions and provide suggested strategies for working around restrictions and conflicts.

13.3 Converting JCL statements to SCLM macro statements

This section contains examples of JCL runstreams and their SCLM language definition equivalents.

13.3.1 Executing programs

The SCLM FLMTRNSL macro is similar to a JCL EXEC (EXECUTE)statement.

Example 13-1 shows a single JCL statement that runs a program named IEFBR14.

Example 13-1 Running program IEFBR14 from JCL

```
//STEP1    EXEC    PGM=IEFBR14
```

Example 13-2 shows an SCLM FLMTRNSL macro that performs the same task as the JCL statement in Example 13-1.

Example 13-2 Running program IEFBR14 from an SCLM translator

```
*
     FLMTRNSL   COMPILE=IEFBR14,FUNCTN=BUILD,PORDER=0
```

FLMTRNSL's COMPILE option specifies the name of the program to execute (IEFBR14.) The FUNCTN parameter specifies here that IEFBR14 will be invoked when the user requests a

BUILD, and the PORDER value of 0 tells SCLM that neither an option list nor a ddname substitution list will be passed to IEFBR14.

Example 13-3 is a slightly more complex example. We want to use a translator program named GAC to copy the contents of TSOSCxx.DEV1.SOURCE(MEMBER1) into TSOSCxx.DEV1.LIST(MEMBER1). The GAC program itself requires a SYSIN data set, which is empty in this example.

Example 13-3 Running program GAC from JCL

```
//STEP1    EXEC    PGM=GAC
//SYSIN    DD      DUMMY
//INPUT    DD      DSN=TSOSCxx.DEV1.SOURCE(MEMBER1),DISP=SHR
//OUTPUT   DD      DSN=TSOSCxx.DEV1.LIST(MEMBER1),DISP=SHR
```

Example 13-4 shows the SCLM language definition that performs the same task as the JCL in Example 13-3.

Example 13-4 Running program GAC from an SCLM translator

```
*
         FLMTRNSL   COMPILE=GAC,FUNCTN=BUILD,PORDER=0
*
          FLMALLOC   IOTYPE=A,DDNAME=SYSIN
           FLMCPYLB   NULLFILE
*
          FLMALLOC   IOTYPE=A,DDNAME=INPUT
           FLMCPYLB   TSOSCxx.DEV1.SOURCE(MEMBER1)
*
          FLMALLOC   IOTYPE=A,DDNAME=OUTPUT
           FLMCPYLB   TSOSCxx.DEV1.LIST(MEMBER1)
*
```

As before, the FLMTRNSL macro is used to specify the name of the program to run. The FLMALLOC and FLMCPYLB statements allocate the existing data sets to ddnames.

13.3.2 Conditional execution

Example 13-5 program XYZ runs only if the return code from program ABC is 4 or less.

Example 13-5 Specifying conditional execution with JCL

```
//STEP1     EXEC    PGM=ABC
//STEP2     EXEC    PGM=XYZ,COND=(4,LT)
```

In SCLM, the GOODRC parameter on the FLMTRNSL macro allows you to specify return code values for conditional execution. In Example 13-6, the GOODRC parameter for program ABC is set to 4. If ABC ends with a return code greater than 4, processing ends; program XYZ will not execute.

Example 13-6 Specifying conditional execution in an SCLM Language definition

```
*
     FLMTRNSL   COMPILE=ABC,FUNCTN=BUILD,PORDER=0,GOODRC=4
     FLMTRNSL   COMPILE=XYZ,FUNCTN=BUILD,PORDER=0
```

Archived

In Example 13-7, program XYZ runs only if the return code from program ABC is 4 or less. Program MBS is to execute after program XYZ regardless of the previous return codes.

Example 13-7 More complex conditional expression using JCL

```
//STEP1     EXEC    PGM=ABC
//STEP2     EXEC    PGM=XYZ,COND=(4,LT)
//STEP3     EXEC    PGM=MBS
```

In SCLM, the GOODRC parameter on the FLMTRNSL macro specifies when to skip all remaining translators in the language definition. Therefore, in Example 13-8, the FLMTCOND macro is used so that execution can skip program XYZ but continue with program MBS.

Example 13-8 More complex conditional expression with SCLM language definition

```
*
     FLMTRNSL   COMPILE=ABC,FUNCTN=BUILD,PORDER=0
     FLMTRNSL   COMPILE=XYZ,FUNCTN=BUILD,PORDER=0
      FLMTCOND ACTION=SKIP,WHEN=(*,GT,4)
     FLMTRNSL   COMPILE=MBS,FUNCTN=BUILD,P0RDER=0
```

13.3.3 Sample JCL conversion

This section contains commented sample JCL and language definitions that perform the same tasks: invoking the CICS Transaction Server (CICS TS) preprocessor and then invoking the Enterprise COBOL compiler to produce an object module.

Example 13-9 on page 296 contains the JCL used to accomplish these tasks, while Figure 13-10 contains the equivalent SCLM language definition. Each sample contains comments with step numbers. The step descriptions that follow relate a line or command from the JCL to the equivalent SCLM language definition macro, option, or command:

1. The JCL has a job step named TRN, which is the first translator called in this job.

   SCLM uses an FLMTRNSL macro to call this translator. This is the first FLMTRNSL macro for build in the language definition.

2. Job step TRN executes a program called DFHECP$1, the CICS preprocessor for COBOL.

   SCLM uses the COMPILE=DFHECP$1 statement on the FLMTRNSL macro.

3. 3. The STEPLIB statement in job step TRN tells the job where to find the program DFHECP$1.

   SCLM uses the DSNAME option on the FLMTRNSL macro. Both the STEPLIB and DSNAME point to the same data set, CICS.TS31.CICS.SDFHLOAD.

4. The SYSIN statement defines the data set that contains the member to compile.

   SCLM uses an FLMALLOC macro to allocate the SYSIN data set to a ddname for the CICS preprocessor. Because we are using PORDER=1, the FLMALLOC macro assigns the ddname, SYSIN, that the CICS preprocessor is expecting.

5. The TRN job step sends the preprocessor listing to the printer using the SYSPRINT statement.

   SCLM uses an FLMALLOC macro to allocate an output data set to the ddname SYSPRINT.

6. The SYSPUNCH statement in the TRN step creates the output of the CICS preprocessor and passes it to the next job step (COB) as a temporary file.

Archived

SCLM uses an FLMALLOC macro with IOTYPE=W to allocate a work (temporary) file with the ddname of SYSPUNCH. This work file is passed to the next FLMTRNSL macro.

7. The JCL has a job step named COB, which is the second translator called in this job.

   SCLM uses an FLMTRNSL macro to call this translator. This is the second FLMTRNSL macro for build in our language definition.

8. The job step COB executes (EXEC PGM=) a program called IGYCRCTL, the compiler for Enterprise COBOL.

   SCLM uses the COMPILE=IGYCRCTL statement on the FLMTRNSL macro.

9. To pass compiler options to the Enterprise COBOL compiler, the COB job step uses a PARM=parameter of the EXEC statement.

   SCLM uses the OPTIONS= parameter of the FLMTRNSL macro to perform the same task.

10. This job has conditional execution for the COB step via the COND(5,GE) JCL parameter of the EXEC statement. The COB step will not execute if the return code of the TRN step is greater than 4.

    SCLM sets the GOODRC keyword parameter for the TRN step (CICS preprocessor) equal to 4. Build halts execution of all translators following the TRN step in the language definition if the return code from the TRN step is greater than 4.

11. The STEPLIB statement in the COB job step tells the job where to find the program IGYCRCTL.

    SCLM uses the DSNAME= option on the FLMTRNSL macro. Both the STEPLIB and DSNAME point to the same data set,ECOBOL.V340.SIGYCOMP.

12. The SYSLIB statement in job step COB tells the job where to find the system type includes.

    The language definition uses the FLMSYSLB macro with IOTYPE=I and the FLMINCLS macro to do the same task.

    SCLM allocates these project data sets allocated for IOTYPE=I before the data sets on the FLMCPYLB macro(s). ALCSYSLB=Y parameter must be specified on the FLMLANGL macro to ensure that the FLMSYSLB data sets are allocated to the IOTYPE=I ddnames.

    Because PORDER=3 is being used, the SYSLIB DD is the fourth ddname passed to the compiler in a ddname substitution list. The COBOL compiler uses the fourth ddname as SYSLIB no matter what value is assigned to the DDNAME keyword parameter on the FLMALLOC macro.

13. For each system library specified for the SYSLIB DD, the language definition has an FLMSYSLB macro. In this case both CICS.TS31.CICS.SDFHCOB and CICS.TS31.CICS.SDFHMAC are specified.

14. The COB job step sends the compile listing to the printer using the SYSPRINT statement.

    SCLM uses an FLMALLOC macro to allocate an output data set to the ddname SYSPRINT.

15. In the COB job step, the SYSIN DD statement identifies the data set that contains the member to compile. This is the output of the CICS preprocessor step TRN.

    SCLM uses an FLMALLOC macro with IOTYPE=U to refer to a ddname from a prior step. The language definition instructs MVS to allocate the data set assigned in the TRN step to the ddname SYSPUNCH.

16. The SYSLIN statement in the COB step identifies the output data set for object code created by the COBOL compiler.

    The language definition uses an FLMALLOC macro with IOTYPE=O to allocate an output file. This FLMALLOC macro is the first in the COB FLMTRNSL because when using PORDER=3, the Enterprise COBOL compiler expects the output data set ddname to be first in a ddname substitution list.

17. The COB step allocates SYSUT1 as a temporary work file for the Enterprise COBOL compiler.

    SCLM's language definition uses an FLMALLOC macro with IOTYPE=W to perform the same task. This must be the eighth file provided to the OS COBOL compiler because PORDER=3 tells SCLM that we are using a ddname substitution list.

18. The COB step allocates SYSUT2 as a temporary work file for the Enterprise COBOL compiler.

    SCLM's language definition uses an FLMALLOC macro with IOTYPE=W to perform the same task. This must be the ninth file provided to the Enterprise COBOL compiler because we are using a ddname substitution list.

19. The COB step allocates SYSUT3 as a temporary work file for the Enterprise COBOL compiler.

    SCLM's language definition uses an FLMALLOC macro with IOTYPE=W to perform the same task. This must be the tenth file provided to the Enterprise COBOL compiler because we are using a ddname substitution list.

20. The COB step allocates SYSUT4 as a temporary work file for the Enterprise COBOL compiler.

    SCLM's language definition uses an FLMALLOC macro with IOTYPE=W to perform the same task. This must be the eleventh file provided to the Enterprise COBOL compiler because we are using a ddname substitution list.

21. The COB step allocates SYSUT5 as a temporary work file for the Enterprise COBOL compiler.

    SCLM's language definition uses an FLMALLOC macro with IOTYPE=W to perform the same task. This must be the twelfth file provided to the Enterprise COBOL compiler because we are using a ddname substitution list.

22. SCLM language definition only

    The language definition uses PORDER=3 for the Enterprise COBOL compiler step (COB) to use a ddname substitution list. A ddname substitution list provides an ordered list (defined by the translator) of ddnames such that the position of a ddname in the list, and not the actual ddname, is used by the translator for a specific file.

    The input file for the compiler must be the output file from the CICS preprocessor. The ddname assigned in the TRN step is SYSPUNCH. Because this file has already been allocated to SYSPUNCH, another way (besides ddname) is needed to pass this file as the input to the compiler. By using PORDER=3, SCLM passes all the files that can be used by the OS COBOL compiler in the order specified for this compiler. To use PORDER=3, a specific parameter string must be built. The language definition must have an FLMALLOC macro for each of these parameters.

    Those FLMALLOCs that are tagged for STEP 22 are not applicable for the Enterprise COBOL compiler. SCLM places 8 bytes of hexadecimal zeros into the ddname substitution list for each FLMALLOC with IOTYPE=N.

Archive

Note: For reference purposes, the language definition shown in Example 13-10 on page 297 contains comments with step numbers placed in the middle of commands; for this language definition to assemble and link into a project definition, these comments must be removed.

Example 13-9 JCL: Invoke CICS Preprocessor and COBOL Compiler

```
//USERIDC  JOB (AS05CR,T12,C531),'USERID',NOTIFY=USERID,CLASS=A,
//    MSGCLASS=0,MSGLEVEL=(1,1)
//*
//*      THIS PROCEDURE CONTAINS 2 STEPS
//*      1.   EXEC THE CICS TS PREPROCESSOR
//*      2.   EXEC THE ENTERPRISE COBOL COMPILER
//*
//*      CHANGE THE JOB NAME AND THE ACCOUNTING INFORMATION TO MEET THE
//*      REQUIREMENTS OF YOUR INSTALLATION.
//*
//*      CHANGE 'PROGNAME' TO THE NAME OF THE CICS/COBOL PROGRAM YOU
//*      WANT TO COMPILE.  CHANGE 'USERID' TO YOUR USERID.
//*
//*      CHANGE 'DEVLEV' TO THE GROUP THAT CONTAINS THE PROGRAM TO BE COMPILED.
//*
//* STEP 1: TRN STATEMENT; STEP 2: EXEC PGM STATEMENT
//*
//TRN    EXEC PGM=DFHECP1$,
//*
//* STEP 3: STEPLIB STATEMENT
//*
//       REGION=2048K
//STEPLIB  DD  DSN=CICS.TS31.CICS.SDFHLOAD,DISP=SHR
//*
//* STEP 4: SYSIN STATEMENT
//*
//SYSIN    DD  DSN=USERID.DEVLEV.SOURCE(PROGNAME),DISP=SHR
//*
//* STEP 5: SYSPRINT STATEMENT
//*
//SYSPRINT DD  SYSOUT=*
//*
//* STEP 6: SYSPUNCH STATEMENT
//*
//SYSPUNCH DD  DSN=&&SYSCIN,
//             DISP=(,PASS),UNIT=SYSDA,
//             DCB=BLKSIZE=0,
//             SPACE=(CYL,(5,2))
//*
//* STEP 7: COB STATEMENT;  STEP 8:  EXEC PGM STATEMENT
//* STEP 9: PARM STATEMENT; STEP 10: COND STATEMENT
//*
//COB    EXEC PGM=IGYCRTCL,REGION=2048K,COND=(4,GT),
//       PARM='NOTRUNC,NODYNAM,LIB,SIZE=256K,BUF=32K,APOST,DMAP,XREF'
//*
//* STEP 11: STEPLIB STATEMENT
```

```
//*
//STEPLIB  DD  DSN=Ecobol.V340.Sigycomp,DISP=SHR
//*
//* STEP 12: SYSLIB STATEMENT; STEP 13: DD STATEMENT CONCATENATION
//*
//SYSLIB   DD  DSN=CICS.TS31.CICS.SDFHCOB,DISP=SHR
//         DD  DSN=CICS.TS31.CICS.SDFHMAC,DISP=SHR
//*
//* STEP 14: SYSPRINT STATEMENT
//*
//SYSPRINT DD  SYSOUT=*
//*
//* STEP 15: SYSIN STATEMENT
//*
//SYSIN    DD  DSN=&&SYSCIN,DISP=(OLD,DELETE)
//*
//* STEP 16: SYSLIN STATEMENT
//*
//SYSLIN   DD  DSN=USERID.DEVLEV.OBJ(PROGNAME),DISP=SHR
//*
//* STEP 17: SYSUT1 STATEMENT
//*
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(5,2))
//*
//* STEP 18: SYSUT2 STATEMENT
//*
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(5,2))
//*
//* STEP 19: SYSUT3 STATEMENT
//*
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(5,2))
//*
//* STEP 20: SYSUT4 STATEMENT
//*
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(5,2))
//*
//* STEP 21: SYSUT5 STATEMENT
//*
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(5,2))
```

Example 13-10 SCLM Language Definition: Invoke CICS Preprocessor and COBOL

```
*********************************************************************
*         SCLM LANGUAGE DEFINITION FOR
*       ENTERPRISE COBOL WITH CICS TS PREPROCESSOR 3.1
*
* CICS TS OUTPUT IS PASSED VIA THE CICSTRAN DD ALLOCATION TO ENTERPRISE COBOL.
*
* POINT THE FLMSYSLB MACRO(S) AT ALL 'STATIC' COPY DATASETS.
* CUSTOMIZE THE 'OPTIONS' AND 'GOODRC' FIELDS TO YOUR STANDARDS.
* ADD THE 'DSNAME' FIELD IF THE TRANSLATOR IS IN A PRIVATE LIBRARY.
* WHEN A NEW TRANSLATOR VERSION REQUIRES TOTAL RECOMPILATION FOR THIS
* LANGUAGE, THE 'VERSION' FIELD ON FLMLANGL SHOULD BE CHANGED.
*********************************************************************
*
```

Archived

```
COBCICS  FLMSYSLB  CICS.TS31.CICS.SDFHCOB
*
FLMLANGL  LANG=COBCICS,VERSION=CICSTS31,ALCSYSLB=Y
*
* PARSER TRANSLATOR
*
          FLMTRNSL   CALLNAM='SCLM COBOL PARSE',                      C
               FUNCTN=PARSE,                                          C
               COMPILE=FLMLPCBL,                                      C
               PORDER=1,                                              C
               OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,)
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
* BUILD TRANSLATORS
*         - CICS PRECOMPILE - STEP NAME TRN
*
* STEP 1
          FLMTRNSL   CALLNAM='CICS PRE-COMPILE',                      C
               FUNCTN=BUILD,                                          C
* STEP 2
               COMPILE=DFHECP1$,                                      C
* STEP 3  (* STEPLIB *)
             DSNAME=CICS.TS31.CICS.SDFHLOAD,                          C
               VERSION=3.1,                                           C
* STEP 10 (* COND *)
               GOODRC=4,                                              C
               PORDER=1
* STEP 4  (* SYSIN *)
           FLMALLOC  IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,          C
               DDNAME=SYSIN
* STEP 5   (* SYSPRINT *)
           FLMALLOC  IOTYPE=O,RECFM=FBA,LRECL=121,                    C
               RECNUM=35000,PRINT=Y,DDNAME=SYSPRINT
*
* STEP 6   (* SYSPUNCH *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,                      C
               RECNUM=5000,DDNAME=SYSPUNCH
*
* STEP 7   (*COBOL INTERFACE - STEP NAME COB *)
* STEP 8
          FLMTRNSL   CALLNAM='COBOL COMPILE',                         C
               FUNCTN=BUILD,                                          C
               COMPILE=IGYCRCTL,                                      C
* STEP 11  (* STEPLIB *)
               DSNAME=ECOBOL.V340.SIGYCOMP,                           C
               VERSION=3.4.0,                                         C
               GOODRC=4,                                              C
* STEP 22
               PORDER=3,                                              C
* STEP 9   (* PARMS *)
               OPTIONS=(NOTRUNC,NODYNAM,LIB,SIZE=256K,BUF=32K,APOST,  C
               DMAP,XREF)* DDNAME ALLOCATIONS
* STEP 16
```

Archived

```
*  1       (* SYSLIN *)
           FLMALLOC  IOTYPE=O,KEYREF=OBJ,RECFM=FB,LRECL=80,            C
               RECNUM=5000,DFLTTYP=OBJ

* STEP 22
*  2       (* N/A *)
           FLMALLOC  IOTYPE=N
* STEP 22
*  3       (* N/A *)
           FLMALLOC  IOTYPE=N
* STEP 12; STEP 13
*  4       (* SYSLIB *)
           FLMALLOC  IOTYPE=I,KEYREF=SINC
* STEP 15
*  5       (* SYSIN *)
           FLMALLOC  IOTYPE=U,KEYREF=SINC,DDNAME=SYSPUNCH
* STEP 14
*  6       (* SYSPRINT *)
           FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=VBA,LRECL=137,         C
               RECNUM=500000,PRINT=Y,DFLTTYP=LIST
* STEP 22
*  7       (* SYSPUNCH *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
* STEP 17
*  8       (* SYSUT1 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* STEP 18
*  9       (* SYSUT2 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* STEP 19
* 10       (* SYSUT3 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* STEP 20
* 11       (* SYSUT4 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* STEP 22
* 12       (* SYSTERM *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
* STEP 21
* 13       (* SYSUT5 *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
* STEP 22
* 14       (* SYSUT6 *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
*
* 5694-A01 (C) COPYRIGHT IBM CORP 1980, 2006
*
```

Archived

Archived

14

Debugging and fault analysis with SCLM

In this chapter we provide information on how to set up SCLM translators to include listing files or side files for the IBM Debug Tool and Fault Analyzer products. There are multiple ways in which to create the listing or side file, and we look at what we regard as the recommended method. However, we also look at other methods to create and store the listing and side files.

Depending on how source code management is done in your shop, this may also have an outcome on how Debug Tool and Fault Analyzer access these files. We look at the different methods of providing the listing or side file location to these tools.

We also look at invoking the debugger in order to complete the picture, from compilation through testing, and we consider invocation in three different environments:

- Testing by invoking directly from TSO
- Testing by invoking through CICS and CEDF
- Testing by invoking and testing and invoking through the WebSphere® debugger

The main focus of this chapter is on how to set up SCLM to use Debug Tool and Fault Analyzer, so we do not delve too deeply into the setup and configuration of these tools. However, we do provide some guidelines and hints on their use.

© Copyright IBM Corp. 2007. All rights reserved.

14.1 Debug Tool

One of the key requirements in application development is the ability to be able to debug your programs. Over the last 15 years things have become much easier with the advent of debugging tools. The IBM offering in this area is IBM Debug Tool. It is therefore important that SCLM be correctly able to utilize the debug options and create the listings and side files required by the debugger to enable successful debugging of your code.

History of SCLM and Debug Tool

Historically SCLM did not interface very well with Debug Tool's predecessor, Code/370. To understand why, you need to understand how and why SCLM works the way it does. When SCLM performs a build, such as a COBOL compile, it creates outputs to temporary OBJ and LISTING datasets. In fact, in your language translators, the FLMALLOC statement information relates to the temporary SCLM datasets with regard to RECNUM and DCB information, not to the permanent SCLM controlled PDS or PDSE where project outputs will be stored. The reason for the build being performed into temporary datasets is so that if the build fails, then SCLM will not copy the invalid build output to the project libraries. If the GOODRC value is not exceeded for the build, then SCLM copies from the temporary data sets it has used, to the permanent project data sets.

For example, when you perform a compile through normal JCL and the compile fails, the OBJ will be empty. If you have a link-edit step that runs immediately after the compile, then the load module generated will be invalid, even though the link-edit may work. Obviously your JCL will have condition code checking to stop this happening. But with SCLM this situation cannot arise, as the last good OBJ will still exist. So if another developer is linking the same load module, then they will still get a valid load module.

However, with normal SCLM processing, this caused a problem with Code/370. The name of the listing file is stored in the object deck and subsequently in the load module. So from an SCLM perspective, this is a temporary z/OS dataset name, such as SYS06160.T111209.RA000.DOHERTL.SYSDEBUG.H01. When Code/370 was invoked, of course, it could not find the listing. SCLM had copied the listing file to a permanent PDS. So the developer had to set the listing to the correct name once they went into Code/370.

How things have changed

Over the years, for many different reasons, Debug Tool has enhanced the ways that a user can tell the tool where to find the listings if the listing name specified in the load module cannot be found. These now include the following possibilities:

- Enter the SET SOURCE command with the name of the location or file.
- Enter the SET DEFAULT LISTINGS command with the name of the location or file.
- Specify the EQADEBUG DD statement with the name of the location or file.
- Code the EQAUEDAT user exit with the new location.

Some of these, such as using the SET SOURCE command, are more manual for the user, but options such as using the EQADEBUG can be integrated into your test procedures or CICS region JCL so that Debug Tool will be able to find the listing automatically.

Other enhancements have included extending the compilers to create a side file that contains the debugging hooks.

So depending on how you specify the location of the listing or side files, this will alter the way you define the language translator information for SCLM.

14.1.1 Setting up your language translators

There are a number of things to consider as an SCLM administrator when setting up your SCLM translators for debugging, such as:

- What compiler options are you going to use?

  For a full explanation or the compiler options needed for debugging, see the relevant guide for your compiler or refer to *Supporting On Demand Business Applications with the IBM Problem Determination Tools (APA, DT, DT with Advanced Facilities, FA, File Export, FM,WS,* SG24-7192. However, throughout the samples, we use the following debug compiler options:

  - TEST(ALL,SYM):

    These options provide maximum debugging capabilities and use the compiler listing to enable stepping through the source code.

  - TEST(ALL,SYM,SEPERATE):

    These options provide maximum debugging capabilities and by using the SEPERATE option (or SEP) the compiler is instructed to store debugging information and symbol tables in a separate file called a side file. The SEPERATE option is available with Enterprise COBOL for z/OS and z/OS and the Enterprise PL/I for z/OS Version 3 Release 5.

- Are you using compiler listings or side files to debug programs?

  SCLM translators normally store the compiler listing in SCLM control libraries. By using compiler options that do not specify the SEPERATE option, these listings will be used by the debugger. However, with the later compilers it is more common to use side files to store the debugging information. A separate debug side file requires an extra FLMALLOC statement.

- Do you intend to use the EQADUBUG DD in your test procedures?

  Depending on how you intend to set up Debug Tool, this will have an effect on how you have to set up your translators to ensure that Debug Tool can find your listings or side files. For example, if you are not going to be using the EQADEBUG DD or the SET DEFAULT LISTINGS, then if you use standard SCLM FLMALLOC statements, your load module will have a temporary data set name imbedded in it and Debug Tool will not find the listing or side file.

Taking these things into consideration, we now look at some examples of setting up language translators that suit your needs.The following sample setups will be performed:

1. Create debug side file under SCLM control using default temporary name.
2. Create debug side file under SCLM control using correct side file name at time of compile.
3. Create debug side file under SCLM control using correct side file, updating the side file name as the member is promoted from group to group.
4. Create debug side file outside of SCLM control.
5. Create listing file under SCLM control using correct listing name at time of compile.

In the following examples we focus mainly on creating debug side files, as this should be considered the preferred approach with the more recent versions of the compilers. However, the process for using listings is exactly the same, except that the SEPERATE option is not used and the SYSDEBUG DD is omitted. There is an example of using listings also to clarify this.

Archived

14.1.2 Option 1: Debug side file under SCLM - default temporary name

This could be described as being the default behavior, and is the easiest to set up from an SCLM language translator point of view. For most implementations of SCLM with Debug Tool this option will work just fine. Using the EQADEBUG DD or SET DEFAULT LISTING is the easiest way to let Debug Tool know where to find the listings.

Note: If you are also going to be using Fault Analyzer, then the preferred way to find the listings may be by use of the EQAUEDAT user exit. This is a user exit that can be coded to give a location of the debug side file based on the location of the load module. For example if the load module is being run from the PROD library, then EQAUEDAT can be set up to get the side file from the PROD library. Fault Analyzer will also run the Debug Tool EQAUEDAT user exit to determine the location of the side files.

As you are storing the Debug Tool side file in SCLM, you have to define the type to SCLM. In your project definition, make sure you have specified a FLMTYPE macro for the side files. In our example, we use SYSDEBUG as the type, so you require an FLMTYPE SYSDEBUG. Allocate the PDS or PDSE that will contain the side files for each level of the hierarchy.

The *Debug Tool Version 7 Configuration Guide* states:

> If the users use COBOL or PL/I separate debug files, verify that the users specify the following attributes for the PDS or PDSE that contains the separate debug files:
>
> – RECFM=FB
> – LRECL=1024
> – BLKSIZE set so that the system determines the optimal size
>
> **Important:** Users must allocate files with the correct attributes to optimize the performance of Debug Tool.

Note: Side files can be created using the TEST SEPERATE option under the following COBOL and PL/I compilers:

- Enterprise COBOL for z/OS and OS/390, Version 3
- COBOL for OS/390 & VM, Version 2 Release 2
- COBOL for OS/390 & VM, Version 2 Release 1 with APAR PQ40298
- Enterprise PL/I for z/OS(R), Version 3 Release 5

SCLM language translator requirements

Next we look at the language translator requirements for both Enterprise COBOL and PL/I. For this example we are compiling a batch program that can be run from a TSO session.

Enterprise COBOL language translator

Taking the build translator part of the default Enterprise COBOL translator as shipped with SCLM. Member FLM@COBE in library ISP.SISPMACS, the first change we make is to modify the compiler options. As shipped, the compiler options are shown in Example 14-1.

Example 14-1 Enterprise COBOL compile options

```
FLMTRNSL   CALLNAM='ENTERPRISE COBOL COMPILER',              C
               FUNCTN=BUILD,                                 C
               COMPILE=IGYCRCTL,                             C
               DSNAME=ECOBOL.V340.SIGYCOMP,                  C
               VERSION=3.1,                                  C
               GOODRC=0,                                     C
               PORDER=1,                                     C
               OPTIONS=(XREF,LIB,APOST,NODYNAM,LIST,NONUMBER,NOSEQ)
```

We change these options as shown in Example 14-2.

Example 14-2 Enterprise COBOL compile options with debug option TEST(ALL,SYM,SEP)

```
FLMTRNSL   CALLNAM='ENTERPRISE COBOL COMPILER',              C
               FUNCTN=BUILD,                                 C
               COMPILE=IGYCRCTL,                             C
               DSNAME=ECOBOL.V340.SIGYCOMP,                  C
               VERSION=3.1,                                  C
               GOODRC=0,                                     C
               PORDER=1,                                     C
               OPTIONS=(XREF,LIB,APOST,NODYNAM,LIST,NONUMBER,NOSEQ,TESTC
               (ALL,SYM,SEP))
```

Next we add an extra SYSDEBUG DD. Depending on if you are using DDNAME substitution, this will alter how you specify the DD. If you are using PORDER=1, so no DDNAME substitution, then add the SYSDEBUG DD at the end of the build translator with the following FLMALLOC statements as shown in Example 14-3.

Example 14-3 New FLMALLOC statement for SYSDEBUG DD

```
          FLMALLOC  IOTYPE=O,DDNAME=SYSDEBUG,KEYREF=OUT1,     C
               RECFM=FB,LRECL=1024,                          C
               RECNUM=50000,PRINT=Y,DFLTTYP=SYSDEBUG
```

Notice the LRECL of 1024 for the SYSDEBUG DD. This matches what you have allocated for the SYSDEBUG type datasets as stated in the *Debug Tool Customization Guide* as the most optimum size for performance.

The completed build steps of the translator are shown in Example 14-4.

Example 14-4 Enterprise COBOL language translator with debug options

```
*
       FLMLANGL    LANG=COBEDT,ALCSYSLB=Y,                   C
               LANGDESC='ENTERPRISE COBOL WITH DEBUG TOOL'
*
*********************************************************************
*        --PARSER TRANSLATOR--                                      *
*********************************************************************
*
          FLMTRNSL   CALLNAM='SCLM COBOL PARSE',             C
               FUNCTN=PARSE,                                 C
               COMPILE=FLMLPCBL,                             C
               PORDER=1,                                     C
               CALLMETH=LINK,                                C
               OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,)
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
*********************************************************************
*         --ENTERPRISE COBOL INTERFACE--                            *
*********************************************************************
*
          FLMTRNSL   CALLNAM='ENTERPRISE COBOL COMPILER',    C
               FUNCTN=BUILD,                                 C
```

```
               COMPILE=IGYCRCTL,                                       C
               DSNAME=ECOBOL.V340.SIGYCOMP,                            C
               VERSION=3.1,                                            C
               GOODRC=0,                                               C
               PORDER=1,                                               C
               OPTIONS=(XREF,LIB,APOST,NODYNAM,LIST,NONUMBER,NOSEQ,TESTC
               (ALL,SYM,SEP))
*
*********************************************************************
*         --DDNAME ALLOCATION--                                     *
*********************************************************************
*
          FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,                 C
              RECNUM=5000,DFLTTYP=OBJ
*
          FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
*
          FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000
*
          FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,RECNUM=5000
*
          FLMALLOC  IOTYPE=W,DDNAME=SYSUT2,RECNUM=5000
*
          FLMALLOC  IOTYPE=W,DDNAME=SYSUT3,RECNUM=5000
*
          FLMALLOC  IOTYPE=W,DDNAME=SYSUT4,RECNUM=5000
*
          FLMALLOC  IOTYPE=W,DDNAME=SYSUT5,RECNUM=5000
*
          FLMALLOC  IOTYPE=W,DDNAME=SYSUT6,RECNUM=5000
*
          FLMALLOC  IOTYPE=W,DDNAME=SYSUT7,RECNUM=5000
*
          FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
           FLMCPYLB NULLFILE
*
          FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
           FLMCPYLB NULLFILE
*
          FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,KEYREF=LIST,              C
              RECFM=FBA,LRECL=133,                                     C
              RECNUM=50000,PRINT=Y,DFLTTYP=LIST
*
          FLMALLOC  IOTYPE=O,DDNAME=SYSDEBUG,KEYREF=OUT1,              C
              RECFM=FB,LRECL=1024,                                     C
              RECNUM=50000,PRINT=Y,DFLTTYP=SYSDEBUG
```

Enterprise PL/I language translator

The translator changes required are the same for Enterprise PL/I as they are for COBOL. Taking the sample Enterprise PL/I translator shipped with ISPF, member FLM@PLIE in the ISP.SISPMAC data set, the changes required to add support for debugging are highlighted in Example 14-5.

Archived

Example 14-5 Enterprise PL/I V3.5 language translator with debug options

```
*
*********************************************************************
*         --PL/I ENTERPRISE INTERFACE--                             *
*********************************************************************
*
          FLMTRNSL   CALLNAM='ENTERPRISE PL/I COMPILER',              C
               FUNCTN=BUILD,                                          C
               COMPILE=IBMZPLI,                                       C
               DSNAME=EPLI.V350.SIBMZCMP,                             C
               VERSION=4.0,                                           C
               GOODRC=0,                                              C
               PORDER=1,                                              C
               OPTIONS=(MACRO,OBJECT,SOURCE,XREF,TEST(ALL,SYM,SEP))
*
*********************************************************************
*         --DDNAME ALLOCATION--                                     *
*********************************************************************
*
 FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,RECNUM=5000,DFLTTYP=OBJ
 FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
 FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,RECNUM=5000
 FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000          @02C
*                                                                2@02D
 FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
  FLMCPYLB NULLFILE
 FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,KEYREF=LIST,DFLTTYP=LIST,          C
               PRINT=Y,RECNUM=5000
*
 FLMALLOC  IOTYPE=O,DDNAME=SYSDEBUG,KEYREF=OUT1,                       C
               RECFM=FB,LRECL=1024,                                    C
               RECNUM=50000,PRINT=Y,DFLTTYP=SYSDEBUG
*                                                                     *
```

Building your program

When you compile your program with this translator, the SCLM messages will just have the return code from the single compile step. However in the build report we can see the new debug file that was created as part of the compile, shown in Figure 14-1.

```
*************************************************************************
    ******* B U I L D   O U T P U T S   G E N E R A T E D *******  Page 1

MEMBER     TYPE       VERSION      KEYWORD
------     ----       -------      -------
PRINTAPP   OBJ              6      OBJ
STARTAPP   OBJ             14
PRINTAPP   LIST             6      LIST
STARTAPP   LIST            14
PRINTAPP   SYSDEBUG         5      OUTX
STARTAPP   SYSDEBUG        12
STARTAPP   LOAD            11      LOAD
STARTAPP   LMAP            11      LMAP
```

Figure 14-1 Build report showing generate debug side files

The new SYSDEBUG file is added by SCLM to the build map or the COBOL source file so that when the source is promoted, the debug file will be promoted with it.

If we look inside the OBJ module, which is link-edited into the LOAD module, we see the reference to the debug side file and the fact that it has the temporary data set name allocated by SCLM, as shown in Figure 14-2.

```
.TXT ..m  ..  ..åø·«ì.j.çO..ì}}.ì\}.q.}..ÚK.J.µ±ì.j.ç...&.J. .J.ì.°*ìO..
.TXT ..ö  ..  ...Õ..DEBUGINF.....+.....ó.............SYS06336.T083010.R
.TXT ...  ..  ..A000.DOHERTL.SYSDEBUG.H01.............IDBA....Ü.......
.TXT ...  ..  ......} ...È....=.... ................................
```

Figure 14-2 OBJ deck showing debug side file name

This is not an issue if you are going to be using the EQADEBUG DD statement or SET DEFAULT LISTING to subsequently allocate the correct SCLM controlled SYSDEBUG datasets to your testing procedures or CICS region JCL

Testing using Debug Tool

As we have used the default processing in SCLM, the side file name specified in the load module is the temporary data set name SCLM used during the build process. This means that when Debug Tool starts up, it will not be able to find the listing. However, with the methods that Debug Tool provides to find the listing, there is no problem in debugging when there is a temporary name in the load module. You can use one of the following methods to point to the listing.

Use the SET DEFAULT LISTING command

This can be specified as you go into Debug Tool for the first time manually, or can be set in your preferences file (INSPPREF) so that when you start Debug Tool it is automatically run. In the above example we are running a standalone COBOL or PL/I program, so you would have a procedure, for example written in REXX that runs the program and allocates the required files, such as any input or output files your program needs, including your Debug Tool preferences file. This is shown in Example 14-6.

Example 14-6 REXX to invoke program with Debug Tool with INSPPREF file

```
/* REXX */

   "ALLOC F(SYSPRINT) DA(*)"
   "ALLOC F(INSPPREF) DA('DOHERTL.INSPPREF')"

   "CALL 'SCLM07.DEV1.LOAD(STARTAPP)' '/TEST'"

   "FREE  F(SYSPRINT)"
   "FREE  F(INSPPREF)"
Exit
```

The file ‘DOHERTL.INSPPREF’ contains any Debug Tool preferences you may have set. In this example we are only setting the SET DEFAULT LISTING command as shown in Figure 14-3.

```
------------------------------------------------------------------------------
EDIT       DOHERTL.INSPPREF                                  Columns 00001 00072
Command ===>                                                   Scroll ===> CSR
****** **************************** Top of Data ***************************
000001  SET DEFAULT LISTING ('SCLM07.DEV1.SYSDEBUG','SCLM07.TEST.SYSDEBUG',
000002                       'SCLM07.PROD.SYSDEBUG')
****** **************************** Bottom of Data ************************
```

Figure 14-3 Debug Tool preferences file

Even though the load module has the SCLM temporary name, Debug Tool will find the correct side file in the concatenation of data sets specified in the SET DEFAULT LISTING command, shown in Figure 14-4.

```
COBOL    LOCATION: STARTAPP initialization
Command ===>                                                 Scroll ===> PAGE
MONITOR --+----1----+----2----+----3----+----4----+----5----+----6 LINE: 0 OF 0
****************************** TOP OF MONITOR *******************************
***************************** BOTTOM OF MONITOR *****************************




SOURCE: STARTAPP -1----+----2----+----3----+----4----+----5----+- LINE: 1 OF 51
******************************** TOP OF SOURCE ********************************
       1 000100* -------------------------------------------------          .
       2 000200*    IDE Sample Program                                      .
       3 000300* -------------------------------------------------          .
       4 000400 Identification Division.                                    .
       5 000500 Program-ID.  StartApp.                                      .
LOG 0----+----1----+----2----+----3----+----4----+----5----+---- LINE: 11 OF 14
0011 *** User preferences file commands follow ***
0012  SET DEFAULT LISTINGS ( 'SCLM07.DEV1.SYSDEBUG', 'SCLM07.TEST.SYSDEBUG',
0013 'SCLM07.PROD.SYSDEBUG' ) ;
0014 *** User preferences file commands end ***
```

Figure 14-4 Starting Debug Tool finds the concatenation of debug side files

Use the EQADEBUG DD specification

Just like the SET DEFAULT LISTING command, the EQADEBUG DD can be used directly from your testing procedures to specify the concatenation sequence of the debug side files. A good example of using the EQADEBUG DD is in CICS region JCL. In this example we see that you can also use the EQADEBUG DD instead of the SET DEFAULT LISTING to determine the search order for the debug side files. The preference file could still be used to set other Debug Tool preferences. The REXX to do this is shown in Example 14-7.

Example 14-7 REXX to invoke program with Debug Tool with EQADEBUG DD

```
/* REXX */

   "ALLOC F(SYSPRINT) DA(*)"
   "ALLOC F(INSPPREF) DA('DOHERTL.INSPPREF')"
   "ALLOC F(EQADEBUG) DA('SCLM07.DEV1.SYSDEBUG',
```

```
                          'SCLM07.TEST.SYSDEBUG',
                          'SCLM07.PROD.SYSDEBUG') SHR"

   "CALL 'SCLM07.DEV1.LOAD(STARTAPP)' '/TEST'"

   "FREE  F(SYSPRINT)"
   "FREE  F(INSPPREF)"
   "FREE  F(EQADEBUG)"
Exit
```

Use the SET SOURCE command

This is the most manual of the ways to locate the listing. It is better to use the SET DEFAULT LISTING, or if you have common procedures that your project uses, have your administrator add the EQADEBUG DD to those.

However, if you are testing small programs, you can either use the SET SOURCE command in your Debug Tool session once the compile unit has started the relevant program by typing the command on the command line or by over-typing the listing name in the LIST panel, or you can use the SET SOURCE command for the required program in your preferences file. The problem with the SET SOURCE command is when you have multiple called programs in your application. You can only use the SET SOURCE command once the compile unit has started. So you cannot use multiple SET SOURCE statements in your preferences file.

Once you have told Debug Tool specifically where to find the listing, you are able to continue debugging as shown Figure 14-5.

```
COBOL    LOCATION: PRINTAPP
Command ===>                                                 Scroll ===> PAGE
MONITOR --+----1----+----2----+----3----+----4----+----5----+----6 LINE: 0 OF 0
****************************** TOP OF MONITOR *******************************
***************************** BOTTOM OF MONITOR *****************************




SOURCE: PRINTAPP -1----+----2----+----3----+----4----+----5----+- LINE: 1 OF 28
******************************** TOP OF SOURCE ********************************
      1 000100 Identification Division.                                        .
      2 000200 Program-ID.  PRINTAPP.                                          .
      3 000300                                                                 .
      4 000400 Data Division.                                                  .
      5 000500 Working-Storage Section.                                        .
LOG 0----+----1----+----2----+----3----+----4----+----5----+---- LINE: 31 OF 34
0031  SET SOURCE ON ( "STARTAPP" ) SCLM07.DEV1.SYSDEBUG(STARTAPP) ;
0032  The debug information for STARTAPP has already been validated, changing
0033 the debug file is not allowed. The command will not be performed.
0034  SET SOURCE ON ( "PRINTAPP" ) SCLM07.DEV1.SYSDEBUG(PRINTAPP) ;
PF  1:?          2:STEP       3:QUIT       4:LIST       5:FIND       6:AT/CLEAR
PF  7:UP         8:DOWN       9:GO        10:ZOOM      11:ZOOM LOG  12:RETRIEVE
```

Figure 14-5 Using the SET SOURCE command in Debug Tool

Use the EQAUEDAT user exit

The EQAUEDAT user exit enables the library administrator or system programmer to direct Debug Tool to the location where source, listing, or separate debug files are stored, and requires knowledge of assembler to modify the location of the side files. The EQAUEDAT user exit works on a compile unit level so examining the sample EQAUEDAT supplied by Debug Tool we see that the code requires the current dataset name to know what to set the new location to. To this end, as the data set name provided to the user exit is the temporary SCLM data set name, the EQAUEDAT user exit is not appropriate in this case. It is possible to modify the user exit to base the location to search for on the compile unit name, but that is effectively coding what the EQADEBUG DD or SET DEFAULT LISTING command is going to give you anyway. However, if you will be using Fault Analyzer, that may be preferable.

So for this option, as the load module will contain the temporary data set name, use of the EQADEBUG DD or the SET DEFAULT LISTING command are more appropriate.

14.1.3 Option 2: Debug side file under SCLM - correct side file name at compile time

This option is very similar to option 1 except that the name of the debug side file is a real data set rather than a temporary dataset name. When the side file that is stored in SCLM is promoted with the source code, OBJ and load module, the name of the debug side file will still show the name of the side file that was created at build time. As, by default, SCLM does not re-build on promote, the name of the side file will not change but the location of the side file will.

The advantages this gives over Option 1 are minimal. It allows you to see a more meaningful name in the OBJ or load module, but this will not change how Debug Tool searches for the file. As you start Debug Tool, if the side file has been promoted, you will still get an error saying Debug Tool cannot find the file. You will still have to use one of Debug Tools methods, as listed previously, to locate the side file.

This method may be the preferred approach if you are also going to be using Fault Analyzer at your site. The reason for this is Fault Analyzer will run Debug Tools EQAUEDAT user exit if it has been set up to change the listing location. By having a real data set name passed to EQAUEDAT, it will be easier for the user exit to set the location of the listing based on the location of the load module that is running. This way you will have a single method, EQAUEDAT, defined for both Debug Tool and Fault Analyzer, making it easier to maintain. This also means that you do not have to put EQADEBUG DDs into all your CICS regions.

There are other reasons why you still might want to set up the SCLM language translators in this way. For example, when you start Debug Tool, it will be easier to specify the SET SOURCE command as in the LIST option as you just have to overtype the group level. Also you may only debug at the development level, so for the modules that you have compiled with debug on, they will always be in the development group. In which case you can step only in those modules with debugging turned on, and because the load module has the correct location of the file, Debug Tool will find the file without having to use the SET DEFAULT LISTING command or EQADEBUG DD.

As stated in 14.1.2, “Option 1: Debug side file under SCLM - default temporary name” on page 304, you must create the data sets that contain the side files and add the FLMTYPE macro for this type to your project definition.

SCLM language translator requirements

We now look at the language translator requirements for both Enterprise COBOL and PL/I. For this example we are compiling a CICS program that runs a CICS BMS Map and reads a DB2 table.

Enterprise COBOL language translator

Take the build translator part of the default Enterprise COBOL translator as shipped with SCLM. For member FLM@COBE in library ISP.SISPMACS, the first change we make is to modify the compiler options. This was shown in Option 1, “Enterprise COBOL language translator” on page 304.

Next we add a SYSDEBUG FLMALLOC to the language translator. This is the main difference between the way we set this up compared to Option 1. We initially specify an FLMALLOC to copy into a dataset that is allocated with an IOTYPE=A. This tells SCLM to copy directly into the PDS or PDSE that is allocated, rather than using a temporary intermediate file. This means that the side file name will be a real PDS or PDSE name in the load module. The FLMALLOC statement uses the SCLM variable @@FLMPRJ, @@FLMGRB, and @@FLMMBR to specify the data set name and member. This is shown in Example 14-8.

Example 14-8 FLMALLOC to a dataset using IOTYPE=A

```
*
           FLMALLOC  IOTYPE=A,DDNAME=SYSDEBUG,DISP=SHR
            FLMCPYLB @@FLMPRJ.@@FLMGRB.SYSDEBUG(@@FLMMBR)
*
```

However, using the IOTYPE=A has a drawback in this instance. The DISP=SHR does not work the same way that DISP=SHR works in JCL. In JCL if the member does not exist, then it will be created. In the FLMALLOC if the member does not exist, then the build will fail. We could have used a DISP=OLD on the FLMALLOC statement, in this case the member would get created. But the problem with this is the DISP=OLD will allocate the dataset exclusively to the user doing the build. So with multiple users there will be contention issues. To this end we need to add an extra step to allocate the file if it does not exist. This step needs to be added before the compile step. The language translator step is shown in Example 14-9.

Example 14-9 Add new SYSDEBUG member translator step

```
************************************************************************
*         --ALLOCATE SYSDEBUG MEMBER IF NEW                            *
************************************************************************
*
         FLMTRNSL   CALLNAM='ALLOC SYSDEBUG',                          C
               FUNCTN=BUILD,                                           C
               COMPILE=SELECT,                                         C
               DSNAME=SCLM.PROJDEFS.REXX,                              C
               CALLMETH=ISPLNK,                                        C
               GOODRC=0,                                               C
               PORDER=1,                                               C
               OPTIONS='CMD(EX ''SCLM.PROJDEFS.REXX(ALLOCDBG)''        C
               ''@@FLMPRJ @@FLMGRP @@FLMMBR'')'
*
```

The REXX called by this step that does the allocate is shown in Example 14-10.

Example 14-10 REXX to allocate a SYSDEBUG file member

```
/* REXX */
   parse upper arg prj grp member                  /* parse arguments */

   dbugdsn = "'"prj"."grp".SYSDEBUG("member")'"

   /* If sysdebug member does not exist then create it                */
   If sysdsn(dbugdsn) <> 'OK' Then
   Do
      "ALLOC DA("dbugdsn") FI(SYSDEBUG) SHR"
      out.0 = 1
      out.1 = "dummy"
      "EXECIO * DISKW SYSDEBUG (STEM OUT. FINIS)"
      If rc > 0 Then
      Do
         Say "******   Error creating the dummy sysdebug member, RC="rc
         Exit 12
      End
      "FREE F(SYSDEBUG)"
   End
Exit 0
```

Finally, to ensure that SCLM knows that the SYSDEBUG file is associated with the source, we need a final build step. This IEBGENERs the generated member onto itself. But by using an IOTYPE=O or P, SCLM will update the build map to associate the SYSDEBUG with the program that was compiled. This step is required as the actual compile step used and IOTYPE=A to place the member in the SCLM SYSDEBUG library, so at this time SCLM does not know about the member. Only IOTYPE=O and P will update the build map. The IEBGENER step is shown in Example 14-11.

Example 14-11 IEBGENER step to associate SYSDEBUG with source

```
*--------------------------------------------------------------------*
*-- REGISTER SYSDEBUG TO SCLM                                     --*
*--------------------------------------------------------------------*
         FLMTRNSL                                                    C
             CALLNAM='REG SYSDEBUG TO SCLM',                         C
             FUNCTN=BUILD,                                           C
             COMPILE=ICEGENER,                                       C
             GOODRC=0,                                               C
             PORDER=3
*  1 --- N/A ---
           FLMALLOC IOTYPE=N
*  2 --- N/A ---
           FLMALLOC IOTYPE=N
*  3 --- N/A ---
           FLMALLOC IOTYPE=N
*  4 --- N/A ---
           FLMALLOC IOTYPE=N
*  5 --- SYSIN ---
           FLMALLOC IOTYPE=A
             FLMCPYLB NULLFILE
*  6 --- SYSPRINT ---
```

Archived

```
          FLMALLOC IOTYPE=A
            FLMCPYLB NULLFILE
*  7 --- N/A ---
          FLMALLOC IOTYPE=N
*  8 --- SYSUT1 (INPUT) ---
          FLMALLOC  IOTYPE=U,DDNAME=SYSDEBUG
*  9 --- SYSUT2 (OUTPUT) ---
          FLMALLOC  IOTYPE=O,DDNAME=OUTDEBUG,KEYREF=OUT2,             C
              RECFM=FB,LRECL=1024,                                    C
              RECNUM=50000,PRINT=Y,DFLTTYP=SYSDEBUG
```

The completed build steps of the translator are shown in Example 14-12.

Example 14-12 Complete Enterprise COBOL translator with debugging turned on

```
CBCID2DT FLMSYSLB  CICS.TS31.CICS.SDFHCOB                  *  @01C*
         FLMSYSLB  CICS.TS31.CICS.SDFHMAC                  *  @01C*
*
         FLMLANGL  LANG=CBCID2DT,VERSION=2,ALCSYSLB=Y,CHKSYSLB=BUILD,  C
               LANGDESC='ENTERPRISE COBOL WITH CICS AND DB2'
*
*********************************************************************
*         --PARSER TRANSLATOR--                                     *
*********************************************************************
*
          FLMTRNSL   CALLNAM='SCLM COBOL PARSE',                     C
               FUNCTN=PARSE,                                         C
               COMPILE=FLMLPCBL,                                     C
               PORDER=1,                                             C
               OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,)      *M34FN*
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)                  *M34FN*
*         - CICS PRECOMPILE -            CODE    LINES   *18@01D*
*                                        COMMENT LINES   *14@01D*
*         --COBOL INTERFACE--            CODE    LINES   *29@01D*
*                                        COMMENT LINES   *18@01D*
*
*********************************************************************
*         --ALLOCATE SYSDEBUG MEMBER IF NEW                         *
*********************************************************************
*
          FLMTRNSL   CALLNAM='ALLOC SYSDEBUG',                       C
               FUNCTN=BUILD,                                         C
               COMPILE=SELECT,                                       C
               DSNAME=SCLM.PROJDEFS.REXX,                            C
               CALLMETH=ISPLNK,                                      C
               GOODRC=0,                                             C
               PORDER=1,                                             C
               OPTIONS='CMD(EX ''SCLM.PROJDEFS.REXX(ALLOCDBG)''      C
               ''@@FLMPRJ @@FLMGRP @@FLMMBR'')'
*
*********************************************************************
* COBOL COMPILE AND CICS PRE-PROCESS      CODE    LINES   *32@01A*
* IN ONE STEP                             COMMENT LINES   *19@01A*
*********************************************************************
```

Archived

```
*
          FLMTRNSL   CALLNAM='COBOL COMPILER WITH CICS PREPROCESS',    C
               FUNCTN=BUILD,                                          C
               COMPILE=IGYCRCTL,                                      C
               VERSION=1.0,                                           C
               TASKLIB=TASKLIB,                                       C
               GOODRC=4,                                              C
               PORDER=1,                                              C
               OPTIONS=(NONUM,LIB,XREF(FULL),MAP,OFFSET,NOOPTIMIZE,SQL,C
               CICS(''COBOL3''),TEST(ALL,SYM,SEP))
*
*********************************************************************
*         --DDNAME ALLOCATION--                                     *
*********************************************************************
*
           FLMALLOC  IOTYPE=O,KEYREF=OBJ,RECFM=FB,LRECL=80,           C
               RECNUM=5000,DFLTTYP=OBJ,DDNAME=SYSLIN
*
           FLMALLOC  IOTYPE=I,KEYREF=SINC,DDNAME=SYSLIB
*
           FLMALLOC  IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,          C
               DDNAME=SYSIN
*
           FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=FBA,LRECL=133,        C
               RECNUM=25000,PRINT=Y,DFLTTYP=LIST,DDNAME=SYSPRINT
*
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,          C
               DDNAME=SYSUT1
*
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,          C
               DDNAME=SYSUT2
*
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,          C
               DDNAME=SYSUT3
*
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,          C
               DDNAME=SYSUT4
*
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,          C
               DDNAME=SYSUT5
*
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,          C
               DDNAME=SYSUT6
*
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,          C
               DDNAME=SYSUT7
*
           FLMALLOC  IOTYPE=P,DDNAME=DBRMLIB,MEMBER=@@FLMONM,         C
               DFLTTYP=DBRM,KEYREF=OUT1,                              C
               RECFM=FB,LRECL=80,RECNUM=5000,DIRBLKS=1
*
           FLMALLOC  IOTYPE=A,DDNAME=SYSDEBUG,DISP=SHR
            FLMCPYLB @@FLMPRJ.@@FLMGRB.SYSDEBUG(@@FLMMBR)
*
*          (* TASKLIB*)
```

Archived

```
           FLMALLOC  IOTYPE=A,DDNAME=TASKLIB
            FLMCPYLB  ECOBOL.V340.SIGYCOMP
            FLMCPYLB  CICS.TS31.CICS.SDFHLOAD
            FLMCPYLB  DB2.V810.SDSNLOAD
            FLMCPYLB  DB2.V810.SDSNEXIT
*                                                                      *
*----------------------------------------------------------------------*
*-- REGISTER SYSDEBUG TO SCLM                                        --*
*----------------------------------------------------------------------*
         FLMTRNSL                                                      C
              CALLNAM='REG SYSDEBUG TO SCLM',                          C
              FUNCTN=BUILD,                                            C
              COMPILE=ICEGENER,                                        C
              GOODRC=0,                                                C
              PORDER=3
*  1 --- N/A ---
           FLMALLOC IOTYPE=N
*  2 --- N/A ---
           FLMALLOC IOTYPE=N
*  3 --- N/A ---
           FLMALLOC IOTYPE=N
*  4 --- N/A ---
           FLMALLOC IOTYPE=N
*  5 --- SYSIN ---
           FLMALLOC IOTYPE=A
            FLMCPYLB NULLFILE
*  6 --- SYSPRINT ---
           FLMALLOC IOTYPE=A
            FLMCPYLB NULLFILE
*  7 --- N/A ---
           FLMALLOC IOTYPE=N
*  8 --- SYSUT1 (INPUT) ---
           FLMALLOC  IOTYPE=U,DDNAME=SYSDEBUG
*  9 --- SYSUT2 (OUTPUT) ---
           FLMALLOC  IOTYPE=O,DDNAME=OUTDEBUG,KEYREF=OUT2,            C
              RECFM=FB,LRECL=1024,                                     C
              RECNUM=50000,PRINT=Y,DFLTTYP=SYSDEBUG
```

Archived

Building your program

When you compile your program with this translator, the SCLM messages will have return codes from the three build steps as shown in Figure 14-6. It should be noted that even though this is a COBOL/CICS/DB2 program, we have used the single step Enterprise COBOL compiler so there are no CICS or DB2 precompile steps.

```
FLM49000 - INVOKING BUILD PROCESSOR
FLM09002 - THE BUILD REPORT WILL APPEAR IN DOHERTL.BUILD.REPORT68
FLM09006 - THE BUILD LISTING WILL APPEAR IN DOHERTL.BUILD.LIST68
FLM42000 - BUILD PROCESSOR INITIATED - 03:40:06 ON 2006/12/07
FLM44500 - >> INVOKING BUILD TRANSLATOR(S) FOR TYPE: COBOL    MEMBER: RDBKC01
FLM06501 - TRANSLATOR RETURN CODE FROM ===> ALLOC SYSDEBUG       ===> 0
FLM06501 - TRANSLATOR RETURN CODE FROM ===> COBOL COMPILER W     ===> 4
FLM06501 - TRANSLATOR RETURN CODE FROM ===> REG SYSDEBUG TO      ===> 0
FLM46000 - BUILD PROCESSOR COMPLETED - 03:40:12 ON 2006/12/07
FLM09008 - RETURN CODE = 0
```

Figure 14-6 Build messages showing all three translator steps

The build report shows the new debug file that was created as part of the compile, shown in Figure 14-7.

```
*************************************************************************
    ******* B U I L D   O U T P U T S   G E N E R A T E D *******  Page 1

MEMBER     TYPE       VERSION     KEYWORD
------     ----       -------     -------
RDBKC01    OBJ             11     OBJ
RDBKC01    LIST            11     LIST
RDBKC01    DBRM            11     OUTX
RDBKC01    SYSDEBUG         4
```

Figure 14-7 Build report showing generate debug side files

The new SYSDEBUG file is added by SCLM to the build map or the COBOL source file so that when the source is promoted, the debug file will be promoted with it.

If we look inside the OBJ module, which is link-edited into the LOAD module, we see that the reference to the debug side file has the current location of the file, as shown in Figure 14-8.

```
.TXT ..\  ..  ..}.ï\}.q.}..ÚK.Jqs]ï.j.ç...&.Ju .Jqï.°*ïO...ÕDEBUGINF....
.TXT ...  ..  ... ...................SCLM07.DEV1.SYSDEBUG(RDBKC01).....
.TXT ..*  ..  ..IDBA....Ü.............  ...q ........=....*..............
```

Figure 14-8 OBJ deck showing debug side file name

This is not an issue if you are going to be using the EQADEBUG DD statement to subsequently allocate the correct SCLM controlled SYSDEBUG datasets to your testing procedures or CICS region JCL.

Archived

Testing using Debug Tool

The same methods of testing this option exist for testing option 1. However, if you only do your debugging at development, for example, then you may not have to specify the SET DEFAULT LISTING or EQADEBUG DD, as the side file will be in the location that is specified in the load module. This is the only advantage over option 1.

If you keep your side file with the source as it is promoted through the hierarchy, then the same problems will exist as with option 1. If the load and subsequently the side file are now in PROD and you are trying to debug, then Debug Tool will not find the listing. The load will specify the side file as SCLM07.DEV1.SYSDEBUG, but the side file will be in SCLM07.PROD.SYSDEBUG once it is promoted. To this end, either the EQADEBUG DD should be used in the CICS region JCL as shown in Figure 14-9, or the user should use the SET DEFAULT LISTING when they go into Debug Tool, or through their INSPPREF file.

```
//*
//*        THE DUMP DATASETS
//*
//DFHDMPA  DD   DISP=SHR,DSN=CICSTS31.C64C1IS1.DFHDMPA
//DFHDMPB  DD   DISP=SHR,DSN=CICSTS31.C64C1IS1.DFHDMPB
//SYSABEND DD   SYSOUT=&OUTC
//SYSPRINT DD   SYSOUT=&OUTC
//PRINTER  DD   SYSOUT=&OUTC,DCB=BLKSIZE=121
//*
//*        THE CICS SYSTEM DEFINITION DATASET
//*
//DFHCSD   DD   DISP=SHR,DSN=CICSTS31.C64C1IS1.DFHCSD
//*
//*        SPECIFY EQADEBUG SEARCH ORDER
//*
//EQADEBUG DD   DISP=SHR,DSN=SCLM07.DEV1.SYSDEBUG
//         DD   DISP=SHR,DSN=SCLM07.TEST.SYSDEBUG
//         DD   DISP=SHR,DSN=SCLM07.PROD.SYSDEBUG
//*
```

Figure 14-9 Extract from CICS region JCL showing EQADEBUG DD allocation

When the CICS transaction is tested using CEDF, the side file will be found by searching the EQADEBUG DD specified in the region JCL.

Using this method where the load module will contain a real data set name may make using the EQAUEDAT easier, especially if your site uses Fault Analyzer in addition to Debug Tool. If this is the case, the EQAUEDAT exit could be coded such that it takes the location of the side file passed in and substitutes all but the last level qualifier (DEBUG in our example) with all but the last level qualifier of the load library location.

For example, if the load module is being run from SCLM07.PROD.LOAD and the side file location specified in the load module is SCLM07.DEV1.DEBUG, then EQAUEDAT can be coded so that the side file location is set up as SCLM07.PROD.DEBUG. If SCLM has promoted the side file with the source and load module, then this is where the correct side file will exist for the PROD version of the load module. Example 14-13 shows a sample EQAUEDAT exit that uses all but the last qualifier of the LOADLIB data set name and appends the last qualifier and member name of the original SYSDEBUG data set. This example may be more appropriate in an SCLM environment than the sample shipped with Debug Tool.

Archived

Example 14-13 Sample EQAUEDAT exit

```
*********************************************************************
*                                                                   *
* MODULE NAME = EQAUEDAT                                            *
*                                                                   *
* Sample code to change last qualifier of the load DSN to the       *
* last qualifier of the sysdebug DSN and use the new DSN as the     *
* modified sysdebug DSN                                             *
*  E.g. DEV1.APPL1.LOAD  & TEST.APP1TST.SYSDEBUG(MYAPP)             *
*  giving                                                           *
*       DEV1.APPL1.SYSDEBUG(MYAPP)                                  *
*                                                                   *
*  This code expects the resultant DSN to be 44 char or less.       *
*/*****************************************************************/
*
*        Symbolic Register Definitions and Usage
*
R0       EQU   0             Work register
R1       EQU   1             Parameter list address (upon entry)
R2       EQU   2             Work register - DSN Length
R3       EQU   3             Work register
R4       EQU   4             Address of data set name
R5       EQU   5             Work register
R6       EQU   6             Work register
R7       EQU   7             Work register
R8       EQU   8             Work register
R9       EQU   9             Parameter list address (after CEEENTRY)
R10      EQU   10            Work register
R11      EQU   11            Base register for executable code
R12      EQU   12            Common Anchor Area address
R13      EQU   13            Save Area/Dynamic Storage Area address
R14      EQU   14            Return point address
R15      EQU   15            Entry point address
*
*        Language Codes
*
LANPLI   EQU   1             PL/I
LANCOBOL EQU   2             COBOL
LANC     EQU   4             C
LANASSEM EQU   5             Assembler
*
*        Prologue
*
EQAUEDAT CEEENTRY AUTO=DSASIZ,    Amount of main memory to obtain      *
               PPA=PPA3,          Program Prolog Area for this routine *
               MAIN=NO,           This program is a Subroutine         *
               NAB=NO,            NO- not called from LE-enabled pgm   *
               PARMREG=R9,        R1 value is saved here               *
               BASE=R11           Base register for executable code,
*                                      constants, and static variables
         USING CEECAA,R12         Common Anchor Area addressability
         USING CEEDSA,R13         Dynamic Storage Area addressability
         USING PARMLIST,R9
*
* First scan down debug dataset name looking for last .
```

Archived

```
*
*
         L     R3,DSETNM@          Addr of sysdebug dsn
         L     R3,0(R3)
         L     R6,DSETLEN          Addr of sysdebug dsn length
         L     R2,0(R6)

         LR    R4,R3               Remember start address
         LR    R5,R2               Remember Length

         MVC   LLQPos,=F'0'

NextChar CLI   0(R3),C'.'          Now scan for length for last .
         BNE   NoLLQUpd            Arguably more efficient to
         ST    R3,LLQPos           do in reverse.
NoLLQUpd DS    0H
         LA    R3,1(,R3)
         BCT   R2,NextChar
*
*  We now have
*
* .-----------R5-----------.
* V                        V
* USERID.THIS.DATASET(MEMBER)
* |          |
* |          .------------------LLQPos
* .----------------------------R4
*
         L     R7,LLQPos
         ST    R7,LLQPosSysdebug   Remember start address
         SR    R7,R4
         SR    R5,R7
         ST    R5,LLQLenSysdebug   Length of last qualifier and (mm)

*
* No scan load data set name for last .
*

         L     R3,LDLBMN@          Addr of load dsn
         L     R3,0(R3)
         L     R6,LDLBMNLN         Addr of load dsn length
         L     R2,0(R6)


         LR    R4,R3               Remember start address
         LR    R5,R2
         MVC   LLQPos,=F'0'

NextCh1  CLI   0(R3),C'.'
         BNE   NoLLQUp1
         ST    R3,LLQPos
NoLLQUp1 DS    0H
         LA    R3,1(,R3)
         BCT   R2,NextCh1
*
```

Archived

```
*  We now have
*
*
* .-------R5--------.
* V                 V
* USERID.THIS.LOADLIB
* |          |
* |          .------------------LLQPos
* .----------------------------R4
*
         L     R7,LLQPos
         SR    R7,R4
         ST    R7,LLQLen            Length up to last .

         L     R10,LLQPos           Now move last qualifier and
         L     R7,LLQPosSysdebug    member name (if there is one)
         L     R5,LLQLenSysdebug    from sysdebug dsn over the
         EX    R5,MoveMem           top of the load dsn

         L     R5,LLQLenSysdebug    calculate new combined length
         L     R7,LLQLen
         AR    R5,R7

         L     R6,DSETLEN          Store new length
         ST    R5,0(R6)

         L     R4,LDLBMN@          We now need to copy the newly
         L     R7,0(R4)            formed dsn back over the original
*                                  sysdebug dsn.
         L     R4,DSETNM@
         L     R10,0(R4)

         EX    R5,MoveMem
*
*        Epilogue
*
Exit     DS    0H
         CEETERM RC=0
*
PPA3     CEEPPA ,                  Program Prolog Area for this routine
*
         LTORG ,                   Place literal pool here
         EJECT
MoveMem  MVC   0(0,R10),0(R7)
*
*        Map the Dynamic Storage  Area (DSA)
*
         CEEDSA ,            Map standard CEE DSA prologue
*
*        Local Automatic (Dynamic) Storage..
*
LLQPosSysdebug   DS    F
LLQLenSysdebug   DS    F
LLQPos           DS    F
LLQLen           DS    F
```

Archived

```
DSASIZ           EQU   *-CEEDSA      Length of DSA
         EJECT
*
*        Map the Common Anchor Area (CAA)
*
         CEECAA
*
*        The map of parameters
*
PARMLIST DSECT
DSETNM@  DS    A                  Debug Data dataset name string addr
DSETLEN  DS    A                  Debug Data dataset name len
LANGCODE DS    A                  Language Code
CUNM@    DS    A                  CU name string address
CUNMLEN  DS    A                  CU name length
LDMDMN@  DS    A                  Load module name string address
LDMDMNLN DS    A                  Load module name string length
LDLBMN@  DS    A                  Load library name string address
LDLBMNLN DS    A                  Load library name string length
         END   ,                  of EQAUEDAT
```

For completeness of this test scenario, we can test the transaction in CICS. To use Debug Tool from CICS, you must set up a debug profile using transaction CADP or DTCN in CICS. A typical setup is shown in Figure 14-10 and Figure 14-11.

```
CADP      -    CICS Application Debugging Profile Manager   -     C64C1IS1

List Debugging Profiles    (A=Activate,I=Inactivate,D=Delete,C=Copy)

  Owner    Profile  S Tran Program  Compile Unit Applid   Userid   Term Type
_ $EXAMPLE COMP1    I T*   P*       *            CICSREG1 PANDREWS TTT1 Comp
_ $EXAMPLE COMP2    I TR   *        SAMPCOMPUN + CICSREG2 DRBEARD* TTT2 Comp
_ $EXAMPLE COMP3    I TRN3 PROG3    *            CICSREG3 *        TTT2 Comp
_ $EXAMPLE CORBA    I T*                         *        IORWERTH      Corb
_ $EXAMPLE EJB      I *                          *        *             EJB
_ $EXAMPLE JAVA     I TR*                        *        PENFOLD*      Java
_ CICSUSER RDBK     I TST1 *        *            C64C1IS1 *        *    Comp
_ CICSUSER RDBKWD4Z A TST1 *        *            C64C1IS1 *        *    Comp









8   profile(s). All profiles shown
Enter=Process PF1=Help 2=Filter 3=Exit 4=View 5=Create Comp 6=Create Java
                9=Set display device 10=Edit 11=Sort
```

Figure 14-10 CADP transaction showing RDBK profile to use Debug Tool under CICS

```
CADP      -    CICS Application Debugging Profile Manager   -     C64C1IS1

Set Compiled Debugging Display Device

Debugging Display Device
Session Type             ==>  3270                        (3270,TCP)
3270 Display Terminal    ==>  1497

TCP/IP Name Or Address
==>
==>
==>
==>
Port                     ==>  08001

Type of socket communication ==> Single                   (Single,Multiple)


Display this panel on LE profile activation ==> YES




Enter=Save and return PF1=Help 3=Exit 12=Cancel
```

Figure 14-11 CADP transaction settings for CICS 3270 testing with Debug Tool

To test using CEDF and Debug Tool, you must use 2 terminals. For example, we have started 2 CICS sessions that are assigned terminal IDs 1290 and 1292. The following sequence takes place:

1. Terminal 1292:

   Clear panel and type **`CEDF 1290,on,i`** and press Enter.

   The following message is then displayed on the panel:

   `TERMINAL 1290: OPTION I... REQUESTS THE DEBUG TOOL : EDF MODE ON. DEBUG TOOL ON.`

2. Terminal 1290:

   Clear panel and type transaction name, in our example TST1, press Enter.

3. Terminal 1292:

   EDF session will start, step through EDF until it terminates, at which time control is passed to Debug Tool, which is running on terminal 1290.

4. Terminal 1290:

   Debug Tool starts up with the source code.

14.1.4 Option 3: Debug side file under SCLM - correct side file name at each group

If you do not want to utilize the EQADEBUG DD, SET DEFAULT LISTING, or EQAUEDAT functionality in Debug Tool and you insist that the side file name specified in the LOAD module points to the correct location at all times, then this is possible with SCLM.

Archived

SCLM language translator requirements

This option is very similar to option 2 in its setup. As the correct name of the side file needs to be placed in the OBJ and subsequently the load module the same techniques are used as shown in option 2 previously:

- We use the same REXX step to create the member in the SYSDEBUG file if it does not already exist.
- We use the same compile step with IOTYPE=A to write to the side file member directly to the SYSDEBUG data set, ensuring that the correct name is in the OBJ.
- We use the same IEBGENER step to register the SYSDEBUG file to SCLM.

The only thing we need to change in the translator is the addition of the FLMLRBLD macro. This macro tells SCLM to rebuild the source when it promotes, if the group is listed in the FLMLRBLD macro. So using the translator specified in option 2, the location of the debug side file will be current in the OBJ and load module. The addition of the FLMLRBLD macro is shown in Example 14-14. This is the only change to the translator shown in Figure 14-12 on page 314.

Example 14-14 Addition od FLMLRBLD to language translator

```
CBCID2DT FLMSYSLB  CICS.TS31.CICS.SDFHCOB                *  @01C*
         FLMSYSLB  CICS.TS31.CICS.SDFHMAC                *  @01C*
*
         FLMLANGL  LANG=CBCID2DT,VERSION=2,ALCSYSLB=Y,CHKSYSLB=BUILD,  C
               LANGDESC='ENTERPRISE COBOL WITH CICS AND DB2'
*
         FLMLRBLD  GROUP=(TEST,PROD)
*
```

Building your program

When the module is built at the development level, it will have the same results as shown in Example 14-6 on page 317. The main difference is when you promote the member to the next level in the hierarchy. On the promote to TEST, SCLM will rebuild the COBOL member and therefore relink the load module. The same will happen on the promote to PROD. The build messages from the promote to PROD are shown in Figure 14-12. You can see that after the promote, the build phase is executed again for the COBOL member we changed. In this case, the binds at the PROD level are done also.

Archive

```
FLM59001 - INVOKING PROMOTE PROCESSOR
FLM09002 - THE PROMOTE REPORT WILL APPEAR IN DOHERTL.PROMOTE.REPORT81
FLM09002 - THE BUILD REPORT WILL APPEAR IN DOHERTL.BUILD.REPORT81
FLM09006 - THE BUILD LISTING WILL APPEAR IN DOHERTL.BUILD.LIST81
FLM51000 - PROMOTE PROCESSOR INITIATED - 09:10:09 ON 2006/12/07
FLM52000 - INITIATING VERIFICATION PHASE - 09:10:09 ON 2006/12/07

FLM55000 - INITIATING COPY PHASE - 09:10:10 ON 2006/12/07
FLM57000 - INITIATING PURGE PHASE - 09:10:13 ON 2006/12/07
FLM57001 - INITIATING PURGE FROM GROUP: TEST

FLM42000 - BUILD PROCESSOR INITIATED - 09:10:16 ON 2006/12/07
FLM44500 - >> INVOKING BUILD TRANSLATOR(S) FOR TYPE: COBOL    MEMBER: RDBKC01
FLM06501 - TRANSLATOR RETURN CODE FROM ===> ALLOC SYSDEBUG       ===> 0
FLM06501 - TRANSLATOR RETURN CODE FROM ===> COBOL COMPILER W     ===> 4
FLM06501 - TRANSLATOR RETURN CODE FROM ===> REG SYSDEBUG TO      ===> 0
FLM44500 - >> INVOKING BUILD TRANSLATOR(S) FOR TYPE: ARCHBIND MEMBER: RDBKC01

FLM06501 - TRANSLATOR RETURN CODE FROM ===> DB2 PACKAGE BIND     ===> 0
FLM44500 - >> INVOKING BUILD TRANSLATOR(S) FOR TYPE: ARCHLEC  MEMBER: RDBKC01
FLM06501 - TRANSLATOR RETURN CODE FROM ===> LKED/370             ===> 0
FLM46000 - BUILD PROCESSOR COMPLETED - 09:13:54 ON 2006/12/07
FLM87107 - BUILD SUCCEEDED FOR MEMBER RDBKC01 AT 09:13:54, CODE: 0
FLM58000 - PROMOTE PROCESSOR COMPLETED - 09:13:54 ON 2006/12/07
FLM09008 - RETURN CODE =  0
```

Figure 14-12 Promote messages with FLMLRBLD macro active

If we look inside the OBJ module, which is link-edited into the LOAD module, we see that the reference to the debug side file now has the PROD location of the side file thus as shown in Figure 14-13.

```
.TXT ..\  ..  ..}.ì\}.q.}..ÚK.Jqs]ì.j.ç...&.Ju .Jqì.°*ìO...ÕDEBUGINF....
.TXT ...  ..  ... ..................SCLM07.PROD.SYSDEBUG(RDBKC01).....
.TXT ..*  ..  ..IDBA....Ü.......... ...q ........=....*..............
```

Figure 14-13 OBJ deck showing debug side file name

Testing using Debug Tool

In this case, when you test with Debug Tool, you do not need to use the EQADEBUG DD or SET DEFAULT LISTING command, as the current location of the side file is always current in the load module.

This method should only be used if you have a real requirement to rebuild at each level in the hierarchy. To have the “current” debug file location in the load module is not reason in itself, as it is easy to provide the listing location via the EQADEBUG or SET DEFAULT LISTING command. With source code management, it is better to test what has been developed and then promote what has been tested all the way to PROD, as you do not know 100% what a rebuild at the different levels may drag in to your load module.

14.1.5 Option 4: Debug side file created outside of SCLM control

This option is the same as option 2 or option 3, depending on whether you have one debug side file for all levels of the hierarchy or one per level of the hierarchy. The only difference is that the last IEBGENER step that registers the SYSDEBUG file to SCLM is not included in the language translator.

The building of the member is the same as in option 2 or option 3, except for this difference. The SYSDEBUG file, even though created, will not be added to the build map, so it will not be promoted, as it is not part of the objects that SCLM manages.

Testing this option with Debug Tool is again the same as option 2 or option 3, because Debug Tool does not care that the side file is or is not under SCLM control — just where the side file is located. So if you have set up this translator to rebuild at each level with FLMLRBLD, then you do not require the EQADEBUG DD or SET DEFAULT LISTING statement; otherwise you can do so, depending on where the listing is stored in relation to the load module.

If you are going to create a side file, then it should be under SCLM control.

14.1.6 Using compile listing file instead of side file

The above examples have all dealt with debug side files, as this has become the recommended practice for setting up debugging with Debug Tool. However, it is still possible to use the compile listing with Debug Tool. The process of setting up the translator to use the listing is almost identical to setting it up to use the side file. We take option 1 as an example.

SCLM language translator requirements

In this case the compiler options will be different. To use the listing instead of the side file, just remove the SEPERATE option. So the compiler options will look like Example 14-15.

Example 14-15 Setting the debug compiler options without SEPERATE

```
*********************************************************************
*        --ENTERPRISE COBOL INTERFACE--                             *
*********************************************************************
*
         FLMTRNSL   CALLNAM='ENTERPRISE COBOL COMPILER',             C
              FUNCTN=BUILD,                                          C
              COMPILE=IGYCRCTL,                                      C
              DSNAME=ECOBOL.V340.SIGYCOMP,                           C
              VERSION=3.1,                                           C
              GOODRC=0,                                              C
              PORDER=1,                                              C
              OPTIONS=(XREF,LIB,APOST,NODYNAM,LIST,NONUMBER,NOSEQ,TESTC
              (ALL,SYM))
*
```

The FLMALLOC of the listing dataset will be the same as the default Enterprise COBOL translator as shipped with SCLM. Member FLM@COBE in library ISP.SISPMACSI.

Building your program

Building the program is no different than before, except that there is no SYSDEBUG file created, as the hooks in the load will use the compiler listing file. The build listing returned by SCLM is shown in Figure 14-14, where we see the outputs created by the build.

```
********************************************************************************
    ******* B U I L D   O U T P U T S   G E N E R A T E D *******  Page 1

MEMBER     TYPE       VERSION      KEYWORD
------     ----       -------      -------
PRINTAPP   OBJ              9      OBJ
STARTAPP   OBJ             17
PRINTAPP   LIST             9      LIST
STARTAPP   LIST            17
STARTAPP   LOAD            14      LOAD
STARTAPP   LMAP            14      LMAP
```

Figure 14-14 Build list when listing used over side file

If we look in the generated OBJ again, we see an SCLM temporary dataset name, this time referring to the compiler listing as shown in Figure 14-15.

```
.TXT ...  ..  .............@...°...%.........................SYSPRINT
.TXT ..È  ..  ..SYS06341.T151802.RA000.DOHERTL.SYSPRINT.H01
.TXT ..Ð  ..  ..       PP 5655-  LineIDïî|&êë-/ÄÃÃÅ.............Ü....
```

Figure 14-15 OBJ showing temporary SCLM listing file

Testing using Debug Tool

Testing programs compiled to use the listing over the side file are really no different. When using the normal Debug Tool options for locating the listing file, you use the same format as in option 1, except substitute the location of the side file, SYSDEBUG, for the location of the compiler listing, LIST. So taking the SET DEFAULT LISTING command set in the preferences file as an example, we set the preferences file as shown in Figure 14-16.

```
--------------------------------------------------------------------------------
EDIT       DOHERTL.INSPPREF                                 Columns 00001 00072
Command ===>                                                   Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001  SET DEFAULT LISTING ('SCLM07.DEV1.LIST','SCLM07.TEST.LIST',
000002                       'SCLM07.PROD.LIST')
****** **************************** Bottom of Data ****************************
```

Figure 14-16 Debug Tool preferences file showing listing use over side file

Even though the load module has the SCLM temporary name for the listing, Debug Tool will find the correct listing in the concatenation of data sets specified in the SET DEFAULT LISTING command, shown in Figure 14-17.

Archive

```
COBOL    LOCATION: STARTAPP initialization
Command ===>                                                Scroll ===> PAGE
MONITOR --+----1----+----2----+----3----+----4----+----5----+----6 LINE: 0 OF 0
****************************** TOP OF MONITOR *******************************
****************************** BOTTOM OF MONITOR ****************************




SOURCE: STARTAPP -1----+----2----+----3----+----4----+----5----+- LINE: 1 OF 51
****************************** TOP OF SOURCE ********************************
       1 000100* ------------------------------------------------------        .
       2 000200*    IDE Sample Program                                         .
       3 000300* ------------------------------------------------------        .
       4 000400 Identification Division.                                       .
       5 000500 Program-ID.  StartApp.                                         .
LOG 0----+----1----+----2----+----3----+----4----+----5----+---- LINE: 11 OF 14
0011 *** User preferences file commands follow ***
0012  SET DEFAULT LISTINGS ( 'SCLM07.DEV1.LIST', 'SCLM07.TEST.LIST',
0013 'SCLM07.PROD.LIST' ) ;
0014 *** User preferences file commands end ***
PF  1:?          2:STEP       3:QUIT       4:LIST       5:FIND       6:AT/CLEAR
PF  7:UP         8:DOWN       9:GO        10:ZOOM      11:ZOOM LOG  12:RETRIEVE
```

Figure 14-17 Starting Debug Tool finds the concatenation of listings

Considerations when temporary name in load is not acceptable

If you are going to be using option 2 or 3 to set up debugging, but wish to use the listing instead of the side file, then you can just change the REXX that allocates the new member, as shown in Example 14-16, and change the SYSDEBUG to be the LIST data set.

Example 14-16 REXX to allocate a LIST file member

```
/* REXX */
   parse upper arg prj grp member                  /* parse arguments */

   listdsn = "'"prj"."grp".LIST("member")'"

   /* If sysdebug member does not exist then create it                */
   If sysdsn(listdsn) <> 'OK' Then
   Do
      "ALLOC DA("listdsn") FI(LIST) SHR"
      out.0 = 1
      out.1 = "dummy"
      "EXECIO * DISKW LIST (STEM OUT. FINIS)"
      If rc > 0 Then
      Do
         Say "******   Error creating the dummy list member, RC="rc
         Exit 12
      End
      "FREE F(LIST)"
   End
Exit 0
```

You can then remove the SYSDEBUG allocations and replace with allocations for the LIST type. In this case you have to pay attention to the compiler expected DCB requirements for the SYSPRINT dataset when setting up the translator. The complete translator is shown in Example 14-17.

Example 14-17 Enterprise COBOL translator with debugging turned on using listing

```
FLMLANGL    LANG=COBEDT1L,ALCSYSLB=Y,                          C
               LANGDESC='ENTERPRISE COBOL WITH DEBUG TOOL'
*
*********************************************************************
*         --PARSER TRANSLATOR--                                     *
*********************************************************************
*
          FLMTRNSL   CALLNAM='SCLM COBOL PARSE',                     C
               FUNCTN=PARSE,                                         C
               COMPILE=FLMLPCBL,                                     C
               PORDER=1,                                             C
               CALLMETH=LINK,                                        C
               OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,)
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
*********************************************************************
*         --ALLOCATE SYSDEBUG MEMBER IF NEW                         *
*********************************************************************
*
         FLMTRNSL   CALLNAM='ALLOC SYSDEBUG',                        C
               FUNCTN=BUILD,                                         C
               COMPILE=SELECT,                                       C
               DSNAME=SCLM.PROJDEFS.REXX,                            C
               CALLMETH=ISPLNK,                                      C
               GOODRC=0,                                             C
               PORDER=1,                                             C
               OPTIONS='CMD(EX ''SCLM.PROJDEFS.REXX(ALLOCLST)''      C
               ''@@FLMPRJ @@FLMGRP @@FLMMBR'')'
*
*
*********************************************************************
*         --ENTERPRISE COBOL INTERFACE--                            *
*********************************************************************
*
          FLMTRNSL   CALLNAM='ENTERPRISE COBOL COMPILER',            C
               FUNCTN=BUILD,                                         C
               COMPILE=IGYCRCTL,                                     C
               DSNAME=ECOBOL.V340.SIGYCOMP,                          C
               VERSION=3.1,                                          C
               GOODRC=0,                                             C
               PORDER=1,                                             C
               OPTIONS=(XREF,LIB,APOST,NODYNAM,LIST,NONUMBER,NOSEQ,TESTC
               (ALL,SYM))
*
*********************************************************************
*         --DDNAME ALLOCATION--                                     *
*********************************************************************
```

Archived

```
*
          FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,                C
              RECNUM=5000,DFLTTYP=OBJ
*
          FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
*
          FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000
*
          FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,RECNUM=5000
*
          FLMALLOC  IOTYPE=W,DDNAME=SYSUT2,RECNUM=5000
*
          FLMALLOC  IOTYPE=W,DDNAME=SYSUT3,RECNUM=5000
*
          FLMALLOC  IOTYPE=W,DDNAME=SYSUT4,RECNUM=5000
*
          FLMALLOC  IOTYPE=W,DDNAME=SYSUT5,RECNUM=5000
*
          FLMALLOC  IOTYPE=W,DDNAME=SYSUT6,RECNUM=5000
*
          FLMALLOC  IOTYPE=W,DDNAME=SYSUT7,RECNUM=5000
*
          FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
           FLMCPYLB NULLFILE
*
          FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
           FLMCPYLB NULLFILE
*
          FLMALLOC  IOTYPE=A,DDNAME=SYSPRINT,DISP=SHR
           FLMCPYLB @@FLMPRJ.@@FLMGRB.LIST(@@FLMMBR)
*
*-------------------------------------------------------------------*
*-- REGISTER LISTING TO SCLM                                      --*
*-------------------------------------------------------------------*
        FLMTRNSL                                                     C
             CALLNAM='REG LISTING TO SCLM',                          C
             FUNCTN=BUILD,                                           C
             COMPILE=ICEGENER,                                       C
             GOODRC=0,                                               C
             PORDER=3
*  1 --- N/A ---
          FLMALLOC IOTYPE=N
*  2 --- N/A ---
          FLMALLOC IOTYPE=N
*  3 --- N/A ---
          FLMALLOC IOTYPE=N
*  4 --- N/A ---
          FLMALLOC IOTYPE=N
*  5 --- SYSIN ---
          FLMALLOC IOTYPE=A
            FLMCPYLB NULLFILE
*  6 --- SYSPRINT ---
          FLMALLOC IOTYPE=A
            FLMCPYLB NULLFILE
*  7 --- N/A ---
```

```
          FLMALLOC IOTYPE=N
*  8 --- SYSUT1 (INPUT) ---
          FLMALLOC  IOTYPE=U,DDNAME=SYSPRINT
*  9 --- SYSUT2 (OUTPUT) ---
          FLMALLOC  IOTYPE=O,DDNAME=OUTLIST,KEYREF=OUT1,              C
              RECFM=FBA,LRECL=121,                                    C
              RECNUM=50000,PRINT=Y,DFLTTYP=LIST
*                                                                    *
```

14.1.7 Invoking the debugger to run on your workstation using WebSphere debugger

Debug Tool now also interfaces with remote debuggers such as the WebSphere for System z debugger. From an SCLM perspective, the setup is no different than has been previously described in the options above. Whether to invoke the WebSphere debugger for your program will vary on whether your program is a batch or CICS program, for example.

Turning debugging on in the WD/z client

Before you can start to debug any programs remotely, you have to turn the remote debugging option on in the WD/z client. This is the same action if you are debugging CICS or batch programs:

1. Start WD/z and make sure you have an active Remote Systems Explorer (RSE) connection to the z/OS host that you are running the host application you are going to debug.
2. Turn on the remote debugging by going to **Window** → **Preferences** → **z/OS Solutions** → **Troubleshooting Preferences** and check the Enable Troubleshooting Menu box. You can see this in Figure 14-18.
3. Check which port your PC will be listening on by going to **Window** → **Preferences** → **Run/Debug** → **Debug Daemon**. By default this will be set to 8001. You have to know this for your host settings. This is shown in Figure 14-19.
4. Go to the z/OS projects perspective and the **Troubleshooting** menu item should now be in the menu bar.
5. Left-click on the **Troubleshooting** menu item and select **Show client info**. This shows the panel shown in Figure 14-20 on page 332, which gives you the IP address of this machine. This is used by the host to communicate with your workstation, so make a note of the IP address.

Archived

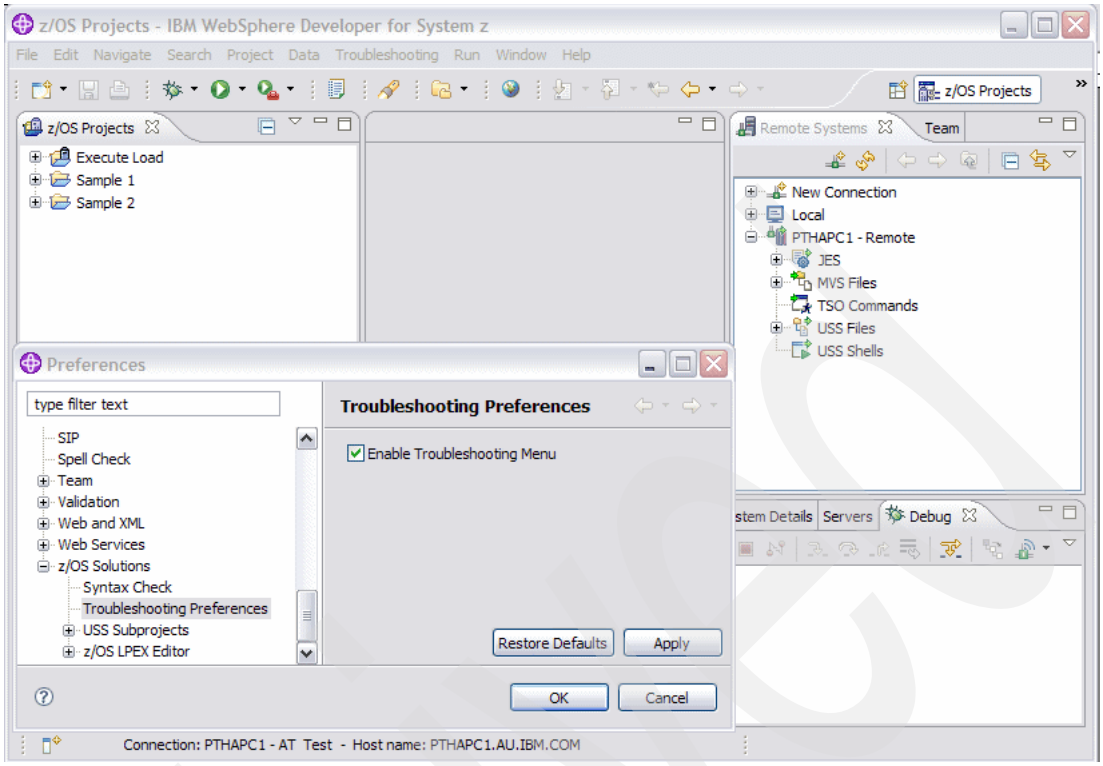


Figure 14-18 WD/z client showing active RSE connection and troubleshooting preferences

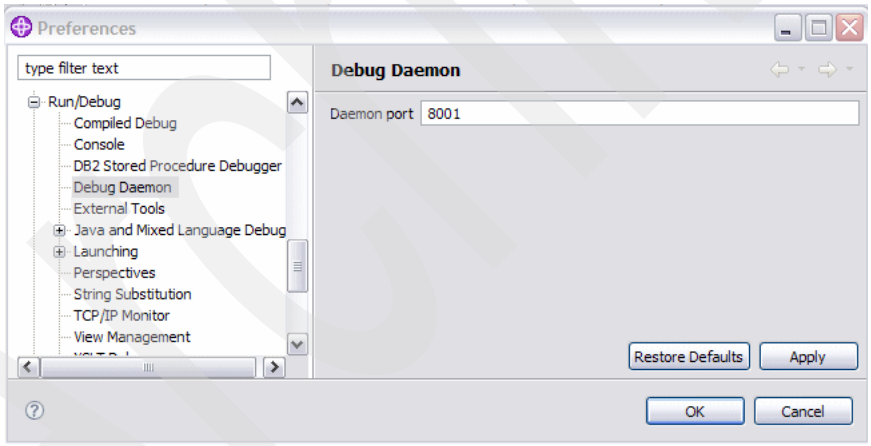


Figure 14-19 Debug Daemon setting

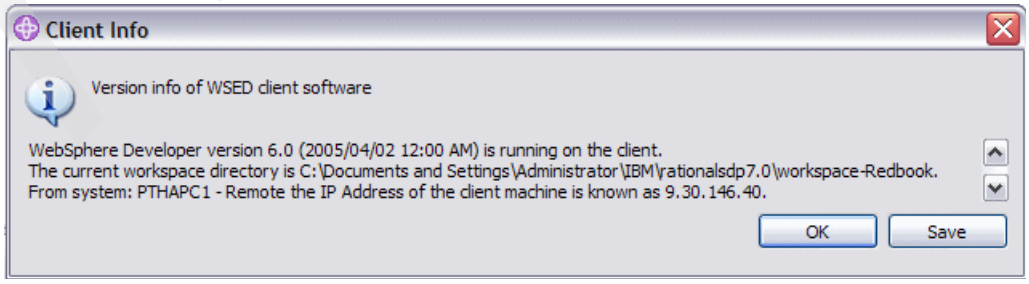


Figure 14-20 Client Info panel showing IP address of workstation

Your client is now ready to accept communication from the z/OS host. So let us now look at what is required on the host to trigger invocation.

Invoking the WebSphere debugger under CICS

If you have a CICS transaction and wish to utilize the WebSphere debugger there are a couple of steps that need to be performed in CICS. Just like invoking Debug Tool solely in CICS you need to set up a debug profile using transaction CADP or DTCN. Figure 14-21 shows the CADP transaction and settings required for remote debugging. We have a profile called RDBKWD4Z that defines the parameters for our remote connection.

```
CADP      -    CICS Application Debugging Profile Manager   -     C64C1IS1

List Debugging Profiles    (A=Activate,I=Inactivate,D=Delete,C=Copy)

  Owner    Profile  S Tran Program  Compile Unit Applid   Userid   Term Type
_ $EXAMPLE COMP1    I T*   P*       *            CICSREG1 PANDREWS TTT1 Comp
_ $EXAMPLE COMP2    I TR   *        SAMPCOMPUN + CICSREG2 DRBEARD* TTT2 Comp
_ $EXAMPLE COMP3    I TRN3 PROG3    *            CICSREG3 *        TTT2 Comp
_ $EXAMPLE CORBA    I T*                         *        IORWERTH      Corb
_ $EXAMPLE EJB      I *                          *        *             EJB
_ $EXAMPLE JAVA     I TR*                        *        PENFOLD*      Java
_ CICSUSER RDBK     I TST1 *        *            C64C1IS1 *        *    Comp
_ CICSUSER RDBKWD4Z A TST1 *        *            C64C1IS1 *        *    Comp









8   profile(s). All profiles shown
Enter=Process PF1=Help 2=Filter 3=Exit 4=View 5=Create Comp 6=Create Java
                 9=Set display device 10=Edit 11=Sort
```

Figure 14-21 CADP transaction showing RDBKWD4Z profile

If we select PF9 - Set display device we can set up the parameters required to enable our CICS transaction to talk to our workstation. In Figure 14-22 we do the following steps:

1. Set the session type to TCP for TCPIP.
2. Set the IP address to the IP address we got from the Show Client Info option in our WD/z client. In the example, this is 9.30.146.40.
3. Set the port number to the port number we got from the Debug Daemon preferences in our WD/z client, by default this is 8001.

Archived

```
CADP      -    CICS Application Debugging Profile Manager   -     C64C1IS1

Set Compiled Debugging Display Device

Debugging Display Device
Session Type             ==>  TCP                          (3270,TCP)
3270 Display Terminal    ==>  1053

TCP/IP Name Or Address
==> 9.30.146.40
==>
==>
==>
Port                     ==>  08001

Type of socket communication ==> Single                    (Single,Multiple)


Display this panel on LE profile activation ==> YES




Enter=Save and return PF1=Help 3=Exit 12=Cancel
```

Figure 14-22 Remote debugging TCPIP settings in transaction CADP

To invoke the debugger to test our transaction, all we need to do is clear the panel and enter our transaction name.

Tip: If the normal 3270 Debug Tool starts rather than the remote debugger, then ensure that the IBM-supplied group DFHSO is installed in the CICS region. You might also have to Inactivate the debug profile and then Activate it again to ensure it is picked up by Debug Tool.

In our example transaction TST1 executes program RDBKC01. So when the remote debugger starts, it starts up program RDBKC01 so that you can step through the code. This is shown in Figure 14-23, where we can see that we have executed the first DB2 call in the program and have received an SQLCODE of -805, so we have a DB2 error.

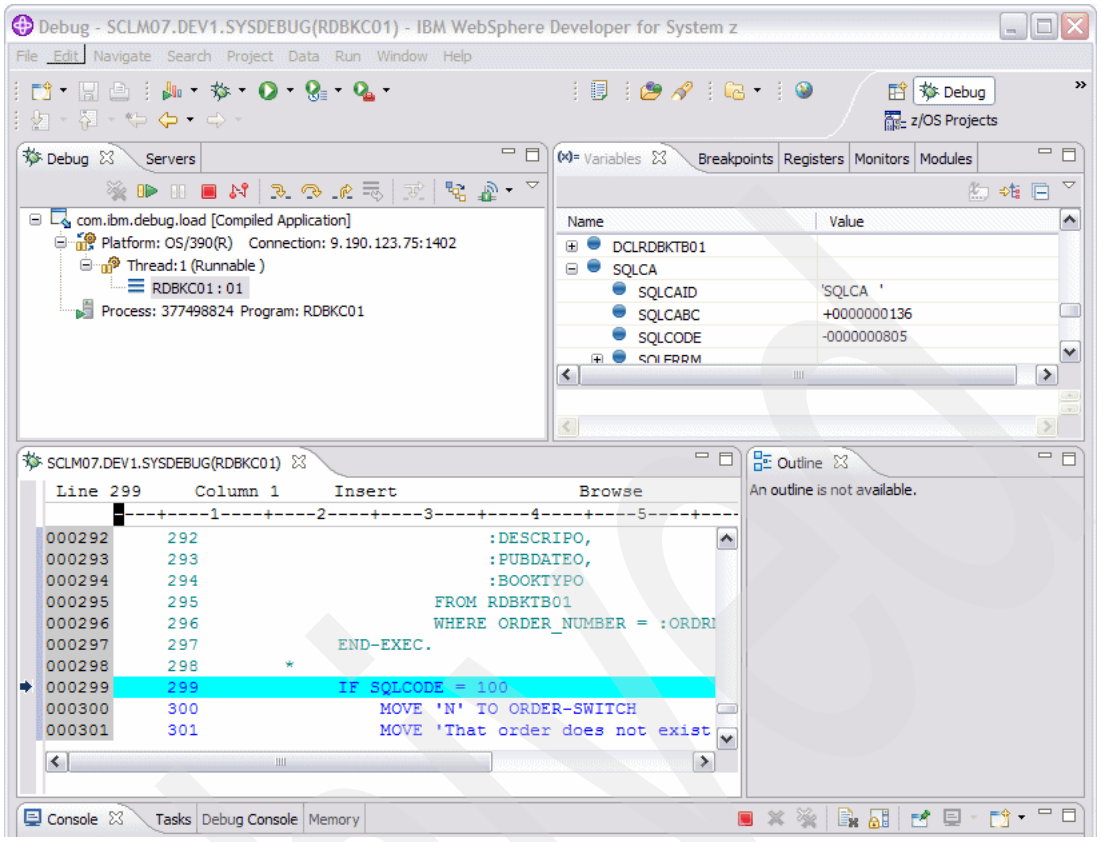


Figure 14-23 Remote debugger in WD/z being invoked from the execution of a CICS transaction

Invoking the WebSphere debugger for batch programs from TSO

Invoking the remote debugger from z/OS is quite straightforward and can be done in a number of ways. In the following examples we look at a PL/I and DB2 program being invoked from a REXX exec plus a COBOL application being invoked from JCL. In both cases the debugger starts and looks the same as the panel shown above.

Invoking a program from REXX to invoke the remote debugger

In Example 14-18 we see the PARM in the DB2 RUN command. Using the TCPIP sub-parameter we pass the IP address of our WD/z client plus the port it is listening on.

Example 14-18 Invoking remote debugger on WD/z from REXX for PL/I and DB2 program

```
/* REXX */

   "ALLOC F(SYSPRINT) DA(*)"
   "ALLOC F(INSPPREF) DA('DOHERTL.INSPPREF')"
   "ALLOC F(EQADEBUG) DA('SCLM07.DEV1.SYSDEBUG',
                         'SCLM07.TEST.SYSDEBUG',
                         'SCLM07.PROD.SYSDEBUG') SHR"

   SUBSYS   = 'DI11'
   DB2_Line = "RUN PROGRAM(RDBKP01) PLAN(RDBKPLN1)" ||,
              " LIB('SCLM07.DEV1.LOAD') " ||,
              " PARM('TEST(,,,TCPIP&9.30.146.40%8001:*)/')"

   queue DB2_Line
```

```
queue "End"
Address TSO "DSN SYSTEM("SUBSYS")"
rcode = RC

"FREE  F(SYSPRINT)"
```

Invoking a program from JCL to invoke the remote debugger

In Example 14-19 we see the program being executed is STARTAPP. The PARM then specifies the TEST option, using the TCPIP sub-parameter to set the IP address of our WD/z client and the Debug Daemon listener port, in this case the default of 8001. The JCL also contains the EQADEBUG DD so that Debug Tool can find the correct side file.

Example 14-19 JCL to test a batch program with the remote debugger

```
//DEBUG     EXEC PGM=STARTAPP,PARM='/TEST(,,,TCPIP&9.30.146.40%8001:*)'
//*
//STEPLIB   DD   DISP=SHR,DSN=SCLM07.DEV1.LOAD
//          DD   DISP=SHR,DSN=SCLM07.TEST.LOAD
//          DD   DISP=SHR,DSN=SCLM07.PROD.LOAD
//*
//EQADEBUG  DD   DISP=SHR,DSN=SCLM07.DEV1.SYSDEBUG
//          DD   DISP=SHR,DSN=SCLM07.TEST.SYSDEBUG
//          DD   DISP=SHR,DSN=SCLM07.PROD.SYSDEBUG
//*
//SYSPRINT  DD   SYSOUT=*
```

This JCL can be submitted from z/OS or through the Submit context menu in the z/IDE in WD/z, as shown in Figure 14-24,

Archived

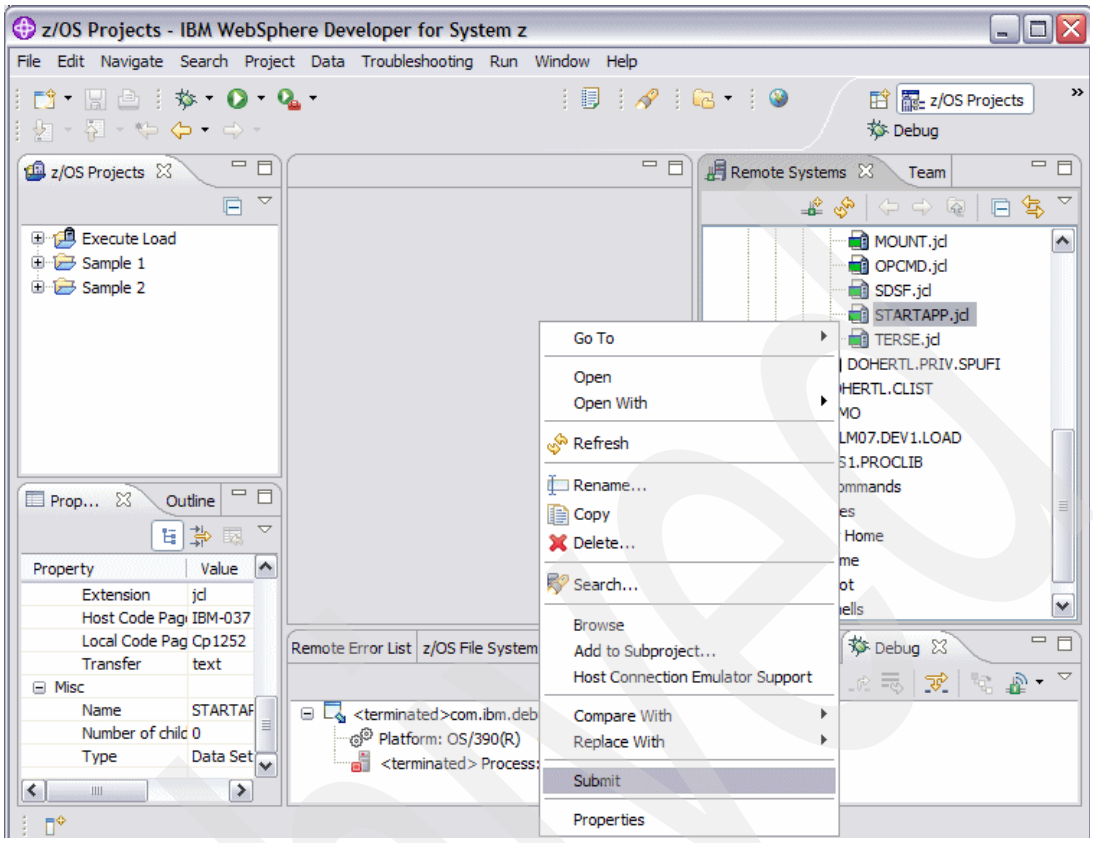


Figure 14-24 Submitting jobs from the z/IDE

The job runs but passes control to the remote debugger. You can then step through the code on your WD/z client. Once finished, or terminated, control is passed back to the JCL and the job finishes.

14.2 Fault Analyzer

Setting up Fault Analyzer within SCLM is in most cases exactly the same as for Debug Tool. The preferred method of providing information to Fault Analyzer is also through side files. The reason is that, if you were going to use listings, then there are certain compiler options that have to be set for Fault Analyzer to be able to analyze the dumps. This information is provided in the side file, which makes things much easier.

The main difference with Fault Analyzer over Debug Tool is during the use of the side file or listing. With Debug Tool, as long as the side file is available, then JCL, CICS Region JCL, or test procedures can be modified to tell Debug Tool the location of the side file. This can happen at any time you need to debug an application. With Fault Analyzer the side file can used at the time of the abend to analyze the abend, or Fault Analyzer can reanalyze the abend after it has occurred. Depending on whether you wish to perform live analysis or not, this will have an effect on how you make the side files available to Fault Analyzer.

14.2.1 Telling Fault Analyzer where to find the side file

Fault Analyzer will perform a series of searches in order to find a suitable side file or listing. These searches are described fully, in sequence, in the section, “Locating compiler listings or side files” in the *Fault Analyzer Users Guide*, SC19-1088-00. The search order that Fault Analyzer uses is as follows:

1. If a COBOL program, then the COBOL IGZIUXB exit
2. The Debug Tool for z/OS EQAUEDAT exit
3. Side files listed in IDISYSDB DD
4. IDILANGX provided side files
5. SYSADATA files
6. Compiler specific listing files (Subsequently converted internally via IDILANGX)

For the last four options in this list, Fault Analyzer uses provided DDs to determine where to locate the side file or listing; the DD can be specified in the JCL if appropriate. If the side files are not specified, the definitions from the PARMLIB configuration member IDICNF00, the IDIOPTS user options file, or an Analysis Control user exit are used to identify these data sets. IDICNF00 holds installation-wide default options and can be created as member IDICNF00 in SYS1.PARMLIB, or any other data set that is part of the logical PARMLIB concatenation.The file allocated to the IDIOPTS DDname holds user options. The data sets specified are as follows:

IDILC PDS(E) data set containing C compiler listings

IDILCOB PDS(E) data set containing COBOL compiler listings (other than OS/VS COBOL)

IDILCOBO PDS(E) data set containing OS/VS COBOL compiler listings

IDISYSDB PDS(E) data set containing COBOL SYSDEBUG side files

IDILPLI PDS(E) data set containing PL/I compiler listings (other than Enterprise PL/I)

IDILPLIE PDS(E) data set containing Enterprise PL/I compiler listings

IDIADATA PDS(E) data set containing SYSADATA from Assembler compilations

IDILANGX PDS(E) data set containing IDILANGX side files for all languages

For a detailed discussion on where Fault Analyzer finds the side files, see the section “Locating compiler listings or side files” in the *Fault Analyzer Users Guide*, SC19-1088-00.

Archived

An example of the PARMLIB configuration member IDICNF00 is shown in Figure 14-25.

```
------------------------------------------------------------------------------
VIEW       SYS1.PARMLIB.APC1.USER(IDICNF00) - 01.04         Columns 00001 00080
Command ===>                                                  Scroll ===> CSR
****** **************************** Top of Data ********************************
000001 INCLUDE
000002 EXCLUDE(TYPE(STC))
000003 EXCLUDE(ABEND(S013,S*37,S213,S806,S913))
000004 RETAINDUMP(ALL)
000005 MMP(512)
000006 DATASETS(
000007          IDIHIST(IDIC1.HIST)
000008          IDIDOC(IDIC1.V7R1M0.SIDIDOC1)
000009          IDIBOOKS(IDIC1.V7R1M0.SIDIBOOK)
000010          IDIMAPS(IDIC1.V7R1M0.SIDIMAPS)
000011          IDILCOB(SCLM07.DEV1.LIST
000012                  SCLM07.TEST.LIST
000013                  SCLM07.PROD.LIST)
000014          IDILPLIE(SCLM07.DEV1.LIST
000015                   SCLM07.TEST.LIST
000016                   SCLM07.PROD.LIST)
000017          IDISYSDB(SCLM07.DEV1.SYSDEBUG
000018                    SCLM07.TEST.SYSDEBUG
000019                    SCLM07.PROD.SYSDEBUG)
000020        )
****** **************************** Bottom of Data ******************************
```

Figure 14-25 IDICNF00 PARMLIB member

Depending on whether you plan to also use Debug Tool may determine what method you chose to use to locate the side file with Fault Analyzer. There is currently only one common point between the two products, which is the EQAUEDAT user exit. This is the debug Tool user exit and Fault Analyzer will run this exit also. If you have a structured approach to your library system, which you should have if you are using SCLM, then using EQAUEDAT may make the most sense.

If a program is being run from SCLM07.PROD.LOAD or possibly an external copy of the member in SCLM07.PROD.LOAD, then it is probably safe to say that the debug side file you should be using to analyze the dump will come from SCLM07.PROD.DEBUG, or whatever you have named your debug file. This would be both true for Fault Analyzer and Debug Tool. So it would be a fairly straightforward process to set up a single EQAUEDAT that satisfies the needs of both products. This would also keep maintenance down to a minimum.

If you are not planning on running Debug Tool, then the simplest option would be to either use the EQAUEDAT user exit or to specify the location of the side file via the IDISYSDB DD in the PARMLIB configuration member IDICNF00, the IDIOPTS user options file, or an Analysis Control user exit.

If you chose not to use the EQAUEDAT user exit, yet are also planning on using Debug Tool, then the location of the side file must be specified in both the IDISYSDB DD for Fault Analyzer and the EQADEBUG DD for Debug Tool (or one of the other Debug Tool methods of specifying the location of the side file).

14.2.2 Using an IDILANGX step in your SCLM translator

As we have said previously, the preferred option for providing Fault Analyzer with the information it needs is via a side file. Fault Analyzer V7 currently supports the use of side files for Enterprise COBOL. For Enterprise PL/I the support of side files in Fault Analyzer is provided via PTF UK23264.

For those users not yet on the levels of compilers that support side files, then listings must be used. It is possible to provide the actual listing location via the specific Fault Analyzer DDs as documented above. But this will be the least efficient method for Fault Analyzer to analyze the abend. Fault Analyzer will internally convert the listing into an IDILANGX file. It is better to create the IDILANGX file, also known as a Fault Analyzer side file, during the compile, then at the time of abend it is already available to Fault Analyzer.

To do this, additional SCLM steps are required in the language translator for the required language. The first thing you must take into consideration if you are using an IDILANGX step is the compiler options. Fault Analyzer requires certain compiler options to be set. If you just use the listing, then Fault Analyzer at analysis time will tell you any options you do not have set. Additionally, when you run the IDILANGX step, it will also tell you if you are missing certain options. A complete list of the compiler options required can be found in the section “Required compiler options for IDILANGX” in the *Fault Analyzer Users Guide*, SC19-1088-00.

Secondly, the IDILANGX step requires the listing being passed in to be allocated to a DD named LISTING. The compiler listing is normally written out to a DD named SYSPRINT. So for an SCLM translator to be able to pass the listing created in DD SYSPRINT to IDILANGX in a DD of LISTING, we need a step to copy the listing from SYSPRINT to LISTING.

Thirdly, we need to add an IDILANGX step to convert the listing. The main parameter is the language, such as COBOL, PL/I or C. Example 14-20 shows an Enterprise PL/I language translator with the correct compiler options, additional copy step, and IDILANGX step.

Example 14-20 Enterprise PL/I language translator with additional steps required for IDILANGX

```
FLMLANGL    LANG=PLEDB2FA,VERSION=1,ALCSYSLB=Y,CHKSYSLB=BUILD,C
               LANGDESC='PLI ENTERPRIZE WITH DB2 AND FA'
*
          FLMINCLS TYPES=(PLI,PLINCL,DCLGEN)
DB2DCLS   FLMINCLS TYPES=(DCLGEN)
*
*********************************************************************
*         --PARSER TRANSLATOR--                                     *
*********************************************************************
*
          FLMTRNSL   CALLNAM='SCLM PL/I PARSE',                      C
               FUNCTN=PARSE,                                         C
               COMPILE=FLMLPGEN,                                     C
               PORDER=1,                                             C
               OPTIONS=(SOURCEDD=SOURCE,                             C
               STATINFO=@@FLMSTP,                                    C
               LISTINFO=@@FLMLIS,                                    C
               LISTSIZE=@@FLMSIZ,                                    C
               LANG=I(I))
*           (* SOURCE      *)
          FLMALLOC  IOTYPE=A,DDNAME=SOURCE
           FLMCPYLB @@FLMDSN(@@FLMMBR)
*
*********************************************************************
```

```
*         --ENTERPRISE PL/I--                                          *
*********************************************************************
*
          FLMTRNSL   CALLNAM='ENTERPRISE PL/I COMPILER',               C
               FUNCTN=BUILD,                                           C
               COMPILE=IBMZPLI,                                        C
               TASKLIB=TASKLIB,                                        C
               VERSION=4.0,                                            C
               GOODRC=4,                                               C
               PORDER=1,                                               C
               OPTIONS=(MACRO,OBJECT,SOURCE,XREF,PP(SQL),MAP,LIST,NEST,C
               AGGREGATE,ATTRIBUTES,NOBLKOFF,OFFSET,OPTIONS)
*
*********************************************************************
*         --DDNAME ALLOCATION--                                        *
*********************************************************************
*
           FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,                C
               RECNUM=5000,DFLTTYP=OBJ
*
           FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,RECNUM=5000
*
           FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000
*
           FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
            FLMCPYLB NULLFILE
*
           FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
            FLMCPYLB NULLFILE
*
           FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,KEYREF=LIST,             C
               DFLTTYP=LIST,PRINT=Y,RECNUM=5000
*
           FLMALLOC  IOTYPE=P,DDNAME=DBRMLIB,MEMBER=@@FLMONM,          C
               DFLTTYP=DBRM,KEYREF=OUT1,                               C
               RECFM=FB,LRECL=80,RECNUM=5000,DIRBLKS=1
*
           FLMALLOC  IOTYPE=A,DDNAME=TASKLIB
            FLMCPYLB  EPLI.V350.SIBMZCMP
            FLMCPYLB  DB2.V810.SDSNEXIT
            FLMCPYLB  DB2.V810.SDSNLOAD
*
           FLMALLOC  IOTYPE=I,DDNAME=DB2DCLS,KEYREF=SINC,INCLS=DB2DCLS
*
*********************************************************************
*         --COPY FILES FROM SYSPRINT TO LISTING                        *
*********************************************************************
*
         FLMTRNSL   CALLNAM='COPY FILES      ',                       C
               FUNCTN=BUILD,                                           C
               COMPILE=COPYFILE,                                       C
               DSNAME=SCLM.PROJDEFS.REXX,                              C
               CALLMETH=TSOLNK,                                        C
```

```
              VERSION=1.0,                                              C
              PORDER=1,                                                 C
              OPTIONS=(SYSPRINT,LISTING),                               C
              GOODRC=0

           FLMALLOC  IOTYPE=W,RECFM=VBA,LRECL=133,                      C
              RECNUM=90000,DDNAME=LISTING
*
*********************************************************************
*         --CREATE IDILANGX FILE                                    *
*********************************************************************
*
          FLMTRNSL CALLNAM='IDILANGX',                                  C
               FUNCTN=BUILD,                                            C
               COMPILE=IDILANGX,                                        C
               DSNAME=IDI.V610.SIDIMOD1,                                C
               VERSION=3.5.2,                                           C
               GOODRC=0,                                                C
               PORDER=1,                                                C
               OPTIONS='@@FLMMBR(PLI ERROR OFT IDILANGX FAULT'
*
*********************************************************************
*         --DDNAME ALLOCATION--                                     *
*********************************************************************
*
           FLMALLOC IOTYPE=U,DDNAME=LISTING
*
           FLMALLOC IOTYPE=P,DDNAME=IDILANGX,DFLTTYP=IDILANGX,          C
              KEYREF=OUT2,BLKSIZE=27998,LRECL=1562,RECFM=VB,            C
              RECNUM=10000,DIRBLKS=50,DFLTMEM=*
*
           FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
```

The REXX to perform the copy that is picked up from SCLM.PROJDEFS.REXX is shown in Example 14-21.

Example 14-21 REXX to copy listing file

```
/* REXX */
/*********************************************************************/
/* Copy file I to file O.  Both are assumed to be pre-allocated.     */
/*********************************************************************/
PARSE UPPER ARG I","O .

"EXECIO * DISKR "I" (STEM R. FINIS "
"EXECIO * DISKW "O" (STEM R. FINIS "

RETURN
```

Archived

With the extra steps, the build messages will be as shown in Figure 14-26.

```
FLM49000 - INVOKING BUILD PROCESSOR
FLM09002 - THE BUILD REPORT WILL APPEAR IN DOHERTL.BUILD.REPORT49
FLM09006 - THE BUILD LISTING WILL APPEAR IN DOHERTL.BUILD.LIST49
FLM42000 - BUILD PROCESSOR INITIATED - 02:13:55 ON 2007/02/08
FLM44500 - >> INVOKING BUILD TRANSLATOR(S) FOR TYPE: PLI      MEMBER: RDBKP01
FLM06501 - TRANSLATOR RETURN CODE FROM ===> ENTERPRISE PL/I      ===> 4
FLM06501 - TRANSLATOR RETURN CODE FROM ===> COPY FILES           ===> 0
FLM06501 - TRANSLATOR RETURN CODE FROM ===> IDILANGX             ===> 0
FLM46000 - BUILD PROCESSOR COMPLETED - 02:14:00 ON 2007/02/08
FLM09008 - RETURN CODE = 0
```

Figure 14-26 Build messages after a build that also creates IDILANGX file

The SCLM process for Assembler is slightly simpler, as the IDILANGX expects the input to be in DD SYSADATA, and this is also the output DD from the assemble. So the extra copy step is not required. An example of an assembler language translator is shown in Example 14-22.

Example 14-22 Assembler language translator with additional IDILANGX step

```
HLAF     FLMSYSLB SYS1.MACLIB
*
         FLMLANGL    LANG=HLAF,VERSION=HLAF1.0,ALCSYSLB=Y,           C
               LANGDESC='HIGH LEVEL ASSEMBLER WITH FAULT ANALYZER'
*
*********************************************************************
*         --PARSER TRANSLATOR--                                     *
*********************************************************************
          FLMTRNSL   CALLNAM='SCLM HLAS PARSE',                      C
               FUNCTN=PARSE,                                         C
               COMPILE=FLMLPGEN,                                     C
               PORDER=1,                                             C
               OPTIONS=(SOURCEDD=SOURCE,                             C
               STATINFO=@@FLMSTP,                                    C
               LISTINFO=@@FLMLIS,                                    C
               LISTSIZE=@@FLMSIZ,                                    C
               LANG=A)            *** THIS IS ASSEMBLER ONLY ***
*          (* SOURCE     *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
*********************************************************************
*         --BUILD TRANSLATOR(S)-- --HIGH LEVEL ASSEMBLER INTERFACE--  *
*********************************************************************
*
         FLMTRNSL   CALLNAM='HL ASM',                               C
               FUNCTN=BUILD,                                        C
               COMPILE=ASMA90,                                      C
               VERSION=1.5,                                         C
               GOODRC=0,                                            C
               PORDER=1,                                            C
               OPTIONS=(XREF(SHORT),LINECOUNT(75),OBJECT,RENT,ADATA),  C
               PARMKWD=PARM1
*
```

Archived

```
*********************************************************************
*         --DDNAME ALLOCATION--                                     *
*********************************************************************
*
          FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=9000
*
          FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,                C
              RECNUM=9000,DFLTTYP=OBJ
*
          FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
           FLMCPYLB NULLFILE
*
          FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
           FLMCPYLB NULLFILE
*
          FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
          FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,KEYREF=LIST,PRINT=Y,     C
              RECFM=FBA,LRECL=121,                                    C
              DFLTTYP=LIST,RECNUM=20000
          FLMALLOC  IOTYPE=W,DDNAME=SYSADATA,RECFM=VB,RECNUM=9000,    C
              LRECL=8188,BLKSIZE=8192
*
*********************************************************************
* IDILANGX BUILD TRANSLATOR                                         *
*********************************************************************
*
          FLMTRNSL  CALLNAM='IDILANGX',                               C
              FUNCTN=BUILD,                                           C
              COMPILE=IDILANGX,                                       C
              DSNAME=IDI.V610.SIDIMOD1,                               C
              VERSION=6.1,                                            C
              GOODRC=0,                                               C
              PORDER=1,                                               C
              OPTIONS='@@FLMMBR(ASM ERROR OFT IDILANGX FAULT'
*
*********************************************************************
*         --DDNAME ALLOCATION--                                     *
*********************************************************************
*          (* IDILANGX *)
          FLMALLOC  IOTYPE=P,DDNAME=IDILANGX,DFLTTYP=IDILANGX,        C
              KEYREF=OUT2,BLKSIZE=27998,LRECL=1562,RECFM=VB,          C
              RECNUM=10000,DIRBLKS=50,DFLTMEM=*
*                                                                   *
```

14.3 Using debug translators

There are a number of ways to pick up debug tool options. In most cases, as Fault Analyzer and Debug Tool are going to want to pick up the side files from the PROD level at the time of an abend, then debugging will be turned on in all language translators.

Archived

Usually, when you specify TEST in combination with any other sub-options (except NONE), the compiler options NOOPTIMIZE and OBJECT automatically go into effect, preventing you from debugging optimized programs. However, if you specify TEST(NONE,SYM) and compile with one of the following compilers, you can specify OPT, allowing you to debug optimized programs:

- Enterprise COBOL for z/OS and OS/390, Version 3 Release 2
- Enterprise COBOL for z/OS and OS/390, Version 3 Release 1 with APAR PQ63235
- COBOL for OS/390 & VM, Version 2 with APAR PQ63234

Most of the examples previously show setting the compiler options to TEST(ALL,SYM,SEPERATE). These examples show how the compiler options are set to produce side files. For more information on the actual compiler options and what they do, you should refer to the *Debug Tool Users Guide*, SC19-1071-00, or to the Redbooks publication, *IBM Application Development and Problem Determination Tools V7 for System z: Application Performance Analyzer, Debug Tool Utilities and Advanced Functions, Fault Analyzer, File Export, File Manager, and Workload Simulator,* SG24-7372.

Your site may not use Fault Analyzer, so may only want to run debug at the development group level during code development. If this is the case you may want to recompile at promotion to the TEST group to turn the debugging off. You can do this by using the FLMLRBLD macro in conjunction with the FLMTOPTS to set different compiler options at different levels.

Finally, you may want to only use debug translators at the development group when the need arises. To do this, you can use an SCLM Alternate project to define the same languages with debug options turned on. Then just build the required module by using the alternate project definition. Then before promoting, build the member again with the primary project definition so that debug options are turned off.

14.4 Conclusions

There are a number of different ways to set up SCLM to provide the required files for debugging and fault analysis. Depending on how you plan to use Debug Tool and/or Fault Analyzer, this will determine the best way for you to set up the translators.

In all cases it is better to use the SYSDEBUG side file if at all possible, as this will make things a lot simpler, especially when Fault Analyzer is used, as the vast array of compiler options will not be required. If Fault Analyzer is used, then the preferred approach would be to use the option to put a real data set in the load module as described in 14.1.3, “Option 2: Debug side file under SCLM - correct side file name at compile time” on page 311 and then resolve this to a correct side file by using the EQAUEDAT user exit. As EQAUEDAT is the common denominator between Debug Tool and Fault Analyzer, and both products are being used, then this option will be preferred in order to minimize customizations in both products to find side files.

If only Debug Tool is used, then any of the methods used to determine the location of the side file will work fine. Using the EQAUEDAT method will again mean that your administrator controls where the side file is found, thus minimizing the amount of customization in CICS region JCL and other procedures. However, this method will involve adding an extra member create and copy step to the SCLM language translators as documented above. If you are happy to code a EQADEBUG DD whenever you need to debug code, then it will be sufficient to use the default SCLM behavior as documented in 14.1.2, “Option 1: Debug side file under SCLM - default temporary name”.

If your site rebuilds code at every level, then using the method described in14.1.4, “Option 3: Debug side file under SCLM - correct side file name at each group” will always put the correct location of the side file into the load module, so you should not need to use EQAUEDAT, EQADEBUG, or any of the other methods to find the side file, as Debug Tool and Fault Analyzer will always find the correct one. However, this is not standard processing in SCLM, as building at the development group and then promoting through the hierarchy is the normal practice.

Archived

Part 4

SCLM: Advanced topics

In this part of the book, we discuss Breeze for SCLM, enhanced access control, and Merge Tool.

Archived

© Copyright IBM Corp. 2007. All rights reserved.

Archived

15

Breeze for SCLM - installation, setup, and use

In this chapter we discuss the installation, setup, and use of the Breeze for SCLM product.

Archived

© Copyright IBM Corp. 2007. All rights reserved.

15.1 SCLM Advanced Edition overview

SCLM Advanced Edition for z/OS is a set of add-on products for IBM SCLM for z/OS. When used with SCLM, it is a fully functional and centralized z/OS software configuration management solution. From the same Eclipse IDE interface, developers and administrators can use a central repository to store, manage, build, and deploy Java and J2EE applications as well as traditional applications such as COBOL and PL/I. It allows the users to invoke all key SCLM functions using a simplified workstation-based graphical interface in addition to the traditional ISPF-based interface. The SCLM Advanced Edition components are shown in Figure 15-1.

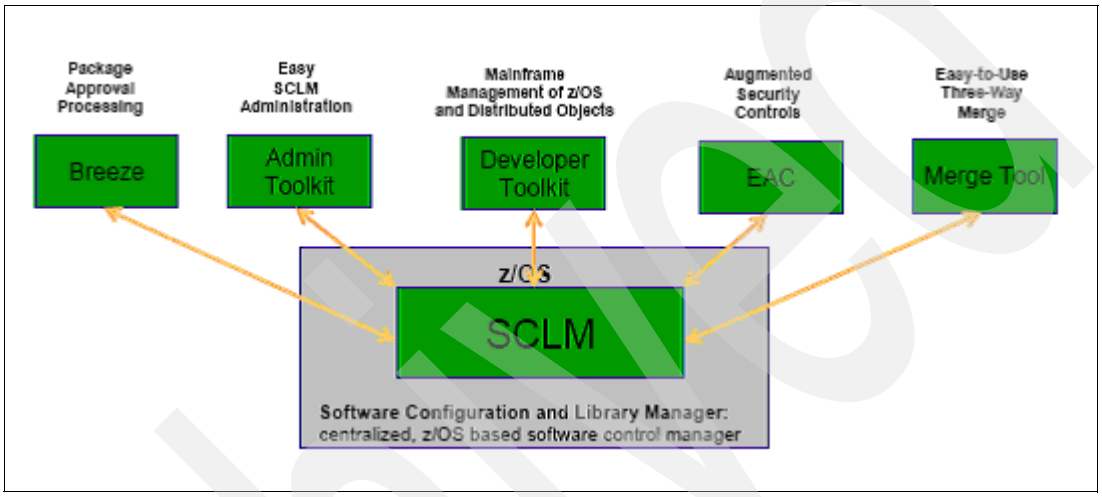


Figure 15-1 SCLM Advanced Edition components

SCLM Advanced Edition for z/OS is a suite of the following add-on products to SCLM:

- SCLM Administrator Toolkit
- Enhanced Access Control for SCLM for z/OS
- SCLM Developer Toolkit
- Breeze for SCLM for z/OS
- Merge Tool for z/OS

IBM SCLM Administrator Toolkit simplifies the administration of SCLM managed projects by means of a workstation-based graphical interface or an ISPF based interface.

Enhanced Access Control provides additional security to protect your SCLM objects from change outside of SCLM.

SCLM Developer Toolkit is an Eclipse-based tool that extends SCLM Services to the workstation. It facilitates collaboration between z/OS and non z/OS-based developers and provides a transparent IDE-based interface to SCLM, long file name support, and a remote portal via SCLM Explorer/Developer View that allows access to other SCLM-based products.

The Breeze product provides for package notification, review, and approval processing throughout the life cycle.

IBM Merge Tool for z/OS and OS/390 provides a variety of tools to perform merge tasks.

15.2 Breeze introduction

The Breeze component of SCLM Advanced Edition adds package approval processing to the SCLM promote process as follows:

- When you use SCLM to promote a package of changed source members, Breeze collects and stores information about the package. (In Breeze, the term *package* refers to an SCLM high-level architecture member.)
- Breeze logs the progress of packages as they are promoted up through the hierarchy of an SCLM project, from development to production.
- Breeze allows you to define approvers who can vote on (approve or veto) the promotion of changes in a particular inventory location (SCLM project, group, type, language, and member name). When an SCLM user attempts to promote a package, Breeze sends a message via e-mail or TSO SEND, or both, to the approvers for the affected inventory locations (and any other users that you nominate). This message includes a link to the Breeze Web interface.
- The Breeze Web interface (a Java applet that runs in your Web browser) enables you to view package information (including the accumulated log of previous activity) and, if you are an approver, to vote on whether a package should be promoted to the next higher group in the SCLM project.

Archived

15.3 Breeze components

The Breeze for SCLM components are shown in Figure 15-2.

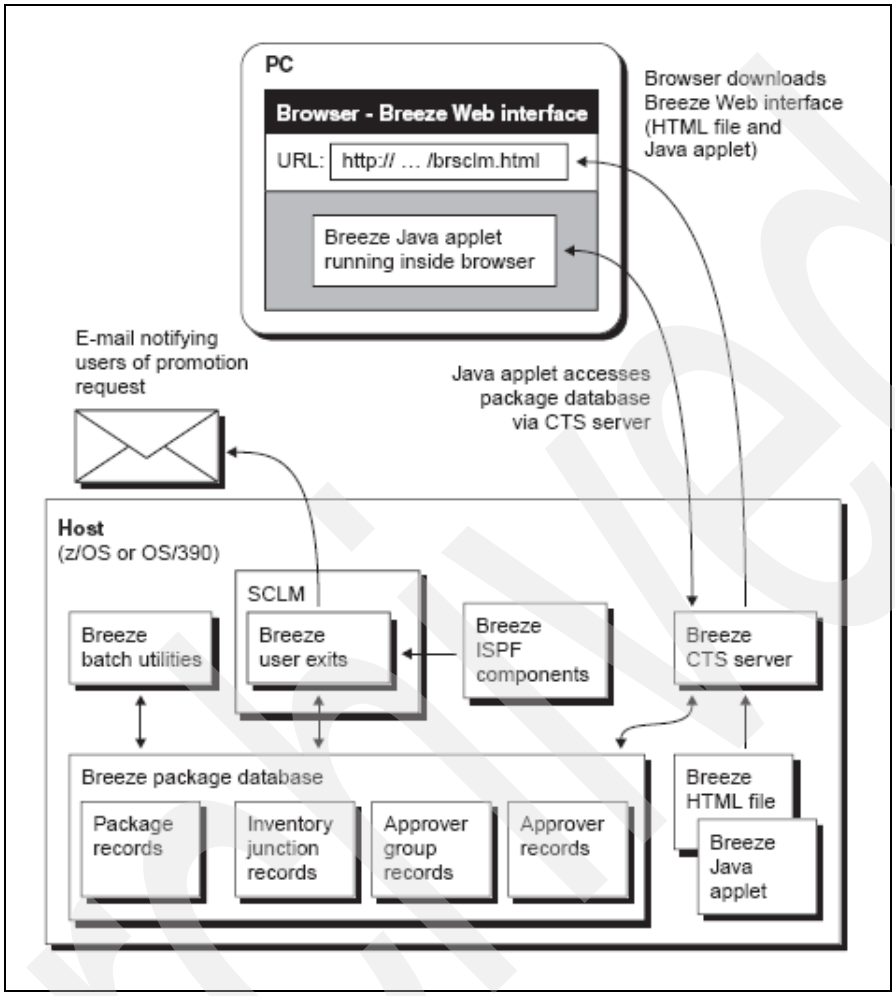


Figure 15-2 Breeze components

Breeze consists of these components:

Package database VSAM data set containing Breeze package records.

User exits Programs that customize the SCLM build and promote functions for Breeze functions (for example, to create package records in the Breeze package database, and send e-mails notifying users of promotion requests).

ISPF components Various ISPF pop-up windows that the Breeze user exits display during SCLM foreground processing.

CTS server An HTTP server that hosts the Breeze Web interface with software that allows the Web interface to access the package database

Web interface A Web page, hosted by the CTS server, that allows users to view and vote on packages.

Batch utilities Various utilities for maintaining or reporting on the Breeze package database (for example, to define approvers, delete old package records, or report on packages).

15.4 Breeze installation

The complete installation tasks for Breeze can be found in the *Breeze Installation Guide*, SC31-8819-08. In this chapter we summarize the installation tasks and present various hints to help you install Breeze in a timely manner.

15.4.1 Breeze installation tasks

1. **System requirements:** Review the system requirement for Breeze, including reserving a dedicated TCP/IP port for the Breeze server and gathering all information about the MVS system. This is covered in steps 1 and 2 of Chapter 3 of the *Breeze Installation Guide*.
2. **Breeze PTFs:** All Breeze PTFs must be installed prior to using it. The current list includes UW86723, UW88287, UW92089, UW92342, UW93213, UA01834, UA02586, UA05538, UA05687, UA07596, UA10080, UA15482, UA15864, UA16617, UA17694, UA18438, UA19824, UA24695, UA29279.
3. **Breeze database:** This is a VSAM file that contains all data relating to Breeze packages. Create it using the supplied job and load with demo data. The demo data will be used during the installation verification test (IVT) and can be deleted later using the Breeze utility jobs.

   Note 1: All SCLM users will require update access to the Breeze database.

   Note 2: Over time, the Breeze database will grow. It can be archived to reduce the size of it when it gets too big.

4. **CIGINI:** This load module contains various product parameters. It contains Product password, Breeze database name, Load library name, and Java control dataset. This load module must be available to all other breeze load modules.
   - Most defaults can be taken in the JCL that creates CIGINI.
   - Most users will only need to add a job card and change "`tdisk`" to your work unit for temp disk datasets.
   - You will have to change the breeze load lib and breeze database name if you use other than the defaults.
   - The Breeze password must be kept as "PASSWORD".

   Creating this load module is covered in step 5 of Chapter 4 of the *Breeze Installation Guide*.
5. **Breeze Server:** This server job combines an HTTP server with software that allows the Web interface to access the package database. This server job can be run initially under a individual's user ID for testing, but should be set up as a started task when Breeze is ready for use. Setting up the Breeze server job is covered in steps 7 and 8 of Chapter 5 of the *Breeze Installation Guide.*

   Two other jobs, BZZSMPKG and BZZSMPRT, are submitted by the server; one to submit the package approval (BZZSMPKG) and the other (BZZSMPRT) to retrieve data from the host to display in the user's Web interface. If you use EAC to protect your SCLM datasets, then you will have to make a modification to BZZSMPKG to allow it to work. This is discussed in 15.9.2, "EAC and Breeze" on page 387 later in this chapter.
   - These two jobs are submitted by the server on behalf of the Breeze user, therefore you must give the user ID that is running the Breeze server the authority to submit jobs by

creating a SURROGAT for each user that will use the Breeze server to vote or browse members.

– As new users join or leave the team, these SURROGAT records must be maintained.

BZZSMPRT is used when the user of the Breeze Web application needs to browse SCLM information about package members. This job writes out the information to temp files that the Breeze server job reads and passes on to the user.

BZZSMPRT can be customized to create the temp dataset name to begin with the user's user ID along with the date and time or by preceding the user ID with a fixed high level qualifier such as SCLMTEMP or SCLMBREZ.

– In the former case, then the user ID of the Breeze Server ID must be given alter access to all Breeze user's user IDs.

– In the latter case, then the Breeze server's user ID will need alter access to the specified fixed high level and all breeze user's will need read access to it.

If you use a fixed high level qualifier for the temp dataset name, you may want to create a SMS rule to delete the files created under this high level qualifier on a daily basis.

6. **Java Control:** These are a number of members that control the way that Breeze works. The customized versions are stored in Breeze's SBZZJAVA dataset. They consist of the following members:

   – **BZZ$CNTL:** This stores the TCPIP port number used by the Breeze server job. This member is set up in the previous step.

   – **BRHTML:** This is the HTML that contains the Breeze Web interface. It must be configured with the IP address of the system where Breeze is installed and the TCPIP port number used by the Breeze server job. Setting up this member, testing it, and the Breeze Web interface is covered in step 9 of Chapter 6 of the *Breeze Installation Guide.*

   – **$$$$SMTP:** SMTP parameters. This member must be set up even if you do not set up Breeze to send out e-mail because it is used also for TSO SEND notification. Setting up this member and testing it is covered in step 13b of Chapter 9 of the *Breeze Installation Guide.*

   – **$$HTML:** This contains message text, including Web link, to be used for notification messages. This member must be set up even if you do not set up Breeze to send out e-mail, since it is used also for TSO SEND notification. Setting up this member and testing it is covered in step 13c of Chapter 9 of the *Breeze Installation Guide.* This step also documents a number of ways to customize the e-mail that is sent out.

   – **$$COLL:** This says whether you want collisions reported or not.

   – **$$EMER:** This is a list of user IDs allowed to do emergency promotes

   – **$$EXCL:** This is a list of SCLM types you want Breeze to ignore in its inventory collection

7. **SCLM Integration:** There are several steps that must be done to integrate Breeze with SCLM. This step is covered in step 11 and 12 of chapter 8 of the *Breeze Installation Guide.*

   a. The Breeze fields must be added to the SCLM promote panel, FLMP#P. The fields to be merged in are in a set of panels in the Breeze SBZZPENU dataset. You should merge in the fields based upon the member for the level of your operating system. Here is the full list:

      - BZZP#P28 OS/390 2.8 version of FLMP#P
      - BZZP#P29 OS/390 2.9 version of FLMP#P
      - BZZP#P2A OS/390 2.10 and z/OS 1.1 version of FLMP#P

- BZZP#P2C z/OS 1.2 version of FLMP#P
- BZZP#P2D z/OS 1.5 version of FLMP#P
- BZZP#P2E z/OS 1.6 version and above of FLMP#P

The base FLMP#P panel can be modified or better yet, placed in a dataset under your PROJDEFS dataset, such as SCLM.PROJDEFS.PANELS, and have the SCLM user's logon proc changed to include this dataset ahead of the base ISPF dataset for DDNAME ISPPLIB.

b. The Breeze variables must be added to the SCLM promote skeleton, FLMP$. The variables to be merged in are in a set of skeletons in the Breeze SBZZSENU dataset. You should merge in the fields based upon the member for the level of your operating system. Here is the full list:

- BZZP$28 OS/390 2.8 version of FLMP$
- BZZP$29 OS/390 2.9 version of FLMP$
- BZZP$2A OS/390 2.10 and z/OS 1.1 version of FLMP$
- BZZP$2C z/OS 1.2 version of FLMP$
- BZZP$2D z/OS 1.5 version of FLMP$
- BZZP$2E z/OS 1.6 version and above of FLMP$

c. The Breeze datasets must be added to the SCLM batch skeleton, FLMLIBS. The datasets to merge in are found in the member BZZFLMSK in SBZZSENU.

The base FLMP$ and FLMLIBS skeletons can be modified or better yet, placed in a dataset under your PROJDEFS dataset, such as SCLM.PROJDEFS.SKELS, and have the SCLM user's logon proc changed to include this dataset ahead of the base ISPF dataset for DDNAME ISPSLIB.

d. The four Breeze user exits must be called from the SCLM Build notify and three promote user exit points:

- A call to BZZSME01 in SBZZCLIB must be added to the Build notify exit point.

  Here is an example of how our common Build exit from Chapter 12, "SCLM user exit processing" on page 241 looks after it was modified to add this call as shown in Example 15-1.

Example 15-1 Common Build Exit

```
/* REXX */
ARG parm                  /* Parse arguments into variable parm */
bldntfrc = 0
ADDRESS ISPEXEC
"SELECT CMD(EX 'SCLM07.PROJDEFS.REXX(COPYEXTR)' '"parm"')"
bldntfrc = MAX(bldntfrc,rc)

"SELECT CMD(EX 'SCLM07.PROJDEFS.REXX(BINDEXTR)' '"parm"')"
bldntfrc = MAX(bldntfrc,rc)

"SELECT CMD(EX 'SCLM07.PROJDEFS.REXX(DEPLOY)' '"parm"')"
bldntfrc = MAX(bldntfrc,rc)

"SELECT CMD(EX 'BZZ.SBZZCLIB(BZZSME01)' '"parm"')"
bldntfrc = MAX(bldntfrc,rc)
exit bldntfrc
```

- A call to BZZSME02 in SBZZCLIB must be added to the Promote Verify exit point.

  We also added a check to this exit point to ensure that all promotes are done through a high level ARCHDEF, *which is a Breeze requirement.* Even with Breeze installed, individual members could be promoted without Breeze verification if this check is not added. This code assumes that all high level ARCHDEFs are stored in the PACKAGE type, which is our convention for the example projects in this book. You will have to adjust for your projects accordingly.

  Here is an example of how our common promote verify exit from Chapter 12, “SCLM user exit processing” on page 241 looks after it was modified to add this code and call. Notice that the parameters must be parsed to determine the type as shown in Example 15-2.

Example 15-2 Common Promote Verify Exit

```
/* REXX */
ARG parm                  /* Parse arguments into variable parm */
CALL Init
prmvfyrc = 0

if type <> 'PACKAGE' then do
   say 'All promotes must use the PACKAGE type'
   exit 20
   end

ADDRESS ISPEXEC
"SELECT CMD(EX 'BZZ.SBZZCLIB(BZZSME02)' '"parm"')"
prmvfyrc = MAX(prmvfyrc,rc)

if prmvfyrc = 0 then do
   "SELECT CMD(EX 'SCLM07.PROJDEFS.REXX(BKUPPROD)' '"parm"')"
   prmvfyrc = MAX(prmvfyrc,rc)
   end

exit prmvfyrc

Init:
   /* Break out the parms passed from the SCLM user exit */
   PARSE UPPER VAR parm extyp ',' proj ',' prjdf ',' tsouid ',',
         fromgrp ',' type ',' member ',' scope ',' mode ',' togrp

   extyp    = STRIP(extyp,'T')    /* Exit Type - where in SCLM?     */
   proj     = STRIP(proj,'T')     /* Project - high level qualifier */
   prjdf    = STRIP(prjdf,'T')    /* Alternate Project Definition   */
   tsouid   = STRIP(tsouid,'T')   /* User executing SCLM function   */
   fromgrp  = STRIP(fromgrp,'T')  /* From grp(Promote) or build grp */
   type     = STRIP(type,'T')     /* Type from the SCLM panel       */
   member   = STRIP(member,'T')   /* Member from the SCLM panel     */
   scope    = STRIP(scope,'T')    /* Scope entered on SCLM panel    */
   mode     = STRIP(mode,'T')     /* Mode entered on SCLM panel     */
   togrp    = STRIP(togrp,'T')    /* To grp (Promote), N/A (Build)  */
return
```

Archived

- An alternate way to protect against non-high level ARCHDEF promotes is to edit BZZSMEP1 in your SBZZCLIB dataset, look for the NOTHL routine as shown below and modify the ret_code = 0 line to read ret_code = 8.

```
NOTHL:

/* ---------------------- */
/* set pkg flag to no     */
/* ---------------------- */

    VPKGYN = 'N'
    ADDRESS  ISPEXEC  'VPUT (VPKGYN)'  PROFILE
    say member' is not a HL ARCHDEF - BREEZE processing bypassed'
    ret_code = 0         <- Modify this line to ret_code = 8
    CALL EXIT
```

- A call to BZZSME03 in SBZZCLIB must be added to the Promote Copy exit point.

  Here is an example of how our common promote copy exit from Chapter 12, “SCLM user exit processing” on page 241 looks after it was modified to add this call as shown in Example 15-3.

Example 15-3 Common Promote Copy Exit

```
/* REXX */
ARG parm                  /* Parse arguments into variable parm */
prmcpyrc = 0

ADDRESS ISPEXEC
"SELECT CMD(EX 'BZZ.SBZZCLIB(BZZSME03)' '"parm"')"
prmcpyrc = MAX(prmcpyrc,rc)
exit prmcpyrc
```

- A call to BZZSME04 in SBZZCLIB was required in earlier releases of Breeze, but it is no longer necessary.

8. **SMTP server:** If you want Breeze to send notification via e-mail to approvers, then you have to configure Breeze to use the SMTP server to handle the e-mail. TSO Send can be used if E-mail notification is not required. This configuration and testing can be found in chapter 9 of the *Breeze Installation Guide.* If you have to use secure SMTP, then you must follow the instructions in this chapter to set up Breeze to use it.

Archived

15.5 Defining approver records

After you have installed Breeze, you will have to set up the records in the Breeze database that define your approval processing. You only need to define these records if your site wants to use Breeze to vote on package promotions, or notify users of package promotions. If your site wants to use Breeze only to view and record packages as they are promoted through the project hierarchy, then you do not need to modify Breeze after installation.

Figure 15-3 shows how Breeze monitors and controls promotions.

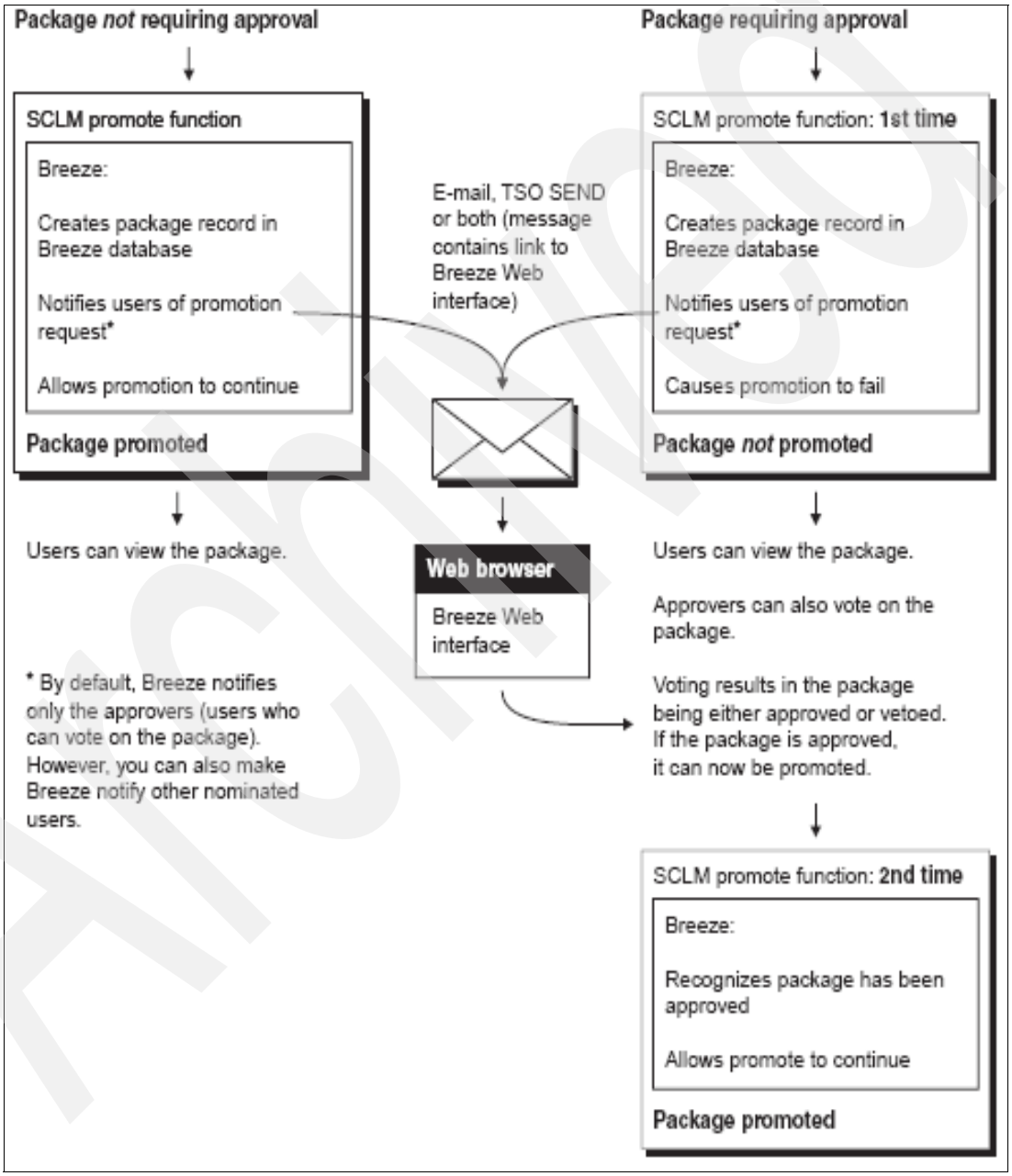


Figure 15-3 How Breeze identifies approvers

Archive

There are three record types that are required to be setup to use Breeze to vote on package promotions:

- **Inventory junction records**: Link inventory locations (identified by SCLM project, group, type and language) with approver groups.
- **Approver group records**: Define approver group names and quorums, which approvers they contain, and various attributes for each approvers role in the group (for example, whether an approver's vote is required when deciding the final result of the group vote).
- **Individual approver user records**: Define approvers, including their e-mail addresses, phone numbers and real names.

15.5.1 Defining inventory locations - concepts

The concept of approver records in Breeze is based on two assumptions:

- Different users are responsible for different members in an SCLM project, depending on where the member is in the project hierarchy, and what kind of data the member contains. The users are defined to Breeze in the approver and approver group records.
- You can identify who is responsible for a member by its "inventory location" — that is, its SCLM project, group, type, language and member name. Inventory locations are tied to approver group records with a junction record.

Before you can define approver junction records, you have to study the architecture of the SCLM projects at your site, and identify who is responsible for the members in each part of the hierarchy for a particular project and how they can be best identified to tie them to approver groups.

1. First you have to take a look at your groups and determine who is responsible for approving members that are to be promoted out of them, for example, from group DEV1 to TEST or from UAT to PROD.
2. Next, if a group is not specific enough to define your approver processing, determine if a specific type within a group is enough to properly define your approval processing. For example, a DBA may need to identify all DB2 programs or DCLGENs being promoted and they can be identified by a member of type DBRMLIB or DCLGEN being promoted.
3. If a group and / or a type is not specific enough to define your approver processing you can look to see if a particular source language type can further identify members to be approved. For example, batch programs and CICS programs may have to be approved by two sets of approvers and then the language of those programs can be used to identify batch versus CICS programs.
4. Finally if group, type and language is not specific enough to define your approver processing, individual members can be identified as needing to be approved by specific approver groups. This last method is not recommended, as it requires a junction record for each member, which can be very time consuming to set up and maintain.

15.5.2 Defining inventory junction records - JCL

To define inventory junction records, you use Breeze batch utility commands. Each inventory junction record requires an inventory location and an approver group name it is to be tied to. Wildcards may be used for any of the inventory locations (project, group, type, language or member). If a package contains any members from an inventory location specified in an inventory junction record, then the approver group (and all associated approvers) are assigned to approve or veto the package.

Use Utility Job BZZSMJIJ, as shown in Example 15-4, in your SBZZJCL dataset to define your inventory locations. Specify the Approver group to the right of the = sign. Include the other values as needed or enter * to match any member, project, group, type or language.

Example 15-4 Job BZZSMJIJ - Defining inventory locations

```
//....your job card...
//*
//STEP1   EXEC PGM=BZZSAPIJ
//STEPLIB  DD  DSN=BZZ.SBZZLOAD,DISP=SHR
//CIGLOG   DD  SYSOUT=*
//CIGRPT   DD  SYSOUT=*
//CIGOUT   DD  SYSOUT=*
//CIGIN    DD  *
ASSIGN TO APPROVER GROUP = ................
           MEMBER            = .........
       OF  PROJECT           = .........
           ALTERNATE PROJECT = .........
           GROUP             = .........
           TYPE              = .........
           LANGUAGE          = .......
    .
/*
```

The parameters are:

- APPROVER GROUP: 1–16 character approver group name
- MEMBER: 1–8 character SCLM member name
- OF PROJECT: 1–8 character SCLM project name
- ALTERNATE PROJECT: 1–8 character SCLM project name
- GROUP: 1–8 character SCLM group name
- TYPE: 1–8 character SCLM type member name
- LANGUAGE: 1–8 character

Notes:

1. Breeze interprets the inventory location that you specify in inventory junction records as the source location of members (before promotion). During approver group collection processing, Breeze looks at the source location of each valid member and then finds the associated inventory/approver group record.
2. You can specify wild cards (*) for project, group, type, language and member names: you can specify "*" on its own to match any name, or you can specify a partial name followed by "*" (for example, "DEV*" matches all names beginning with "DEV": DEV1, DEV2 etc.). However, we recommend that you specify at least the complete project name, and as many other parts of the inventory location as possible, so that the approver group is responsible only for appropriate inventory locations.

15.5.3 Defining approvers and approver groups - concepts

An approver is a user assigned to monitor source changes within a certain inventory location. An approver group is a group of 1–n approvers (users) with e-mail and/or TSO SEND access that has been assigned to monitor a certain inventory location. Typically, the approver groups contain peer or management members of an application team. Then, as the package is promoted up the project hierarchy, the approver groups tend to be product control personnel and security.

Archived

How Breeze identifies the approvers for a package

During the SCLM promote function, Breeze determines the inventory location of each source member in the package. If any of these inventory locations match Breeze inventory junction records, then Breeze uses the approver groups named in the inventory junction records to collect a list of approvers for the package. This is known as “content and approver collection”.

When you assign an approver to an approver group, you can nominate the approver as “required”. Within an approver group, every required approver must vote before the group vote can be decided as “approved” or “vetoed”.

For each approver group, you specify a “quorum”, the number of votes required before the group vote can be decided. The quorum must be equal to or greater than the number of required approvers in the group. (An approver group with required approvers cannot have a quorum of zero.)

Approve only once

An enhancement to typical approval processing is that Breeze has an “approve only once” option. This means that if the user has previously voted on the package when it was promoted from a lower group in the project hierarchy, then their vote will be reapplied to later requests for promotion. This allows the user to vote only once, and not be required for further intervention as the package is promoted up the project hierarchy.

Notifying approvers

When package promotion is requested, Breeze can notify users in two ways: e-mail and TSO SEND. Most organizations typically use both because they complement each other. TSO SEND is immediate so the approvers can be notified quickly. However, e-mail is trackable while TSO SEND is not and it can be sent to all users while TSO SEND can not because not all approvers have TSO access. Another advantage to e-mail notification is that the message contains a link to the Breeze Web interface. Users can then just click on the link and go directly to the Web interface.

However, not all approvers want to be e-mailed about pending approvals. They may prefer to check periodically from their desktop.

Whether or not users are notified is controlled in two ways. First, to shut off all notifications of approvers, do not provide a $$HTML member in SBZZJAVA, the trigger that e-mail is implemented for the product. (See the *Breeze Installation Guide* for more information on e-mail.) To shut off e-mails for select users or groups, use the individual user ID attribute definitions to limit to TSO only.

15.5.4 Defining approver groups - JCL

Use Utility Job BZZSMJG1 in SBZZJCL to define approver groups as shown in Example 15-5.

Example 15-5 Job BZZSAPG1 - Defining Approver Groups

```
//...your job card
//STEP1   EXEC PGM=BZZSAPG1
//STEPLIB  DD  DSN=BZZ.SBZZLOAD,DISP=SHR
//CIGLOG   DD  SYSOUT=*
//CIGRPT   DD  SYSOUT=*
//CIGOUT   DD  SYSOUT=*
//CIGIN    DD  *
       ADD     APPROVER GROUP = ...............
```

```
               QUORUM         = ....
       .
       ADD     APPROVER GROUP    = ................
       INCLUDE USERID            = .......
               REQUIRED          = .
               APPROVE ONLY ONCE = .
               NOTIFY ONLY       = .
       .
/*
```

The first two lines of input must be defined once. This sets the QUORUM. The next set of lines must be repeated for each user in the approver group.

QUORUM: The minimum number of votes required in the approver group before the group vote can be decided as “approve” or “veto”. An approver group with required approvers cannot have a quorum of zero. The quorum must be equal to or greater than the number of required approvers in the group. Even if there is a quorum of votes, the group vote cannot be decided until all required approvers have voted.

USERID: 1–8 character user ID of the approver to be added to the group, deleted from the group, or whose approver group details you want to update or list.

REQUIRED: Required approver. Must vote before the package status can be determined.

APPROVE ONLY ONCE: If the user has previously voted on the package when it was promoted from a lower group in the project hierarchy, then their vote will be reapplied to later requests for promotion. This allows the user to vote only once, and not be required for further intervention as the package is promoted up the project hierarchy.

NOTIFY ONLY: The user is not allowed to vote but is notified when a package requests approval.

Uses of the NOTIFY ONLY flag

This flag can be very useful if you have employees in a group who must be notified of a change but they do not have the authority to approve the change. For example, all members of the operations group need to know about upcoming changes to jobs because they need to schedule and execute them, but only the manager of that group can approve the job changes. You can then create a operations approver group where the manager of the group and their backup can vote, but department members are listed in the group as notify only. You might also want to create a “NOTIFY ONLY” approver group. In this case every approver is “NOTIFY ONLY”. When the first promote executes the approver group automatically is set to APPROVED.

15.5.5 Defining approvers - JCL

Use Utility Job BZZSMJU1 in SBZZJCL to define approvers as shown in Example 15-6.

Example 15-6 Job BZZSAPU1 - Defining approvers

```
//STEP1   EXEC PGM=BZZSAPU1
//STEPLIB DD  DSN=BZZ.SBZZLOAD,DISP=SHR
//CIGLOG   DD  SYSOUT=*
//CIGRPT   DD  SYSOUT=*
//CIGOUT   DD  SYSOUT=*
//CIGIN    DD  *
```

```
        ADD    APPROVER USER
               ID            = ........
               NAME          = '........................'
               PHONE         = ..............
               EMAIL1        = '........................'
               EMAIL2        = '........................'
               EMAIL3        = '........................'
               TSOSEND       = .
               REQUIRED      = .
               APPROVE ONLY ONCE = .
             .
/*
```

The parameters are as follows:

ID 1–8 character approver user ID.

NAME 1–30 character user name. If the name contains spaces, then it must be enclosed in quotes.

PHONE Up to a 10 digit telephone number.

EMAIL1–3 1–50 character e-mail address. You may enter up to three e-mail addresses for each specified user ID. If the address contains spaces, then it must be enclosed in quotes.

TSO SEND **Y** - Notify via TSO SEND and e-mail.

N - Notify via e-mail only.

Blank - Defaults to Y.

REQUIRED No longer used.

APPROVE ONLY ONCE No longer used.

If there is no Approver User record, then the notification message is sent as a TSO SEND message.

15.5.6 Example of JCL to define all three types

Example 15-7, Example 15-8, and Example 15-9 are three example jobs that show the creation of approver user, approver group and approval junction records.

This first job creates three approver user records and then lists the approvers in the database.

Example 15-7 BZZSMJU1 - Definer approver user records

```
//BZZSMJU1 JOB (#ACCT),'BREEZE JOB',CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//* -------------------------------------------------------------------*
//*  NAME:    BZZSMJU1                                                 *
//*  PURPOSE: MAINTAIN USER/APPROVER ATTRIBUTE RECORDS.                *
//* -------------------------------------------------------------------*
//*
//STEP1   EXEC PGM=BZZSAPU1
//STEPLIB  DD  DSN=BZZ.SBZZLOAD,
//             DISP=SHR
//CIGLOG   DD  SYSOUT=*
//CIGRPT   DD  SYSOUT=*
//CIGOUT   DD  SYSOUT=*
```

```
//CIGIN    DD  *
        ADD    APPROVER USER
               ID            = DOHERTL
               NAME          = 'LIAM DOHERTY'
               PHONE         = 123456789
               EMAIL1        = 'DOHERTL@AU1.IBM.COM'
               TSOSEND       = Y
           .
        ADD    APPROVER USER
               ID            = PCECK
               NAME          = 'PETER ECK'
               PHONE         = 123456789
               EMAIL1        = 'PCECK@US.IBM.COM'
               TSOSEND       = Y
           .
        ADD    APPROVER USER
               ID            = JMEHTA
               NAME          = 'JYOTINDRA MEHTA'
               PHONE         = 123456789
               EMAIL1        = 'JMEHTA@IN.IBM.COM'
               TSOSEND       = Y
           .
        LIST   APPROVER USER
               ID            = *
               BUILD REPORT
           .
/*
```

Example 15-8 creates two approver groups, one approver group for development and one approver group for TEST. It adds the three users we defined as approvers to each approver group. Finally, it LISTs all approver groups in the database.

Example 15-8 BZZSMJG1 - Defining approver group records

```
//BZZSMJG1 JOB (#ACCT),'BREEZE JOB',CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//* -------------------------------------------------------------------*
//*  NAME:    BZZSMJG1                                                 *
//*  PURPOSE: MAINTAIN APPROVER GROUP RECORD DEFINITIONS.              *
//* -------------------------------------------------------------------*
//STEP1   EXEC PGM=BZZSAPG1
//STEPLIB  DD   DSN=BZZ.SBZZLOAD,
//              DISP=SHR
//CIGLOG   DD  SYSOUT=*
//CIGRPT   DD  SYSOUT=*
//CIGOUT   DD  SYSOUT=*
//CIGIN    DD  *
      ADD     APPROVER GROUP = DEVAPPR
              QUORUM         = 1
      .
      ADD     APPROVER GROUP    = DEVAPPR
      INCLUDE USERID            = PCECK
              REQUIRED          = N
              APPROVE ONLY ONCE = N
              NOTIFY ONLY       = N
      .
      ADD     APPROVER GROUP    = DEVAPPR
```

```
        INCLUDE USERID            = DOHERTL
                REQUIRED          = N
                APPROVE ONLY ONCE = N
                NOTIFY ONLY       = N
        .
        ADD     APPROVER GROUP    = DEVAPPR
        INCLUDE USERID            = JMEHTA
                REQUIRED          = N
                APPROVE ONLY ONCE = N
                NOTIFY ONLY       = N
        .
        ADD     APPROVER GROUP = TESTAPPR
                QUORUM         = 1
        .
        ADD     APPROVER GROUP    = TESTAPPR
        INCLUDE USERID            = PCECK
                REQUIRED          = N
                APPROVE ONLY ONCE = N
                NOTIFY ONLY       = N
        .
        ADD     APPROVER GROUP    = TESTAPPR
        INCLUDE USERID            = DOHERTL
                REQUIRED          = N
                APPROVE ONLY ONCE = N
                NOTIFY ONLY       = N
        .
        ADD     APPROVER GROUP    = TESTAPPR
        INCLUDE USERID            = JMEHTA
                REQUIRED          = N
                APPROVE ONLY ONCE = N
                NOTIFY ONLY       = N
        .
        LIST    APPROVER GROUP    = *
                BUILD REPORT
        .
/*
```

Example 15-9 creates junctions records for each approver group that ties the members in the development groups (using DEV*) to the development approver group and the TEST group members to the TEST approver group. It then reports on the junction records.

Example 15-9 BZZSMJIJ - Defining junctions records

```
//BZZSMJIJ JOB (#ACCT),'BREEZE JOB',CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//* ------------------------------------------------------------------*
//*  NAME:    BZZSMJIJ                                                 *
//*  PURPOSE: MAINTAIN INVENTORY JUNCTION RECORDS                      *
//* ------------------------------------------------------------------*
//STEP1   EXEC PGM=BZZSAPIJ
//STEPLIB  DD  DSN=BZZ.SBZZLOAD,
//             DISP=SHR
//CIGLOG   DD  SYSOUT=*
//CIGRPT   DD  SYSOUT=*
//CIGOUT   DD  SYSOUT=*
//CIGIN    DD  *
    ASSIGN TO APPROVER GROUP = DEVAPPR
```

Archived

```
            MEMBER            = *
         OF PROJECT           = SCLM07
            ALTERNATE PROJECT = *
            GROUP             = DEV*
            TYPE              = *
            LANGUAGE          = *
      .
      ASSIGN TO APPROVER GROUP = TESTAPPR
            MEMBER            = *
         OF PROJECT           = SCLM07
            ALTERNATE PROJECT = *
            GROUP             = TEST
            TYPE              = *
            LANGUAGE          = *
      .
      LIST APPROVER GROUP = *
           BUILD REPORT
      .
/*
```

15.5.7 Defining watch records

You can optionally define records that will notify approver groups if there is a promotion activity. These users are not able to vote. Basically it works the same as an inventory junction record, except that on promotion, the user is not asked to vote on the package, but is only informed of the promotion activity after the promotion occurs.

```
Use Utility Job BZZSMJP1
ADD PROGRAM WATCH FOR
MEMBER              = PAYPROG1
OF PROJECT = SCLMTEST
ALTERNATE PROJECT   = SCLMTEST
GROUP               = TEST
TYPE                = SOURCE
LANGUAGE            = PLIO
NOTIFY APPROVER GROUP = AUDITGRP
```

Note: Watch occurs during the real promote, not first promote, so these records cannot be used to notify users ahead of time that the members will be promoted.

Archived

15.6 Breeze and the SCLM promote process

Next we discuss Breeze and the SCLM promote process.

15.6.1 Promotions without Breeze

Developers use the SCLM promote function to promote packages. The SCLM promote function has three phases:

1. Verify that the package has been built using the latest source member changes.
2. Copy the source members and any build outputs to the next higher group in the project hierarchy.
3. Purge the source members from the current group.

Without Breeze, and if there are no problems, then SCLM performs each phase immediately, one after the other. However, if a phase returns a non-zero return code (indicating that a problem occurred), then SCLM does not proceed to the next phase, and the promotion fails.

15.6.2 Promotions with Breeze

After Breeze is installed, developers still use the SCLM promote function to promote packages. However, if Breeze determines that a package requires approval, then the developer must use the SCLM promote function *twice*.

1. The first time, Breeze collects the approver information and then notifies the approvers for the package, and causes SCLM to fail the promotion.
2. The second time, if the approvers have voted to approve the package, then Breeze allows the promotion to proceed.

Breeze promote user exits

Breeze introduces two user exits that customize the SCLM promote function:

1. Promote Verify: SCLM invokes Breeze “promote verify” user exit here.
2. Promote Copy: SCLM invokes Breeze “promote copy” user exit here.

Here is a summary of what these Breeze exits do:

- **Verify exit during first promote:** If the normal SCLM verify phase succeeds, then the Breeze promote verify user exit determines if the promote input member is a valid package (a high-level architecture member). If it is not a valid package, then the Breeze exit returns control to SCLM with a return code of zero, and the promotion proceeds with no further involvement from Breeze. *We recommend, though, that you modify the breeze exit to return an 8 if the member is not a package to prevent members from being promoted outside of Breeze control.* We discussed this situation in 15.4.1, “Breeze installation tasks” on page 353 above.

  If it is a valid package, then the Breeze exit records the package information in the Breeze package database, and continues processing.

  – If the Breeze exit determines that the package does not require approval, then it sets the package status to APPROVED, and returns control to SCLM with a return code of zero, allowing the promotion to proceed.

  – If the package does require approval, then the Breeze exit sets the package status to PENDING, notifies the approvers, and returns a non-zero return code to SCLM. This causes the promotion to “fail”. The approvers can now view and vote on the package.

- **Verify exit during second promote:** If the voting results in a package status of APPROVED (instead of VETOED), then the developer can now promote the package by retrying the promote function. This time, the Breeze promote verify user exit recognizes that the package has been approved, and returns control to SCLM with a return code of zero, allowing the promotion to proceed.

Instead of manually using the promote function a second time, you can schedule the Breeze promotion sweep batch utility, BZZSMJS1, to automatically promote packages that have been approved.

- **Copy exit during second promote:** If the normal SCLM copy phase succeeds, then the Breeze promote copy user exit updates the package status to PROMOTED. Otherwise, the Breeze exit either updates the package status to INCOMPLETE or leaves the package in an APPROVED status, depending upon installation settings. The parameter “RESET NOT REQUIRED ON INCOMPLETE” as specified in the CIGINI file controls this behavior.
- **Purge exit during second promote:** Breeze does not affect the normal SCLM Purge phase.

15.6.3 Promotion with Breeze: Step-by-step procedure

Next we cover promotion with Breeze in a step-by-step manner.

Creating and building the package

The first step of promoting with Breeze is to create a package containing the items you want to promote. All programs should be included into the package using an INCL of the program’s ARCHDEF. Member types that are not built, such as JCL, should be included with a PROM statement. This is illustrated in the following example that was first presented in the ARCHDEF chapter. In Example 15-10, four programs are being included with an INCL statement, and a JCL and DOC member are being included with a PROM statement. After you create the package, you must build it using the SCLM build function, which is described in Chapter 9, “SCLM utilities” on page 185.

Example 15-10 Example package

```
* NAME: P2007601
* DESC: PACKAGE FOR 2007601
* DEFECT: 2007601
*
* ADD SOME NEW FUNCTIONALITY AND FIXING A DEFECT
*
*    ARCHDEF MEMBERS
*
INCL  SCLMTST1   ARCHDEF      * Batch Program
INCL  SCLMTST1   LECDEF       * Batch DB2 Program
INCL  SCLMTST2   LECDEF       * Online DB2 Program
INCL  SCLMTST3   ARCHDEF      * Online Program
*
*    JCL MEMBERS
*
PROM  PAY63702   JCLLIB       * Job Control Language
*
*    DOC MEMBERS
*
PROM  P2007601   DOC          * Documentation
```

15.6.4 Requesting approval to promote the package

After creating and building the package, use the SCLM promote function to request approval to promote the package. This is the “first promote” discussed previously. When you select the promote option from SCLM option 5 or SCLM option 3.1, you are presented with the SCLM promote panel, as shown in Figure 15-4, that incorporates the Breeze attribute fields.

```
                        SCLM Promote - Entry Panel
Command ===>

Promote input:
  Project . . . : SCLMTEST
  From group  . . DEV1
  Type  . . . . . PACKAGE              Enter "/" to select option
  Member  . . . . P2007601             Workstation Promote

  Mode  . . 1  1. Conditional          Scope . . . 1  1. Normal
               2. Unconditional                       2. Subunit
               3. Report                              3. Extended
Output control:
               Ex Sub                  Process . . 2  1. Execute
  Messages . . 1  2    1. Terminal                    2. Submit
  Report . . . 3  2    2. Printer
                       3. Data set     Printer . . *
                       4. None         Volume  . .
Breeze control: Clear Package: N (Y/N) Override Dates: N (Y/N)
  Window (yy/mm/dd - hh:mm) 07 / 09 / 01 00 : 00 Thru 07 / 09 / 30 00 : 00
  Description: FIX DEFECT 2007601                        Type: ST (ST/EM)
```

Figure 15-4 SCLM Promote - Entry Panel

The package Type (standard or emergency) does not affect approval, and is for documentation purposes only, except for one key difference: if your administrator has created a $$EMER member in the SBZZJAVA data set, then only the user IDs listed in that member can build or promote emergency packages. If there is no $$EMER member in the SBZZJAVA data set, then there is no difference in behavior between standard and emergency packages.

The Window attributes consist of a start and end date between which the package can be promoted. If the dates are left blank then the Breeze default dates are used.

The Description attribute allows the developer to enter a description for the package. If this is left blank then the package will be given a blank description.

The Clear Package attribute, if set to “Y” will allow the developer to automatically clear the package if it exists at the next level of the hierarchy. This enables packages to be reused without having to manually clear same named packages using the BZZSMJD1 job. If the developer does not clear the package, either manually with the BZZSMJD1 job or by using the Clear Package attribute, they will get a message from Breeze stopping the promote as Breeze will not allow the contents of a package to be overwritten unless the status of that package is set to blank.

Archive

The Override attribute, if set to “Y” will allow the developer to override these parameters on the second promote if they were entered incorrectly on the first promote. If after approval the second promote has been invoked but the promotion date window has not been reached, due to being entered incorrectly on the first promote, by using the override attribute the developer can set the promotion window to it’s correct value. If the developer invokes the SCLM promote service as opposed to using the foreground SCLM promote, then Breeze will assign default attributes to the package.

When the SCLM promote function starts, it produces messages similar to these:

FLM59001 - INVOKING PROMOTE PROCESSOR
FLM09002 - THE PROMOTE REPORT WILL APPEAR IN BABEL.PROMOTE.REPORT02
FLM09004 - THE PROMOTE MESSAGES WILL APPEAR IN BABEL.PROMOTE.MSGS02

If the normal promote verify phase succeeds, then SCLM invokes the Breeze promote verify user exit. The Breeze exit calls the RPTARCH SCLM service to generate an architecture report, producing a message similar to this:

FLM87107 - RPTARCH SUCCEEDED FOR MEMBER PKG1 AT 16:33:04, CODE: 0

Breeze uses this report to determine if the promote input member is a valid package (that is, a high-level architecture member). If the member is not a package, then Breeze performs no further processing; the exit returns control to SCLM, and SCLM continues with a normal promote.

Content and approver collection Breeze uses the architecture report it generated earlier to determine the inventory location of each source member in the package. If any of these inventory locations match Breeze inventory junction records, then Breeze uses the approver groups named in the inventory junction records to collect a list of approvers for the package. This is known as “content and approver collection”. At this point, Breeze displays the message shown in Figure 15-5.

```
*******************************************************
* Content and approver collection in progress for     *
* Package P2007601                                    *
* From Group = DEV1                                   *
* To Group = TEST                                     *
*******************************************************
```

Figure 15-5 Breeze - Approver collection in progress

If Breeze determines that the package has no approvers, or all its approvers belong to groups with a quorum of zero, then the package needs no approval before promotion. Breeze sets the package status to APPROVED.

If the package has at least one approver in a group whose quorum is greater than zero, then the package requires approval before promotion. Breeze sets the package status to PENDING, and displays the following pop-up window, as shown in Figure 15-6.

```
*
           Package Approver Reminder Prompt Option

==> The package P2007601  has 0001 approver groups
assigned, containing 0003 individual approvers
of which 0000 are required.

The package is not eligible for promotion until the
package is approved. All users have been notified that
their vote is required.

To approve or veto the package promotion, please use
the Breeze for SCLM Web Interface.
          End = Return Help = PF1
```

Figure 15-6 Breeze package appprover reminder pop-up panel

If you execute the promote in a submitted job, then similar information will appear in SYSTSPRT.

Breeze notifies all approvers for the package via e-mail, TSO SEND or both, depending on the approver records. For more information about approver records.

Returning control to SCLM

After notifying any approvers and updating the Breeze database, the Breeze exit returns control to SCLM. If the package status is PENDING (requires approval), then the Breeze exit returns control to SCLM with a return code of 4, causing the promotion to fail. If the package status is APPROVED, then the Breeze exit returns control to SCLM with a return code of zero, allowing the promotion to continue. The package can now be viewed using the Breeze Web interface.

15.6.5 Voting on the package

If the package status is PENDING, approvers can vote on the package. For more information see 15.7, “Viewing and voting on packages using the Breeze Web interface”.

What if the package is vetoed?

If voting results in a package being vetoed, it generally means that you must make further changes to the source members in the package (to satisfy the approvers’ reasons for vetoing the package). Then you have to rebuild the package, and request approval to promote the rebuilt package. Before rebuilding the package, run the Breeze clear batch utility, BZZSMJD1. This sets the package status to blank, and deletes the votes and previously collected content and approver data. When you use the SCLM promote function to request approval for the rebuilt package, the Breeze promote verify user exit will perform content and approver collection for the package, and change its status from blank to PENDING (keeping the package activity log, with its record of the previous veto). Approvers can now view and vote on the rebuilt package.

Rebuilding a package during voting or after approval

If you rebuild an existing package when it is in PENDING or APPROVED status, Breeze changes the package status to blank. When the rebuilt package is promoted, Breeze will also reissue the e-mail and TSO notifications to all users who have approved the package, notifying each user that the package must be re-approved.

Archived

Also, users defined to watchlists will receive Breeze notification if a package clear occurs as the result of an SCLM build being performed. If the package has been promoted to a higher level in the hierarchy and then one of the modules is changed and the package rebuilt, Breeze will display panel BZZSM014 which states that the package exists at the higher level already. What must be done in this case is to use the CLEAR utility or promote with the CLEAR option to clear all reference of the voting on the package, so that it can be reused. Alternatively, a different package ID could be used. It should also be noted that vetoed packages will be cleared if an SCLM build is performed on the package.

15.6.6 Promoting the approved package

If the package required approval, and voting has resulted in the package being approved, then the package can now be promoted. To promote the package, retry the SCLM promote function. This time, the Breeze promote verify user exit recognizes that the package has been approved, and allows the promotion to proceed. The Breeze promote verify user exit allows the promotion to proceed only if all of the following conditions are true:

- The package is in APPROVED status.
- The current date is inside the package execution window.
- The SCLM package build time stamp has not changed since content and approver collection was performed.
- If the package type is emergency and your administrator has created a $$EMER member, then your user ID must be listed in the member. (If the $$EMER member exists, then only the user IDs listed in this member can build or promote emergency packages.

If the promotion proceeds, and the subsequent copy and purge phases also succeed, then the Breeze promote exit updates the package status to PROMOTED.

Automatically promoting approved packages

Instead of manually retrying the SCLM promote function, you can schedule the Breeze promotion sweep batch utility, BZZSMJS1, to automatically promote packages that have been approved and are within their execution window. For more information, see 15.9.1, “Breeze sweep job” on page 381.

Problems during promotion of the approved package

If, during the second promote, there is a problem with the promote (such as a space problem in the target data set), the promote fails. In this instance, Breeze does one of two things:

- If “RESET NOT REQUIRED ON INCOMPLETE” is specified in the CIGINI file, Breeze leaves the package in APPROVED status. This means that, once the problem is rectified, the promotion is retried without having to RESET the package. The disadvantage is that the sweep job still picks up the problem package until the problem is rectified.
- If “RESET NOT REQUIRED ON INCOMPLETE” is not specified in the CIGINI file, Breeze changes the status of the package being promoted to INCOMPLETE. In this case, the sweep job (BZZSMJS1) does not pick up the problem package. Also, it allows you to list the problem packages with the Web browser by selecting “Promotion Failed” under “Packages by Status”. Once you have rectified the issues causing the promote problem, you can run the RESET utility with job BZZSMJD1. This resets the status of the package back to its previous status of APPROVED. The promote can then be invoked again. If the promote is successful, the status of the package is changed to PROMOTED.

15.7 Viewing and voting on packages using the Breeze Web interface

An e-mail message is sent to all potential approvers of a package. The e-mail message contains a link to the Web page that displays the Breeze interface. To start using the Breeze Web interface, you can either click on the link in the e-mail message, or ask your Breeze administrator for the Web address (URL) of the interface, and enter the address in your Web browser. The Web address of the Breeze Web interface has the following format where ipAddress is the TCPIP address of your host computer and port is the TCPIP port number that the Breeze server job is using to communicate with the Breeze Web applications:

```
http://ipAddress:port/brsclm.html
```

This Web page downloads a Java applet that displays in your Web browser window. When the Breeze Web interface loads, it displays a login prompt as shown in Figure 15-7.

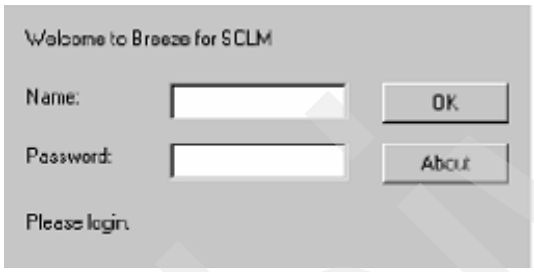


Figure 15-7 Breeze logon panel

After you logon with your mainframe ID and password, the Breeze Web interface will load as shown in Figure 15-8.

Archived

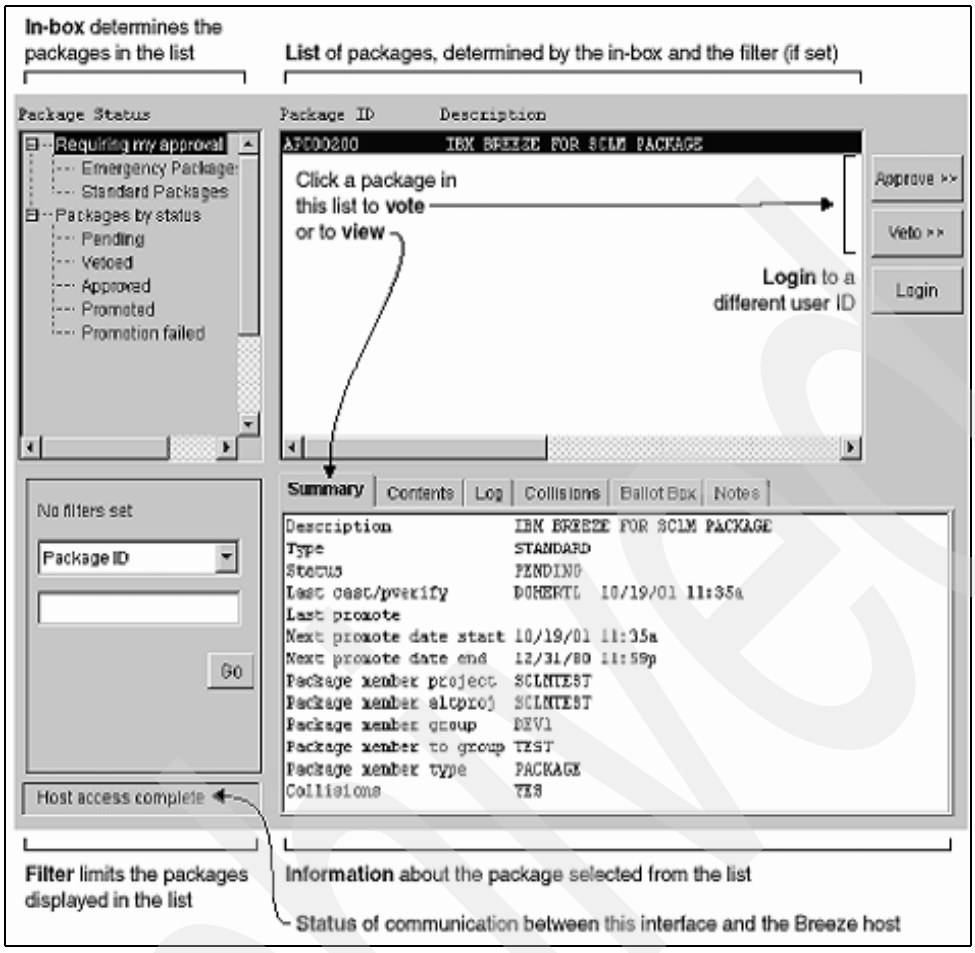


Figure 15-8 Breeze main panel

The main panel is divided into several areas as shown in Table 15-1.

Table 15-1 Main panel areas

In-box	Lets you determine which packages are displayed in the list according to package status and whether you are an approver for the package.
Filter	Limits the packages displayed to those that meet the criteria you specify.
List	Displays a list of packages determined by the In-box and any filter that you set. To view or vote on a package, you select the package from this list.
Information	Displays information about the package that you select from the list.
Status	Describes communication between the Breeze Web interface and its host.

The information panel contains the following information in tabs as shown in Table 15-2.

Table 15-2 Tab information

Summary	General Information about the package, such as status and group information
Contents	A list of members included in this package. You can also Browse the member and its SCLM accounting information from here
Log	A log of Breeze activity against the package

Summary	General Information about the package, such as status and group information
Collisions	List members that are affected by other packages
Ballot Box	A list of the approver groups and approvers and the votes that they have cast, plus the current status
Notes	A list of notes that approvers may have made when they voted on a package

15.7.1 Selecting a package for viewing or voting

Before you can view or vote on a package, you must click the **Package ID** or its description in the list on the main panel to select it. Before you can do this, you must display the package in the list. To determine which packages are displayed in the list, click an **In-box** item at the top left of the main panel as shown in Table 15-3.

Table 15-3 Packages information

Requiring my approval	Displays packages that require your approval, that is, the packages in Pending status for which you are an Approver. When you log in, this is the default.
Emergency Packages	Displays only the emergency packages that require your approval.
Standard Packages	Displays only the standard packages that require your approval.
Packages by status	Displays all packages regardless of their status or approvers.
Pending	Displays packages that require approval and for which voting is still in progress.
Vetoed	Displays packages that have been voted on and vetoed.
Approved	Displays packages that have been voted on and approved.
Promoted	Displays packages that have been promoted successfully.
Promotion failed	Displays packages that were approved but, for some reason (such as a problem in SCLM), have not been promoted.

When the list displays the package you want, click on it in the list to select it.

To vote on a package for which you are an Approver, you must select the package from the list displayed when you click on **Requiring my approval** or one of its sub-items. This activates the **Approve** and **Veto buttons**, allowing you to cast a vote. If you select the same package from the “Packages by status” list, then these buttons remain disabled.

15.7.2 Filtering packages from the list

You can limit the list to display packages that meet certain criteria as shown in Table 15-4.

Table 15-4 Packages by criteria

Criteria	**Value**
Package ID	SCLM Package ID with 1-16 characters.
Promotion Window	The time frame in which the package can be promoted.
Build User ID	User ID of the person who built the package.
Promote User ID	User ID of the person who promoted the package.

Criteria	Value
Promote Date	Date the package was promoted.
Last Update User ID	User ID of the person who made the last update to the package.
Last Update Date	Date the package was last updated.

15.7.3 Voting on a package

You can vote on a package if all of these conditions are true:

1. You are an approver for the package. Before you can be an approver, your Breeze administrator must define the necessary Breeze records on the host.
2. The package is still in Pending status, that is, when you click **Requiring my approval** in the **In-box**, the package appears in the package list.
3. You have not already voted on the package.
4. You have selected **Requiring my approval** (or one of its sub-items: **Standard Packages** or **Emergency Packages**) in the **In-box**.

To vote on a package:

1. Click on the package in the list on which you want to vote.
2. To cast a vote for approval, click the **Approve button**.
3. To cast a vote against approval, click the **Veto button**.

- If any of the conditions listed above are not true, then these buttons are disabled, and you will not be able to cast a vote.
- A voting dialog appears that is similar to Figure 15-9 depending on whether you clicked on **Approve** or **Veto**.

Figure 15-9 Voting Dialog

This is the first in a sequence of voting dialogs. Each of these panels has a **Prior button** and a **Cancel button**. To step back through the sequence, click the **Prior** button. To return to the main panel without voting, click the **Cancel** button. On the first dialog in the sequence, shown above, both the **Prior** and **Cancel** buttons take you back to the main panel without voting.

4. To leave a note explaining your vote to other users, select **Add notes to package**.
5. Click the **Next button** to proceed.
6. If you chose **Do not add notes to package**, go to the next step. If you chose to add notes, then a dialog displays with a text box where you can enter your notes.
   - You can enter up to 480 characters.
   - To insert a line break in your notes, press **Enter**.

Archived

– When you finish entering your notes, click the **Next button** to proceed.

– The final voting dialog displays.

7. Click the **Submit** button to cast your vote as shown in Figure 15-10.

Figure 15-10 Submitting your vote

15.7.4 How voting results reaches approved or vetoed status

Here is how a package in pending status reaches either approved status or vetoed status:

Approved status

Every approver group for the package approves the package, that is, in every approver group:

- Every required approver has voted either “for” or “against”. A package can be approved even if one or more required approvers vote against it.
- The number of votes is equal to or greater than the quorum (minimum number of votes required for the approver group).
- There are a majority of “for” votes.

The package now can be promoted.

Vetoed status

One approver group for the package vetoes the package, that is, in one of the approver groups:

- Every required approver has voted either “for” or “against”. A package can be vetoed even if one or more required approvers vote for it.
- The number of votes is equal to or greater than the quorum (minimum number of votes required for the approver group).
- There are at least as many “against” votes as “for” votes.

If one approver group vetoes a package, then Breeze immediately sets the package to vetoed status even if required approvers in other approver groups have not yet voted.

Archived

15.8 Viewing package information

To view detailed information about a package:

1. Select the package you want from the package list.
2. Click one of the tabs in the information area of the main panel. These tabs are described under the headings that follow.

15.8.1 Summary tab

The Summary tab displays various information about the selected package as shown in Figure 15-11 and explained in Table 15-5.

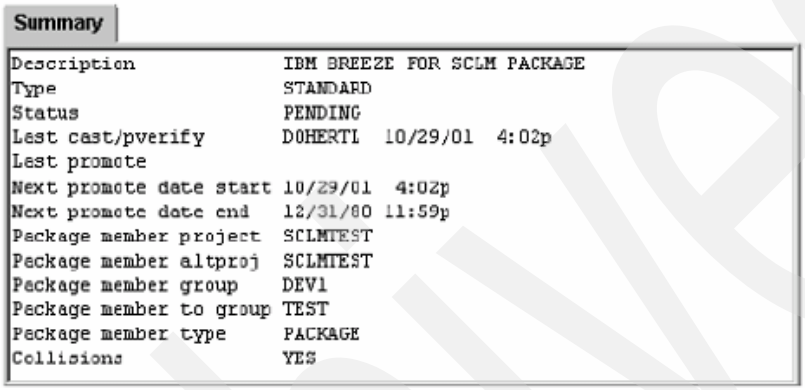


Figure 15-11 Summary tab

Table 15-5 Summary tab explanations

Description	If a developer used the SCLM Promote function in foreground to request promotion for this package, then this is the description that the developer entered on the Breeze pop-up window. Otherwise, if the developer used the SCLM promote service in a batch job, then this is the default description supplied by Breeze.
Type	Package type: **STANDARD** or **EMERGENCY**.
Status	Current package status.
Last cast/pverify	User ID, time, and date of the most recent invocation of the Breeze “promote verify” user exit for this package, that is, who last used the SCLM promote function for this package, and when they used it.
Last promote	User ID, time, and date of the most recent successful promotion of this package as recorded by the Breeze “promote purge” user exit.
Next promote date start/end	If the package reaches Approved status between these two dates, then the package can be promoted. Otherwise, the package cannot be promoted even if it is approved.
Package member	Package member details. The “**to group**” is the SCLM project group to which the package will be promoted.
Collisions	Indicates whether the current package is in collision with any other packages.

Archived

15.8.2 Contents tab

The **Contents** tab lists the names and inventory locations of the members in the selected package as shown in Figure 15-12. The tab lists each member at its source inventory location (the SCLM project group from where the member is to be promoted) and at its target inventory location (the SCLM project group to where the member will be promoted).

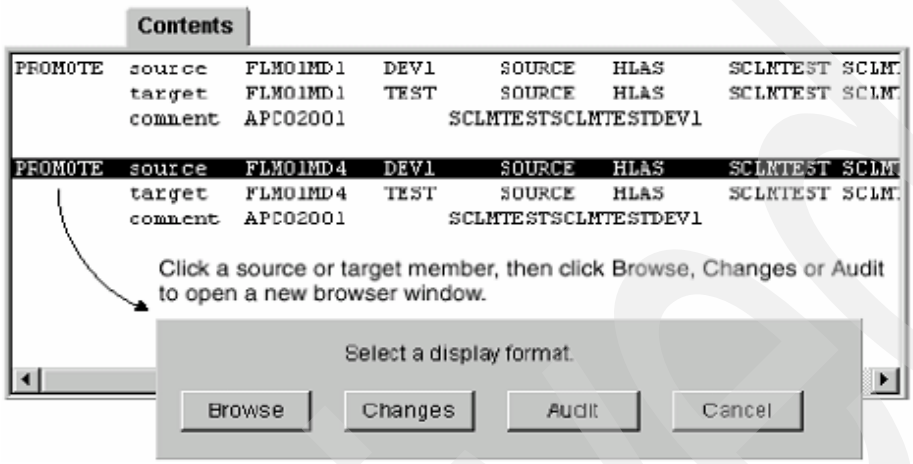


Figure 15-12 Contents tab

To view more detailed information about a member, click the entry for the member in the **Contents** tab. A dialog displays with the following choices, each of which opens a new browser window displaying the selected information:

Browse Displays the contents of the member.

Changes Displays a line-by-line comparison of the new (source) and old (target) members.

Audit Displays audit information for the member.

15.8.3 Log tab

The Log tab displays a history of the actions performed on a package as shown in Figure 15-13.

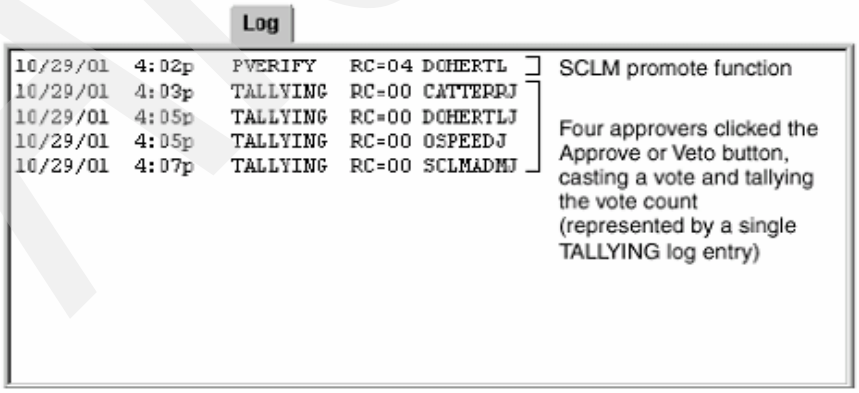


Figure 15-13 Log tab

Archived

For each action, the log displays the date and time, the return code, and the user ID that performed the action. If a package was promoted previously, then the package inherits the existing log, and the log accumulates as the package is promoted up the hierarchy.

15.8.4 Collisions tab

The **Collisions** tab, as shown in Figure 15-14, contains detailed collision information about the current package including which members are causing the collision. The collision may be current or historic depending on the status and location of the package.

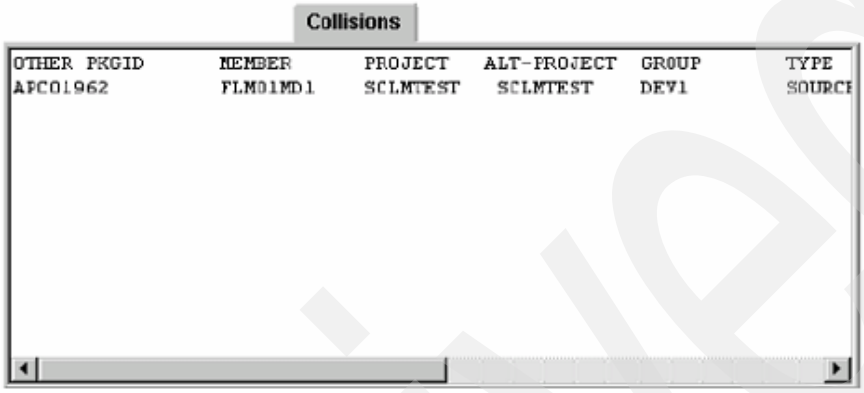


Figure 15-14 Collisions tab

15.8.5 Ballot Box tab

The **Ballot Box** tab, as shown in Figure 15-15, shows the details of the current voting status:

- The approver groups that are responsible for voting on the selected package
- The approvers in each approver group
- Who has voted
- How and when those approvers voted
- Whether a quorum has been reached

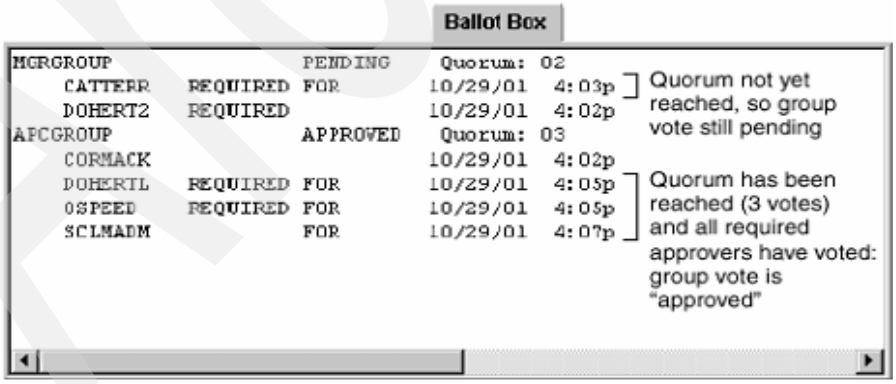


Figure 15-15 Ballot box tab

15.8.6 Notes tab

The **Notes** tab, as shown in Figure 15-16, displays any notes that were added by approvers who voted on the package.

Archived

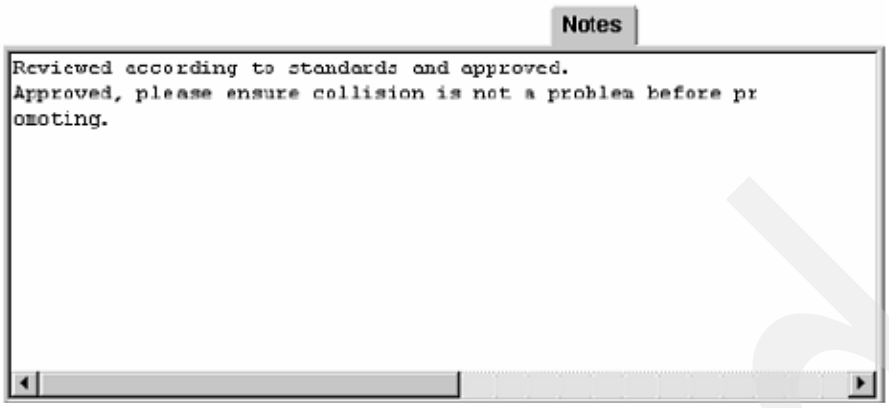


Figure 15-16 Notes tab

15.9 Breeze - other utility jobs

BZZSMJA1: Builds cross-reference list of users, packages and approver groups

BZZSMJV1: Casts and Tallies votes. You can use this job to vote using a batch job if you prefer.

BZZSMJD1: Clears, extracts and purges package records

BZZSMJ06: Runs a package report. You can use this to report on any and all packages in the database. It will give you the same basic information that you can find in the Breeze Web application, namely Status, Contents, Collisions, Notes, Votes and Log.

BZZSMJD1: Used to purge invalid or no longer relevant packages from the Breeze database, to clear the approval status of a existing package and to reset the status of packages in a incomplete status. Clear is used if a package has been promoted to a higher level but is then changed again.

15.9.1 Breeze sweep job

The Breeze sweep job (BZZSMJS1) can be used to promote all packages that are ready for promotion (Package status is approved and execution window is valid). It can be scheduled to run automatically using a job scheduler such as OPC, so on a hourly/nightly/weekly basis it automatically promotes packages that are ready for promotion (Package status is approved and execution window is valid). Before you use the sweep job (BZZSMJS1) you will have to customize it for your location with the same customization you made to FLMLIBS. Specifically, it must be customized with the naming convention for your ISPF and Breeze datasets and with any extra steplib datasets such as the DB2 load library that are needed by user exits.

The OK$CLEAR ddname in the JCL is optional, but we recommended that you include it. If included in the JCL it causes the automatic CLEAR package option to be included in the promote request. The Clear Package option will automatically clear the package if it exists at the next level of the hierarchy. This enables packages to be reused without having to manually clear same named packages using the BZZSMJD1 job. If this option is not set in the sweep job and the package exists in the higher group then Breeze will stop the promote as Breeze will not allow contents of a package to be overwritten unless the status of that package is set to blank. To disable the automatic CLEAR package option, comment out the OK$CLEAR DD statement.

Archived

Customized sweep job

The sweep job promotes all approved packages in all groups. Below we have a modification to the sweep job that will limit the promotions to a specific group as shown in Example 15-11. Using this modification you can have one sweep job for each of the groups in your hierarchy such as one for DEV1 and another for TEST.

If you compare this version of the sweep job to the base version of BZZSMJS1 that comes with Breeze you will see the exact changes that we made to the base version. Besides the usual customizations to add the job cards and update the Breeze and ISPF datasets, we added a step to filter the output of approved packages from the first step to only pass on to the promote step those promote statements for the TEST group. You will have to add a similar step to your sweep job and customize the parameters of the EXEC statement with the group that you want it executed for and change the SYSEXEC statement to point to the dataset where you stored the PROMSEL REXX. You will also need to change the PROMCARD file name in step 1 to APPRCARD. There is also some IF logic that you will also need to add. Your compare will point those lines out to you.

Example 15-11 SWPTEST - Breeze Sweep Job to promote packages in TEST

```
//SWPTEST  JOB (#ACCT),'BREEZE JOB',CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID
//* ------------------------------------------------------------------*
//*  NAME:    BZZSMJS1                                                *
//*  PURPOSE: SUBMIT ALL BREEZE FOR SCLM PACKAGES THAT ARE APPROVED   *
//*           AND HAVE A VALID EXECUTION WINDOW.                      *
//*  NOTE:    THE OK$CLEAR DD DUMMY WILL CAUSE THE CLEAR PACKAGE      *
//*           OPTION TO BE INCLUDED IN THE PROMOTE REQUEST. TO DISABLE*
//*           THE CLEAR TARGET PACKAGE OPTION, COMMENT OUT THE        *
//*           OK$CLEAR DDNAME.                                        *
//* ------------------------------------------------------------------*
//*                                                                   *
//*  TO USE THIS JCL, YOU MUST:                                       *
//*         1) INSERT VALID JOB CARD.                                 *
//*         2) REVIEW THE NAMING CONVENTIONS OF THE ISPF LIBRARIES.   *
//*            THIS SKELETON REFLECTS DEFAULT NAMING STANDARDS.       *
//*            PLEASE MODIFY TO MEET YOUR INSTALLATION STANDARDS.     *
//*         3) MAKE SURE THAT THE STEPLIB POINTS TO YOUR INSTALLATION *
//*            ISPF LOAD LIBRARIES, UNLESS THEY ARE LINKLISTED.       *
//*            REVIEW THIS WITH YOUR ISPF SYSTEM PROGRAMMER.          *
//*         4) MAKE SURE THAT THE STEPLIB POINTS TO THE BREEZE PRODUCT*
//*            LIBRARIES THAT CONTAIN THE CIGINI MODULE.              *
//*                                                                   *
//* ------------------------------------------------------------------*
//STEP1   EXEC PGM=BZZSAPS1
//STEPLIB  DD  DSN=BZZ.SBZZLOAD,DISP=SHR
//CIGLOG   DD  SYSOUT=*
//OK$CLEAR DD  DUMMY
//PROMCARD DD  DSN=&&APPRCARD,DISP=(,PASS),
//         DCB=(LRECL=80,RECFM=FB,BLKSIZE=31200),
//         SPACE=(TRK,(45,45)),UNIT=SYSALLDA
//CIGTMPLT DD *
   FLMCMD PROMOTE,%PROJECT%,%ALTERNATE%,%GROUP%,%TYPE%,+
   %MEMBER%,,N,C,PROMMSGS,PROMREPT,PROMEXIT,COPYERR,+
   Y,Y,,BLDMSGS,BLDREPT,BLDLIST,BLDEXITR
/*
//STEP1IF  IF (STEP1.RC = 0) THEN
```

```
//STEP2    EXEC PGM=IRXJCL,PARM='PROMSEL TEST'
//*--------------------------------------------
//* FILTER INPUT FROM STEP 1 - SEND OUTPUT
//* TO NEXT STEP ONLY FOR SPECIFIED GROUP
//*--------------------------------------------
//SYSTSPRT DD SYSOUT=(*)
//SYSTSIN  DD DUMMY
//APPRCARD DD DSN=&&APPRCARD,DISP=(OLD,DELETE)
//PROMCARD DD DSN=&&PROMCARD,DISP=(,PASS),
//         DCB=(LRECL=80,RECFM=FB,BLKSIZE=31200),
//         SPACE=(TRK,(45,45)),UNIT=SYSALLDA
//*--------------------------------------------
//* THE FOLLOWING DATASET MUST CONTAIN THE REXX
//*--------------------------------------------
//SYSEXEC  DD  DSN=SCLM.PROJDEFS.REXX,DISP=SHR
//STEP2IF  IF (STEP2.RC = 0) THEN
//*-------------------------------------------------------------------*
//* PASS PROMOTE REQUESTS TO SCLM                                     *
//*-------------------------------------------------------------------*
//PROMOTE PROC
//GENERO   EXEC PGM=IEBGENER
//SYSPRINT DD DUMMY
//SYSIN    DD DUMMY
//SYSUT1   DD DUMMY
//SYSUT2   DD DSN=&&CLIST0(TEMPNAME),UNIT=SYSALLDA,
//            SPACE=(TRK,(1,1,2),RLSE),
//            DISP=(NEW,PASS),DCB=(LRECL=80,
//            BLKSIZE=1600,DSORG=PO,RECFM=FB)
//*-------------------------------------------------------------------*
//* PROCESS THE SCLM FLMCMDS FROM STEP1                               *
//*-------------------------------------------------------------------*
//PROMO    EXEC PGM=IKJEFT01,REGION=4096K,TIME=1439,DYNAMNBR=200
//*-------------------------------------------------------------------*
//* STEPLIB LIBRARIES                                                 *
//*-------------------------------------------------------------------*
//STEPLIB  DD  DSN=BZZ.SBZZLOAD,DISP=SHR
//         DD  DSN=ISPFLPA.ZOSR8.SISPLPA,DISP=SHR
//         DD  DSN=ISPFLNK.ZOSR8.SISPLOAD,DISP=SHR
//         DD  DSN=DB2.V810.SDSNLOAD,DISP=SHR
//*-------------------------------------------------------------------*
//* ISPF LIBRARIES                                                    *
//*-------------------------------------------------------------------*
//ISPMLIB DD  DSN=BZZ.V1R1.SBZZMENU,DISP=SHR
//        DD  DSN=ISPF.ZOSR8.SISPMENU,DISP=SHR
//*
//ISPSLIB DD  DSN=BZZ.V1R1.SBZZSENU,DISP=SHR
//        DD  DSN=SCLM.PROJDEFS.SKELS,DISP=SHR
//        DD  DSN=ISPF.ZOSR8.SISPSENU,DISP=SHR
//        DD  DSN=ISPF.ZOSR8.SISPSLIB,DISP=SHR
//*
//ISPPLIB DD  DSN=BZZ.V1R1.SBZZPENU,DISP=SHR
//        DD  DSN=ISPF.ZOSR8.SISPPENU,DISP=SHR
//*
//ISPTLIB  DD  UNIT=VIO,DISP=(NEW,PASS),SPACE=(CYL,(1,1,5)),
//             DCB=(LRECL=80,BLKSIZE=19040,DSORG=PO,RECFM=FB),
```

Archived

```
//             DSN=                  TEMPORARY TABLE LIBRARY
//         DD  DSN=ISPF.ZOSR8.SISPTENU,DISP=SHR
//*
//ISPTABL  DD  UNIT=VIO,DISP=(NEW,PASS),SPACE=(CYL,(1,1,5)),
//             DCB=(LRECL=80,BLKSIZE=19040,DSORG=PO,RECFM=FB),
//             DSN=                  TEMPORARY TABLE LIBRARY
//*
//ISPPROF  DD  UNIT=VIO,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)
//*
//ISPCTL1  DD DISP=NEW,UNIT=SYSALLDA,SPACE=(CYL,(1,1)),
//            DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)   TEMPORARY FILE
//*                                                 TAILORING DATASET
//SYSTERM  DD SYSOUT=*
//*--------------------------------------------------------------------*
//* TEMPORARY CLIST CONTAINING COMMAND TO BE EXECUTED                  *
//*--------------------------------------------------------------------*
//SYSPROC  DD  DSN=&&CLISTO,DISP=(OLD,DELETE)
//         DD  DSN=BZZ.V1R1.SBZZCLIB,DISP=SHR
//         DD  DSN=ISPF.ZOSR8.SISPCLIB,DISP=SHR   CLIST LIBRARY OWO1230
//         DD  DSN=SCLM07.PROJDEFS.REXX,DISP=SHR
//*--------------------------------------------------------------------*
//* PASCAL ERROR MESSAGE FILE                                          *
//*--------------------------------------------------------------------*
//COPYERR  DD SYSOUT=(*),RECFM=FBA,LRECL=133,BLKSIZE=1330
//*--------------------------------------------------------------------*
//* PROMOTE USER EXIT OUTPUT FILE                                      *
//*--------------------------------------------------------------------*
//PROMEXIT DD DSN=&&PROMEXIT,
// DISP=(NEW,DELETE,DELETE),SPACE=(TRK,(5,10),RLSE),
// DCB=(LRECL=160,BLKSIZE=3200,RECFM=FB),UNIT=SYSALLDA
//*--------------------------------------------------------------------*
//* OUTPUT CARD                                                        *
//*--------------------------------------------------------------------*
//PROMREPT DD SYSOUT=*
//PROMMSGS DD SYSOUT=*
//SYSPRINT DD SYSOUT=(*)
//BLDMSGS  DD SYSOUT=*,DCB=(LRECL=80,RECFM=F,BLKSIZE=80)
//BLDLIST  DD SYSOUT=*,DCB=(LRECL=137,RECFM=VBA,BLKSIZE=3120)
//BLDREPT  DD SYSOUT=*,DCB=(LRECL=80,RECFM=FBA,BLKSIZE=3120)
//BLDEXITR DD DSN=&&BLDEXITR,
// DISP=(NEW,DELETE,DELETE),SPACE=(TRK,(5,10),RLSE),
// DCB=(LRECL=160,BLKSIZE=3200,RECFM=FB),UNIT=SYSALLDA
//*--------------------------------------------------------------------*
//* NLS TABLE AND TRANSLATION TABLE NAME FILE                          *
//*--------------------------------------------------------------------*
//ZFLMDD   DD DUMMY
//*--------------------------------------------------------------------*
//FLMMSGS  DD SYSOUT=(*)
//SYSTSPRT DD SYSOUT=(*)
//SYSTSIN  DD DUMMY
```

```
//PROMOTE  PEND
//*-------------------------------------------------------------------*
//* INVOKE THE PROMOTE PROC TO PROCESS QUALIFIED PACKAGES             *
//*-------------------------------------------------------------------*
//BREEZE   EXEC PROC=PROMOTE
//GENERO.SYSUT1  DD  DSN=&&PROMCARD,DISP=(OLD,DELETE)
//*
//PROMO.ZFLMDD DD *
   ZFLMNLST=FLMNLENU    ZFLMTRMT=ISR3278   ZDATEF=YY/MM/DD
/*
//PROMO.SYSTSIN DD *
   ISPSTART CMD(%TEMPNAME)
/*
//ENDIF2 ENDIF
//ENDIF1 ENDIF
```

Example 15-12 is the PROMSEL REXX. It only passes on PROMOTE statements that are for a specific group. The group is the single parameter that is passed in. This REXX must be placed in the location you specify in the SYSEXEC statement of the step that you add to your sweep job. You may want to improve this routine to handle wildcards in the group name passed to it.

Example 15-12 PROMSEL REXX to filter out PROMOTE statements to a specific group

```
/* REXX */
   Arg Group

   Say '*******************************************************'
   Say 'Promotion selected for groups : 'Group
   Say '*******************************************************'
   Say ' '

   j = 0
   i = 1
   PROM = 'No'
   /* read in all PROMOTE CARDs for packages in APPROVED status */
   "EXECIO * DISKR APPRCARD (FINIS STEM apprcard."

   If apprcard.0 = 2 then Do
      say 'There are no packages to process'
      exit 4
   End

   Parse var apprcard.1 cmd PROMOTE',' .

   Do Until (cmd = 'FLMCMD')
      j = j + 1
      promcard.j = apprcard.i
      i = i + 1
      Parse var apprcard.i cmd PROMOTE',' .
   End

   Do While (cmd <> 'IF')
      Parse var apprcard.i cmd PROMOTE','PROJ','ALT','PROMGRP',' .
      /* look for PROMOTE cards that match the specified group */
      If cmd = 'FLMCMD' & Group = PROMGRP Then
```

Archived

```
      Do
         PROM = 'Yes'
         j = j + 1
         promcard.j = apprcard.i
         i = i + 1
         writeName = 'Yes'
         /* write out PROMOTE cards until the next set is found */
         Parse var apprcard.i cmd PROMOTE',' .
         Do Until (cmd = 'FLMCMD' | cmd = 'IF')
            j = j + 1
            promcard.j = apprcard.i
            if writeName = 'Yes' then Do
               Parse var apprcard.i PkgName','rest
               say 'Package' strip(PkgName) 'will be promoted'
               writeName = 'No'
            End
            i = i + 1
            Parse var apprcard.i cmd PROMOTE',' .
         End
      End
      Else
         /* bypass PROMOTE cards until the next match is found */
         Do Until (cmd = 'FLMCMD' | cmd = 'IF')
            i = i + 1
            Parse var apprcard.i cmd PROMOTE',' .
         End

      If cmd = 'IF' Then
         Do while (i <= apprcard.0)
            j = j + 1
            promcard.j = apprcard.i
            i = i + 1
         End
   End

   If PROM = 'Yes' Then
   Do
      promcard.0 = j
      "EXECIO * DISKW PROMCARD (FINIS STEM promcard."
      RC = 0
   End
   Else
   Do
      Say '*******************************************************'
      Say '***  No requests found for promotion group 'Group
      Say '*******************************************************'
      RC = 4
   End

Exit RC
```

Archived

15.9.2 EAC and Breeze

If Enhanced Access Control (EAC) has been installed to protect your SCLM datasets then you must make a modification to BZZSMPKG, the voting job, to allow it to run correctly. The voting job needs to run in a ISPF address space in order for it to coexist with EAC. Please modify your version of BZZSMPKG to look like Example 15-13. You will have to modify the Breeze and ISPF datasets in this example with the equivalent datasets used at your installation.

Example 15-13 BZZSMPKG modified for use with EAC

```
//JC1      JOB (ACCT#),'NAME',CLASS=A,REGION=4096K,
//         MSGCLASS=H,MSGLEVEL=(1,1),
//JC1_USER,PASSWORD
//*  -------------------------------------------------------------------
//STEP010  EXEC PGM=IKJEFT01,REGION=300M,TIME=1439,DYNAMNBR=200
//STEPLIB  DD  DISP=SHR,DSN=BZZ.SBZZLOAD
//ISPPROF  DD  UNIT=VIO,DISP=(NEW,PASS),SPACE=(CYL,(10,10,10)),
//             DCB=(LRECL=80,BLKSIZE=19040,DSORG=PO,RECFM=FB),
//             DSN=                  TEMPORARY TABLE LIBRARY
//*
//ISPMLIB  DD  DSN=BZZ.SBZZMENU,DISP=SHR
//         DD  DSN=ISP.SISPMENU,DISP=SHR
//*
//ISPSLIB  DD  DSN=BZZ.V1R1.SBZZSENU,DISP=SHR
//         DD  DSN=ISP.SISPSENU,DISP=SHR
//*
//ISPPLIB    DD DSN=BZZ.V1R1.SBZZPENU,DISP=SHR
//           DD DSN=ISP.SISPPENU,DISP=SHR
//*
//ISPTLIB  DD  UNIT=VIO,DISP=(NEW,PASS),SPACE=(CYL,(10,10,10)),
//           DCB=(LRECL=80,BLKSIZE=19040,DSORG=PO,RECFM=FB),
//           DSN=                  TEMPORARY TABLE ENURARY
//         DD  DSN=ISP.SISPTENU,DISP=SHR
//*
//ISPTABL    DD UNIT=VIO,DISP=(NEW,PASS),SPACE=(CYL,(10,10,10)),
//           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)
//*
//ISPCTL1  DD DISP=NEW,UNIT=SYSDA,SPACE=(CYL,(1,1)),
//           DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)   TEMPORARY FILE
//*                                                TAILORING DATASET
//SYSUDUMP DD  SYSOUT=*
//SNAPDUMP DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//CIGLOG   DD  DSN=&&CLOG,
//             DISP=(NEW,PASS),UNIT=SYSALLDA,
//             DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1330),
//             SPACE=(CYL,(1,5))
//CIGRPT   DD   SYSOUT=*
//CIGIN DD   *
//SYSPRINT DD  SYSOUT=*
//ISPLOG   DD  DUMMY
```

Archived

```
//FLMMSGS  DD  SYSOUT=*
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
ISPSTART PGM(BZZSAPV1)
//*
```

Archived

16

Enhanced Access Control: Bridging the gap - SCLM to RACF

In this chapter we describe setting up Enhanced Access Control (EAC) applications and profiles to protect your SCLM resources. As SCLM uses normal PDS, PDSE, and VSAM datasets to store members and metadata for those members, there is the possibility that SCLM resources might be accidentally updated outside the control of SCLM, for example, using ISPF edit on an SCLM controlled member rather than SCLM edit.

What happens in this case is that the member is saved in the PDS or PDSE but the SCLM metadata stored in the SCLM account file is not updated, so the date and time values become out-of-sync. If the member updated outside of SCLM control is in a development group, then there is really no problem, as the member just needs to be saved within SCLM. But if the member is in a group higher in the hierarchy, then it is more difficult to fix the SCLM accounting metadata. This is the main reason that EAC was developed, to prevent such accidental updates from occurring.

Note: If you only want to protect against accidental updates from ISPF edit, you can use an ISPF configuration setting called SCLM_WARNING_LEVEL. This indicates the level of SCLM checking that should be done when SCLM-controlled members are processed outside SCLM. Valid values are:

NONE	No checking is done. SCLM-controlled members can be edited and processed outside SCLM.
WARN	If an SCLM-controlled member is processed by Edit or Reset Statistics, a message is displayed to warn the user that the SCLM accounting data will be invalidated by the pending request.
ERROR	When edit of an SCLM member is attempted, an error message is displayed and the edit is denied.

This setting, however, does not protect the member from ISPF 3.3 (Move/Copy), IEBCOPY, or other methods of overwriting the member, only ISPF edit.

© Copyright IBM Corp. 2007. All rights reserved.

Additionally, by providing the mechanism to protect resources such that only SCLM programs have the authority to update certain resources, there are additional benefits in that you can be even more granular with your authority with regard to SCLM. For example, you can say that a development group is only allowed to be updated by the SCLM Edit and Build processes by a certain group, plus the change management group is only allowed to update certain resources via the Promote process. This allows you to define who is allowed to do what in terms of updating the hierarchy, and how they are allowed to perform that update.

In this chapter we provide a brief explanation of EAC terminology before considering some usage scenarios. Additionally, we explore some of the special requirements required concerning Breeze with EAC and IEBCOPY update through EAC.

Restriction: EAC currently works only with RACF. The EAC processor is hooked into the ICHRCX02 RACF post processing user exit such that when RACF flags a violation control is passed to the ICHRCX02 exit. EAC processing is then invoked to check if the access that was denied through RACF is actually permitted through EAC rules. If it is, then the violation is overridden.

Similar processing could be handled in other security managers, such ACF2 or Top Secret, if those products had a means of defining “program pathing”, which is the process of allowing access to certain resources only via certain programs.

Archive

16.1 What happens when EAC is not used

Normally it is possible to protect most SCLM data sets from malicious harm through RACF, but there still exists the possibility of accidental editing outside of SCLM. If this happens at the development group, that is not such an issue, as the member will just need to be saved and rebuilt within SCLM to fix the relationship between the ISPF statistics and the SCLM metadata. However, as a developer might require update authority on the group data sets above the development group in order to effect a promotion, there is also the possibility that the TEST level of data sets could be edited outside of SCLM.

To explain the relationship between the actual member and its accounting information, let us look at the following member in SCLM, shown in Figure 16-1.

```
------------------------------------------------------------------------------
Member List : SCLM07.DEV1.COBOL - HIERARCHY VIEW -              Member 1 of 1
Command ===>                                                 Scroll ===> CSR

A=Account      M=Map        B=Browse      D=Delete      E=Edit
V=View         C=Build      P=Promote     U=Update      T=Transfer

    Member   Status      Account  Language   Text       Chg Date   Chg Time
    RDBKC01             DEV1     CBCID2DT   DEV1       2007/01/03 01:07:35
****************************** Bottom of data ********************************
```

Figure 16-1 SCLM member showing date and time

We can see the changed date and time of the member when we list the member in option 3.1. This date and time is the date and time stored in the ISPF statistics. The same member can be seen by listing the data set in ISPF edit as shown in Figure 16-2.

```
------------------------------------------------------------------------------
EDIT     SCLM07.DEV1.COBOL                                  Row 00001 of 00001
Command ===>                                                  Scroll ===> PAGE
   Name     Prompt   Lib    Size    Created           Changed            ID
. RDBKC01              1     166   2006/11/22   2007/01/03 01:07:35    PCECK
  **End**
```

Figure 16-2 SCLM member listed through ISPF edit

The final thing to look at is the SCLM metadata that is stored in the VSAM account file. When the member statistics and metadata statistics are in sync, then the account record for the member will have the same date and time as shown in Figure 16-3.

Archived

```
         SCLM07.DEV1.COBOL(RDBKC01): Accounting Record
Command ===>


  Physical Data Set . : SCLM07.DEV1.COBOL
  Accounting Status . : EDITABLE        Change Group . . . . : DEV1
  Change User ID  . . : PCECK           Authorization Code . : P
  Member Version  . . : 9               Auth. Code Change  . :
  Language  . . . . . : CBCID2DT        Translator Version . :
  Creation Date . . . : 2006/11/22      Change Date  . . . . : 2007/01/03
  Creation Time . . . : 15:51:32        Change Time  . . . . : 01:07:35
  Promote User ID . . :                 Access Key . . . . . :
  Promote Date  . . . : 0000/00/00      Build Map Name . . . :
  Promote Time  . . . : 00:00:00        Build Map Type . . . :
  Predecessor Date  . : 2006/12/08      Build Map Date . . . : 2007/01/03
  Predecessor Time  . : 05:18:35        Build Map Time . . . : 01:07:35

 Enter "/" to select option
   Display Statistics
   Number of Change Codes      : 0
   Number of Includes          : 3
   Number of Compilation Units : 0
   Number of User Entries      : 0
```

Figure 16-3 SCLM Accounting record for RDBKC01 member

If we now edit this member outside of SCLM, in ISPF edit, save it, and look at the member listed in SCLM option 3.1, we see that the date and time of the member has changed, shown in Figure 16-4.

```
------------------------------------------------------------------------------
Member List : SCLM07.DEV1.COBOL - HIERARCHY VIEW -              Member 1 of 1
Command ===>                                                 Scroll ===> CSR

A=Account      M=Map        B=Browse       D=Delete      E=Edit
V=View         C=Build      P=Promote      U=Update      T=Transfer

    Member   Status      Account  Language   Text       Chg Date   Chg Time
    RDBKC01              DEV1     CBCID2DT   DEV1       2007/01/08 20:09:47
****************************** Bottom of data ********************************
```

Figure 16-4 SCLM member showing date and time after illegal edit

If we were to look at the account record again, we would see that it is has exactly the same information as what is seen in Figure 16-3 above. The problem with the mismatch between the member in the data set and the SCLM metadata for the member can be seen when the member is built or promoted. If we build this source member, SCLM will return to us the following message, shown in Figure 16-5.

```
FLM49000 - INVOKING BUILD PROCESSOR
FLM09002 - THE BUILD REPORT WILL APPEAR IN DOHERTL.BUILD.REPORT45
FLM09006 - THE BUILD LISTING WILL APPEAR IN DOHERTL.BUILD.LIST45
FLM42000 - BUILD PROCESSOR INITIATED - 20:15:21 ON 2007/01/08
FLM07007 - ACCOUNTING INFORMATION IS NOT ACCURATE FOR
           MEMBER: RDBKC01  TYPE: COBOL
           ACCOUNTING GROUP: DEV1     MEMBER GROUP: DEV1
FLM45000 - ERROR PROCESSING CURRENT BUILD
FLM46000 - BUILD PROCESSOR COMPLETED - 20:15:22 ON 2007/01/08
FLM09008 - RETURN CODE = 8
```

Figure 16-5 Accounting error when ISPF stats and metadata are out of sync

As we can see from the message, SCLM will not allow the member to be built due to a mismatch between the accounting data and the ISPF statistics. This also means that the member cannot be promoted, which protects the integrity of the modules further up in the hierarchy. This is a simple scenario, and in this case it can be fixed quite easily by just saving the member in SCLM in the development group. But if the same edit had been performed on a TEST or PROD version of the member, then it would be a bit more complicated to fix the situation, possibly requiring the member that has been edited illegally being saved in the development group and being promoted through the hierarchy.

This is the most common error in SCLM that EAC is going to prevent, but additionally EAC will give more granularity as to who can update what resource using which SCLM function.

16.2 EAC definitions

In order to understand some of the examples later in this chapter, we take a brief look at what definitions are made in EAC to achieve the protection we need.

Enhanced Access Control for SCLM includes two types of definitions to provide access control. Profiles identify the data sets to be controlled, along with the access rules applicable for those data sets. This is similar to RACF, in the sense of who (user or group) has access to the data set resource. Applications define the SCLM program environments that must be used in order to gain access to the data sets — in other words, what programs have authority to update the data set resource.

The Profile and Application definitions are maintained via online panels within the ISPF Dialog. These are then saved in the Enhanced Access Control for SCLM Rule File. A utility program loads the Profile and Application definitions into memory, and the Enhanced Access Control for SCLM Validation Routine uses these definitions to determine how access controls should be applied to data sets under its management.

16.2.1 Profiles

A Profile identifies a data set or RACF generic data set Profile to be validated by Enhanced Access Control for SCLM. Discrete Profiles like SCLM.DEVT.SOURCE describe a specific data set, and Enhanced Access Control for SCLM will always validate against the discrete data set Profile if one has been defined. An illustration is shown in Figure 16-6. Generic Profiles such as SCLM.DEVT.* describe multiple data sets, and match the coding rules for RACF generic data set Profiles. Enhanced Access Control for SCLM will validate against a generic Profile if a discrete data set Profile has not been defined and RACF performed validation against a generic Profile of the same name.

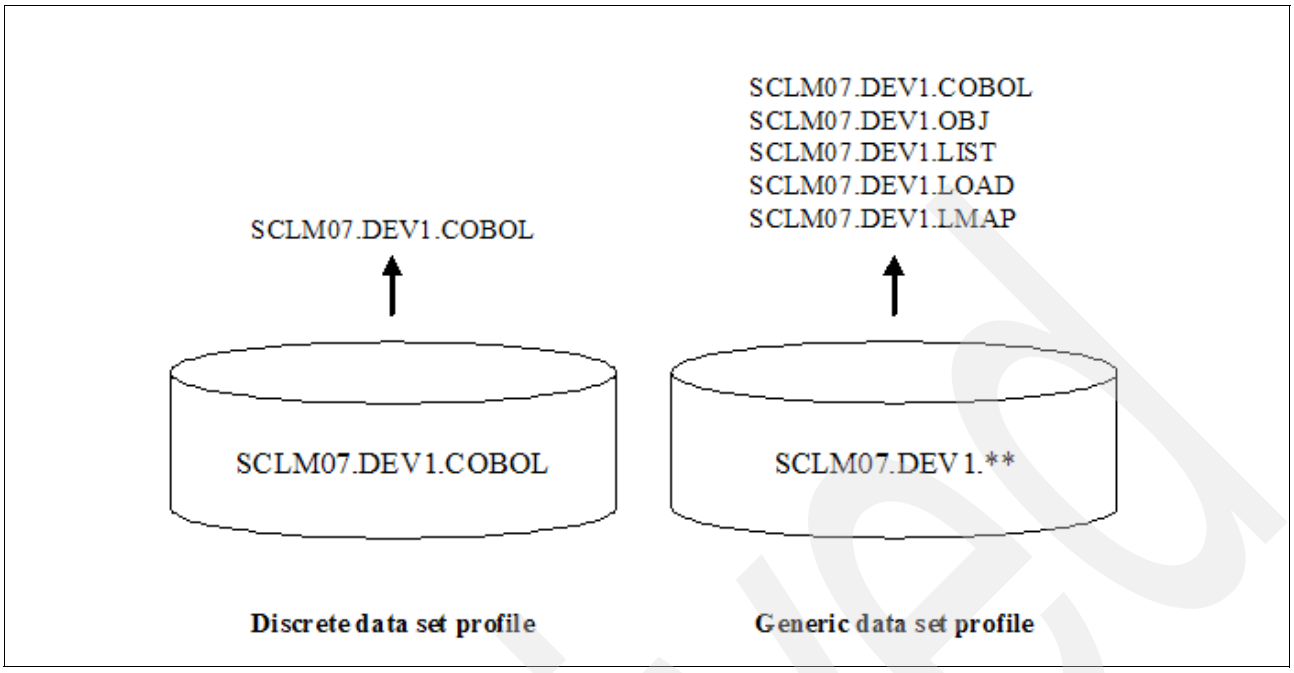


Figure 16-6 Discrete data set profiles and generic data set profiles

The Profiles also define the access rules validated by EAC. These access rules determine who can have access privileges, and the controlling Applications or program environment required in order for those access privileges to apply. The access rule has three parts:

Application Describes the SCLM programs as Applications/Functions

User Describes the RACF user IDs or Groups, or * for all users

Access Describes the access privilege, such as NONE, READ, UPDATE, CONTROL, and ALTER.

Example 16-1 shows a sample Profile definition and its access rules. This is a discrete data set Profile, as it controls only the one data set called SCLM07.DEV1.COBOL.

Example 16-1 Sample EAC Profile

```
Profile: SCLM07.DEV1.COBOL

  Application  Function  User/Group  Access
1 SCLM         EDIT      FRED        READ
2 SCLM         EDIT      PGMRS       UPDATE
3 SCLM         PROMOTE   *           NONE
4 SCLM         PROMOTE   MNGRS       UPDATE
```

This Profile has four access rules:

Access rule 1 This shows that the user ID FRED is assigned the READ access privilege when the SCLM07.DEV1.COBOL data set is accessed via the SCLM EDIT Application and Function program list.

Access rule 2 This is for the RACF Group called PGMRS. They will be assigned the UPDATE privilege if they access the data set by performing an SCLM EDIT.

Access rule 3 This applies when the Application and Function of SCLM PROMOTE is used. In this case, the user value of * applies to all users, and they are assigned a privilege of NONE.

Access rule 4 This also applies to the Application and Function of SCLM PROMOTE, but this case is specifically for the RACF Group called MNGRS, who will be assigned the access privilege of UPDATE.

Enhanced Access Control for SCLM matches the data set access request against its definitions for the Profile, Application, and User/Group to determine the appropriate access privilege. Most-to-least specific matching is performed, therefore:

- Discrete data set Profiles take precedence over generic data set Profiles.
- RACF user IDs take precedence over RACF Groups, which take precedence over * for all users.

16.2.2 Applications

The principal feature of EAC is its ability to limit data set access via specific Applications. These Applications describe the programs that must be used to obtain access: a control program called the High Program and a service program called the Low Program. For example, ISRSCLM is an SCLM High Program, and FLMP can be used as the Low Program for the SCLM Promote function. By defining both the High and Low Programs, SCLM and its various sub-functions can be secured.

Defining the SCLM programs as Applications simplifies access rule writing, as multiple combinations of SCLM programs can be grouped into one or a few applications. The Applications can also be assigned Function names to distinguish various SCLM functions or services. Whereas the Application name is mandatory, the Function name is optional. It provides flexibility in the way Applications are defined.

Example 16-2 shows a sample definition for the Application and Function name of SCLM Promote. While these names are arbitrary, they should be assigned values that readily distinguish their use. You could equally have assigned the Application name of Promote and not used a Function name.

Example 16-2 Sample EAC Application/Function for Promote

```
Application   SCLM                 Application   PROMOTE
Function      PROMOTE        or    Function

High Program  Low Program          High Program  Low Program
FLMCMD        FLMP                 FLMCMD        FLMP
FLMS$SRV      FLMP                 FLMS$SRV      FLMP
ISRSCLM       FLMP                 ISRSCLM       FLMP
```

This Application contains three High and Low Program pairs:

Program pair 1 This specifies the High Program of FLMCMD. This is the controlling program when the SCLM Command Interface is executed. The Low Program is FLMP, because this is the program that it controls in the SCLM Promote service.

Program pair 2 This specifies the High Program of FLMS$SRV. This is the controlling program when the SCLM FLMLNK Subroutine or Call Interface is executed.

Program pair 3 This specifies the High Program of ISRSCLM. This is the controlling program when SCLM is executed online via TSO.

For a full explanation of the EAC product, refer to the manual, *Enhanced Access Control for SCLM for z/OS User's Guide,* SC27-1591.

16.2.3 Understanding the relationship between RACF and EAC

EAC does not replace RACF, but it enhances RACF controls. EAC processing is invoked through the RACF post validation exit ICHRCX02. When a user accesses a data set, then RACF validation is invoked as normal. If the user is not permitted to update the resource through RACF, they will get a RACF violation (S913). Prior to the violation being issued to the user's terminal or job, control is passed to ICHRCX02, where control is then passed to EAC. EAC checks its own rule tables to see if the user has access. If they do, then the RACF violation is overridden and the update is allowed to continue, otherwise the violation is reported as it would normally be.

If you are going to protect a resource with EAC, you need to also define a profile in RACF for that resource. There are two ways that this can be done:

- By defining a one-for-one relationship between the RACF profile and the EAC profile. For example, if you had a profile SCLM07.DEV1.** in RACF you would also define a SCLM07.DEV1.** profile in EAC. The profile in RACF would be defined with no ACCESS rules and a UACC or either READ or NONE depending on your security requirements. The profile defined in EAC would then have the specific access requirements defined.
- By defining a generic RACF profile such as SCLM07.DEV1.** and then defining discrete profiles in EAC.

If you are going to implement EAC for an existing project, then there will be two pieces of work to do: first, to create the EAC profiles so that everyone has the access they require; and second, to remove access through RACF so that all access is handled through EAC.

Important: EAC is a product designed to supplement RACF to provide more granularity for SCLM control. EAC can, however, be used for non-SCLM data set access as well. For example, a person who has access to define profiles in EAC and reload rules into memory can give themselves access to SYS1.** if they wish. To this end, it is noted in the EAC documentation that the EAC administrator should also be a RACF administrator.

16.3 Defining a sample access strategy for an SCLM project

Taking the SCLM project we have used throughout most of this book as an example, we can define certain access rules that we want to implement to control access to the various groups in the hierarchy. If is good practice in SCLM to split your SCLM project account files such that there is one account file per group, this way they can be protected with EAC in the same way that the group data sets are protected. For example, the rules used to protect the production SCLM data sets can also be applied to the production SCLM account file.

Figure 16-7 shows a sample project with three levels in the hierarchy, DEV1, TEST and PROD. Let us assume that we have two distinct groups that need authority to parts in this hierarchy: a development group called DEVELOP and a change management team called CNGCTRL. The developers in the development group will have access to update all parts in the DEV1 group. They also need update authority to all parts in the TEST group, but only via the PROMOTE service.

The personnel in the change management group do not have any authority to edit parts in the DEV1 group. However, for promotion to production, they are going to be the ones to create the change control packages to drive that promotion. So the change control team must be able to edit the PACKAGE type at TEST. They do this using an alternate project definition that defines TEST as the lowest level in the hierarchy. This is required by SCLM, as only the lowest level in the hierarchy can be edited. They will also have update capability on the TEST and PROD levels of the hierarchy, but only through the PROMOTE service.

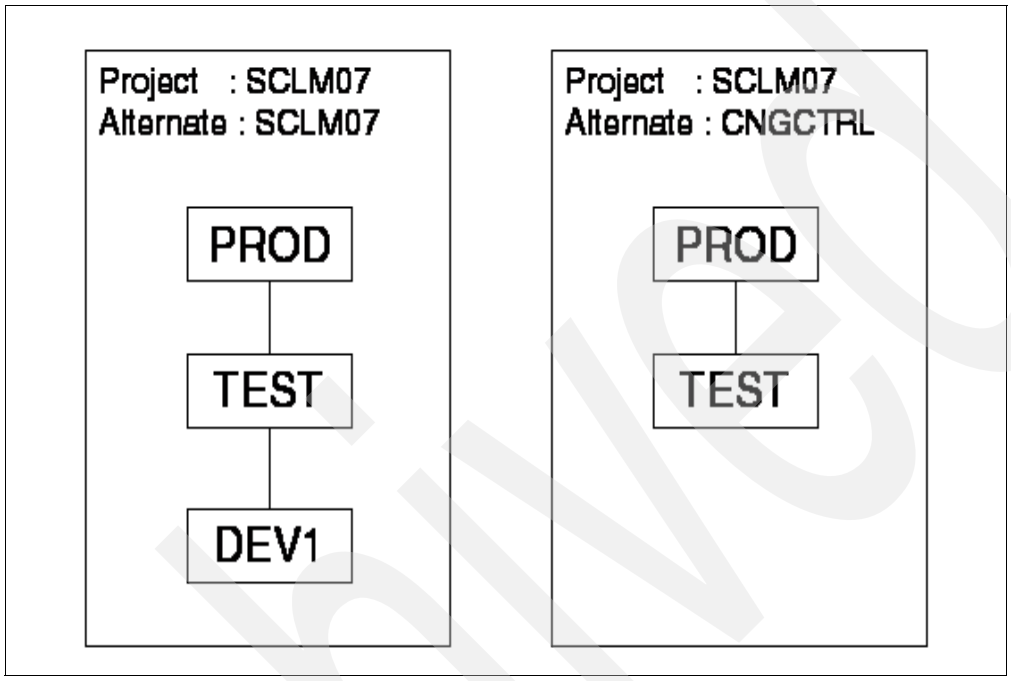


Figure 16-7 Development SCLM project and change control alternate project

16.3.1 Creating rules for development

For our definitions for SCLM we are using the default Application/Function pairs that are shipped with EAC. The first definitions we need to make are those for the development teams. All developers are added to a RACF group of DEVELOP. Currently the only RACF group that exists to protect all of our SCLM project is SCLM07.**, so we need to create RACF profiles for each of the groups. The following RACF profiles are created with a UACC=READ and no specific access rules:

- SCLM07.DEV1.**
- SCLM07.TEST.**
- SCLM07.PROD.**

We also create RACF profiles for the SCLM account files. We have set our project up to use separate account files for each level in the hierarchy as shown in Figure 16-8. There is a general account file for all of the development level groups call SCLM07.ACCOUNT.FILE. Then, using the FLMALTC macro, we define a separate account file for TEST called SCLM07.TEST.ACCOUNT.FILE, and a separate account file for PROD called SCLM07.PROD.ACCOUNT.FILE.

The following profiles are created in RACF for the account files with a UACC=READ, but again, there are no specific access rules:

- SCLM07.ACCOUNT.FILE
- SCLM07.PROD.ACCOUNT.**
- SCLM07.TEST.ACCOUNT.**

You can use a mixture of discrete and generic profiles in EAC (Figure 16-8).

```
         TITLE '*** PROJECT DEFINITION FOR SCLM07       ***'
SCLM07   FLMABEG
*
*****************************************************************
*                PROJECT CONTROLS
*****************************************************************
*
*
         FLMCNTRL ACCT=SCLM07.ACCOUNT.FILE,                     C
               VERS=SCLM07.VERSION.FILE,                        C
               VERCOUNT=5,                                      C
               MAXVIO=999999,                                   C
               VIOUNIT=VIO
*
*----------------------------------------------------------------*
*                ACCOUNT INFORMATIONS                            *
*----------------------------------------------------------------*
TESTALTC FLMALTC  ACCT=SCLM07.TEST.ACCOUNT.FILE,                 X
               VERS=SCLM07.VERSION.FILE
PRODALTC FLMALTC  ACCT=SCLM07.PROD.ACCOUNT.FILE,                 X
               VERS=SCLM07.VERSION.FILE
*
*        ***********************************************************
*        *  DEFINE THE GROUPS                                      *
*        ***********************************************************
*
DEV1     FLMGROUP AC=(P,D),KEY=Y,PROMOTE=TEST
DEV2     FLMGROUP AC=(P,D),KEY=Y,PROMOTE=TEST
TEST     FLMGROUP AC=(P),KEY=Y,PROMOTE=PROD,ALTC=TESTALTC
PROD     FLMGROUP AC=(P),KEY=Y,ALTC=PRODALTC
EMER     FLMGROUP AC=(EMER),KEY=Y,PROMOTE=PROD
*
```

Figure 16-8 Project definition showing use of alternate account files

Archived

We now create matching profiles in EAC for the development level and allow the DEVELOP RACF group to have access to them via all SCLM functions. To do this, we use option **3** within the EAC dialog and then enter the **I** (Insert) command on the empty list as shown in Figure 16-9.

```
------------------------------------------------------------------------------
                              Profile Selection             Profile not found
Command ===>                                                  Scroll ===> PAGE

Profile Prefix (Use + for all Profiles)
         +

Profile Processing
   Enter '/' for selection list

         Profile
     i
****************************** Bottom of data ********************************
```

Figure 16-9 Entering your first profile

We define the profile with the same name as the RACF profiles we have created. So the first one we do is the SCLM07.DEV1.** profile. We can also give it a description if we wish. We use the SCLM/ALL Application/Function pair for all SCLM functions. Figure 16-10 shows the profile we have created.

```
------------------------------------------------------------------------------
                            Profile Maintenance                 Row 1 from 1
Command ===>                                                  Scroll ===> PAGE

  Profile:  SCLM07.DEV1.**
  Data:
  EAC profile to protect SCLM07.DEV1.** data sets



    Enter a prefix value to limit the display of each column
        Application  Function     User/Group   Access
        *            *            *            *

    Enter '/' to select option or overtype rules

        Application +  Function +  User/Group  Access +
        SCLM           ALL          DEVELOP     UPDATE
****************************** Bottom of data ********************************
```

*Figure 16-10 Adding an EAC profile for SCLM07.DEV1.**

Archived

We save the profile by pressing PF3. Next we create a profile for the account file by entering an **I** next to the profile we have just created for SCLM07.DEV1.**. The profile is called SCLM07.ACCOUNT.FILE and the rules defined are the same as for SCLM07.DEV1.** as shown in Figure 16-11.

```
------------------------------------------------------------------------------
                             Profile Maintenance                 Row 1 from 1
Command ===>                                                  Scroll ===> PAGE

  Profile:  SCLM07.ACCOUNT.FILE
  Data:
  EAC profile to protect SCLM07 account file



    Enter a prefix value to limit the display of each column
        Application  Function     User/Group   Access
        *            *            *            *

    Enter '/' to select option or overtype rules

        Application +  Function +  User/Group  Access +
        SCLM           ALL          DEVELOP     UPDATE
****************************** Bottom of data ********************************
```

Figure 16-11 Adding an EAC profile for SCLM07.ACCOUNT.FILE

Pressing PF3 saves the profile.

Tip: When defining the rules, you can use PF4 to bring up a list of available applications, functions, and access rules.

We now need to create some rules for the TEST level in the hierarchy, as the development teams will have authority to promote code up to the TEST group. So again, next to one of the rules we have defined, we enter an **I** to create new rule for the SCLM07.TEST.** profile. This profile uses the SCLM/PROMOTE application/function, as this is the only function with which the development team is allowed to update the TEST level of the project. Figure 16-12 shows the new profile.

This is done similarly for the TEST account file profile. We create a profile in EAC and give the DEVELOP group the same access that we gave to the SCLM07.TEST.** profile.

The development team also require read access to PROD to draw down modules, but this is covered by the UACC=READ access on the RACF profile. As well as keeping the EAC rules limited to just the UPDATE rules, it also means that you do not have to define specific applications/functions and rules for any non-SCLM program that might need read access to the data sets. For example, the SuperC function to search through a data set would require a rule defined for it if there were no UACC=READ on the RACF profile. UACC=READ simplifies what you have to specify to EAC.

Archived

```
------------------------------------------------------------------------------
                            Profile Maintenance                 Row 1 from 1
Command ===>                                                  Scroll ===> PAGE

  Profile:  SCLM07.TEST.**
  Data:
  EAC profile for the SCLM07.TEST.** data sets and account file



    Enter a prefix value to limit the display of each column
        Application  Function     User/Group   Access
        *            *            *            *

    Enter '/' to select option or overtype rules

        Application +  Function +  User/Group  Access +
        SCLM           PROMOTE     DEVELOP     UPDATE
****************************** Bottom of data ********************************
```

*Figure 16-12 EAC profile to allow DEVELOP to update SCLM07.TEST.** on promote*

These are all the profiles required for the development team.

16.3.2 Creating rules for change control team

We now create the profiles for the change control team. First we modify the profile we have just created to give similar access to the change control team on the SCLM07.TEST.** profile as the development team. The change control team is only allowed to update the TEST level of the project with the promote function. The exception to this is the fact that the change control team is allowed to create new packages in the SCLM07.TEST.PACKAGE data set.

This data set is protected with a separate profile discussed shortly. For the purpose of building the package in SCLM07.TEST.PACKAGE, the change control team will need read access to all the TEST level datasets via the build function. So in addition to allowing update via the promote function, we give read access via the build function. This is shown in Figure 16-13.

Archived

```
------------------------------------------------------------------------------
                            Profile Maintenance                 Row 1 from 2
Command ===>                                                  Scroll ===> PAGE

  Profile:  SCLM07.TEST.**
  Data:
  EAC profile for the SCLM07.TEST.** data sets and account file



    Enter a prefix value to limit the display of each column
        Application  Function     User/Group   Access
        *            *            *            *

    Enter '/' to select option or overtype rules

        Application +  Function +  User/Group  Access +
        SCLM           PROMOTE     CNGCTRL     UPDATE
        SCLM           PROMOTE     DEVELOP     UPDATE
****************************** Bottom of data ********************************
```

*Figure 16-13 EAC profile to allow CNGCTRL to update SCLM07.TEST.***

We now update the profile for the TEST account file to add the rules for the change control team. They need to have update using all SCLM functions, as they will be using the edit, build and promote functions to update the TEST account file. The profile after the rule has been added is shown in Figure 16-14.

```
------------------------------------------------------------------------------
                            Profile Maintenance                 Row 1 from 2
Command ===>                                                  Scroll ===> PAGE

  Profile:  SCLM07.TEST.ACCOUNT.**
  Data:
  Profile for TEST level account file



    Enter a prefix value to limit the display of each column
        Application  Function     User/Group   Access
        *            *            *            *

    Enter '/' to select option or overtype rules

        Application +  Function +  User/Group  Access +
        SCLM           ALL         CNGCTRL     UPDATE
        SCLM           PROMOTE     DEVELOP     UPDATE
****************************** Bottom of data ********************************
```

Figure 16-14 TEST account file profile with CNGCTRL rule added

Archived

Next we create a new rule for SCLM07.PROD.**, in the same way as we have done before. The rule definition is shown in Figure 16-15. Similarly, we have to create the SCLM07.PROD.ACCOUNT.** profile with the same UPDATE rule.

```
------------------------------------------------------------------------------
                            Profile Maintenance                 Row 1 from 1
Command ===>                                                  Scroll ===> PAGE

  Profile:  SCLM07.PROD.**
  Data:
  EAC profile for the SCLM07.PROD.** data sets and account file



    Enter a prefix value to limit the display of each column
        Application  Function     User/Group   Access
        *            *            *            *

    Enter '/' to select option or overtype rules

        Application +  Function +  User/Group  Access +
        SCLM           PROMOTE     CNGCTRL     UPDATE
******************************* Bottom of data ********************************
```

*Figure 16-15 EAC profile to allow CNGCTRL to update SCLM07.PROD.** on promote*

One of the other functions to which the change control team are going to require access, is the capability to create a PACKAGE member and build that member, because that is the vehicle that they will use to drive the promote. They must be able to do this at the TEST level, so in SCLM we have created an alternate project to allow the TEST level to be edited as shown in Figure 16-16.

```
*
*-------------------------------------------------------------------*
*              ACCOUNT INFORMATIONS                                 *
*-------------------------------------------------------------------*
TESTALTC FLMALTC  ACCT=SCLM07.TEST.ACCOUNT.FILE,                     X
               VERS=SCLM07.VERSION.FILE
PRODALTC FLMALTC  ACCT=SCLM07.PROD.ACCOUNT.FILE,                     X
               VERS=SCLM07.VERSION.FILE
*
*        ************************************************************
*        *  DEFINE THE GROUPS                                       *
*        ************************************************************
*
TEST     FLMGROUP AC=(P),KEY=Y,PROMOTE=PROD,ALTC=TESTALTC
PROD     FLMGROUP AC=(P),KEY=Y,ALTC=PRODALTC
*
```

Figure 16-16 CNGCTRL alternate project definition

However, we do not want the change control team to have update authority on all data sets in the TEST level of the hierarchy. So we can create a discrete EAC profile that will just provide rules for the SCLM07.TEST.PACKAGE data set.

As we are using a discrete profile in EAC, we do not need to create this profile in RACF, which will use the SCLM07.TEST.** profile when the package data set is accessed. As there is no access through RACF, the violation is passed through to EAC, and EAC will match on the discrete profile for SCLM07.TEST.PACKAGE before matching on SCLM07.TEST.**. So we create a profile for SCLM07.TEST.PACKAGE that is allowed to be updated by the change control team, using all SCLM functions. This profile is shown in Figure 16-17.

```
------------------------------------------------------------------------------
                            Profile Maintenance                 Row 1 from 1
Command ===>                                                  Scroll ===> PAGE

  Profile:  SCLM07.TEST.PACKAGE
  Data:
  Discrete profile for SCLM07.TEST.PACKAGE



    Enter a prefix value to limit the display of each column
        Application  Function     User/Group   Access
        *            *            *            *

    Enter '/' to select option or overtype rules

        Application +  Function +  User/Group  Access +
        SCLM           ALL          CNGCTRL     UPDATE
******************************** Bottom of data ********************************
```

Figure 16-17 EAC profile to allow CNGCTRL to create packages in SCLM07.TEST.PACKAGE

These are all the required profiles for the change control team. So in summary, the rules we have added are all shown in Figure 16-18.

```
------------------------------------------------------------------------------
                            Profile Selection                   Row 1 from 8
Command ===>                                                  Scroll ===> PAGE

Profile Prefix (Use + for all Profiles)
        +

Profile Processing
   Enter '/' for selection list

        Profile
        SCLM07.ACCOUNT.FILE
        SCLM07.DEV1.**
        SCLM07.PROD.**
        SCLM07.PROD.ACCOUNT.**
        SCLM07.TEST.**
        SCLM07.TEST.ACCOUNT.**
        SCLM07.TEST.PACKAGE
******************************** Bottom of data ********************************
```

Figure 16-18 List of EAC Rules added

16.3.3 Loading the new rules into memory

Once all the rules have been defined and stored in the rule file, you need to load the rules into memory. EAC should be fully installed and active. This can be verified by selecting option 2 from the EAC main menu. If EAC is correctly configured, you will see some information similar to what is shown in Figure 16-19.

```
------------------------------------------------------------------------------
                              Status Information
Command ===>

    Activity Status  . : ACTIVE
    MVS Subsystem  . . : HSS#
    Rule file in use . : HSS.V1R1.RULEFILE
    Rules loaded . . . : 2007-01-14 at 20:08:10 by DOHERTL
```

Figure 16-19 EAC status information panel

The job to load the rules into memory is provided in the SHSSSAMP library in member HSSRLOAD and contains the following JCL as shown in Example 16-3.

Example 16-3 Sample rule load JCL procedure

```
//*
//RULELOAD PROC HSSLINK=HSS.V1R1.SHSSLINK,
//            RULEFILE=HSS.V1R1.RULEFILE,
//            SSID=HSS#
//*
//RULELOAD EXEC PGM=HSSSSINT,PARM='SSID=&SSID'
//STEPLIB  DD DISP=SHR,DSN=&HSSLINK
//HSSLINK  DD DISP=SHR,DSN=&HSSLINK
//HSSRULES DD DISP=SHR,DSN=&RULEFILE
//SYSPRINT DD SYSOUT=*
//         PEND
//*
//LOAD     EXEC RULELOAD
```

Submit the job and the rules you have just created will be loaded into memory.

Tip: To make it easier to update the rules in memory, assign a PF Key in the EAC application to submit the rules load job. On the EAC main menu, type **KEYS** on the command line, then in one of the unused PF Keys, assign a submit command as shown:

```
Key       Definition                                    Format  Label
 F1 . . .  HELP                                         SHORT   Help
 F2 . . .  SPLIT                                        NO      Split
 F3 . . .  EXIT                                         SHORT   Exit
 F4 . . .  tso sub 'hss.v1r1.instjcl(HSSRLOAD)'         LONG    Ruleload
 F5 . . .
```

Archived

16.3.4 Checking access

With the rules set up as above, users will only be able to perform the actions that they have been authorized to do so by EAC.

Violation when editing outside of SCLM

If a developer accidentally edits a member in the SCLM07.DEV1.COBOL in ISPF edit instead of through SCLM edit, then they will get a RACF violation as shown in Figure 16-20.

```
ICH408I USER(DEVELOP ) GROUP(APC     ) NAME(DEVELOPMENT          )
  SCLM07.DEV1.COBOL CL(DATASET ) VOL(C$US01)
  INSUFFICIENT ACCESS AUTHORITY
  FROM SCLM07.DEV1.** (G)
  ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
IEC150I 913-38,IFG0194E,DEVELOP,ISPF,ISP20532,E200,C$US01,SCLM07.DEV1.COBOL
```

Figure 16-20 RACF violation when trying to edit outside of SCLM

The violation is also logged in EAC. By going to the EAC main menu and selecting option **5** you can see the list of violations since the last IPL. Figure 16-21 shows these violations.

```
------------------------------------------------------------------------------
                            Violation Selection                Row 1 from 5
Command ===>                                                 Scroll ===> PAGE

Enter a prefix value to limit the display of each column
    Date        Time      User      Data Set
    *           *         *         *

Enter '/' to view Violation details

    Date        Time      User      Data Set
_   2007-01-16  20:53:09  DEVELOP   SCLM07.DEV1.COBOL
_   2007-01-16  20:52:12  DEVELOP   SCLM07.DEV1.COBOL
_   2007-01-16  20:48:47  DEVELOP   SCLM07.DEV1.COBOL
_   2007-01-16  20:42:28  DEVELOP   SCLM07.DEV1.COBOL
_   2007-01-14  20:09:06  DEVELOP   SCLM07.DEV1.COBOL
****************************** Bottom of data ********************************
```

Figure 16-21 List of EAC violations

Selecting the last violation in the list, we can see more details on the violation the developer just received, shown in Figure 16-22. The violation detail with a reason code of 08 is telling us that there are no rules matching the application/function that had the high and low program used to access the profile.

```
------------------------------------------------------------------------------
                              Violation Detail
Command ===>

Validation Details
   Profile used  . . : SCLM07.DEV1.**
   Application . . . :
   Function  . . . . :
   User or Group . . :
   Violation reason  : Reason code 08: The Profile has no Applications with
   High-Low programs that matched the execution conditions.


Violation Details
   Data set  . . . . : SCLM07.DEV1.COBOL
   Date  . . . . . . : 2007-01-16          Time  . . . . . . : 20:53:09
   User  . . . . . . : DEVELOP             Group . . . . . . : APC
   Access required . : READ                Access granted  . : NONE

Enter '/' to select option
   /  Display program chain details
```

Figure 16-22 More detail on the violation showing reason code 08

Tip: Position the cursor underneath the “Reason code 08” and press PF1 for help. The EAC extended help listing more information on the violation reason codes.

Selecting the program chain details, we can see the actual high and low program used, as shown in Figure 16-23.

```
------------------------------------------------------------------------------
                            Violation Programs              Row 1 to 2 of 2
Command ===>                                               Scroll ===> PAGE


                        Program   Library Notes
                        ISPTASK   APF, TASKLIB
                        ISREDIT   APF, TASKLIB
******************************** Bottom of data ********************************
```

Figure 16-23 Violation programs panel

This tells us that there are no rules that allow ISPF edit (ISREDIT) to edit the data set protected by the profile SCLM07.DEV1.**.

Tip: If you wish to add an application/function for ISPF edit, you can use the information contained in the Violation Programs panel to define a high/low program pair that can then be used by an EAC profile.

Archived

Violation when performing a function you are not entitled to do

The change control team is only permitted to edit SCLM07.TEST.PACKAGE plus update TEST and PROD through promote. If a member of the change control tries to edit a member at the DEV1, they will get the following two RACF violations, shown in Figure 16-24. This is because there are two profiles that are protected at the development level.

```
ICH408I USER(CNGCTRL ) GROUP(APC     ) NAME(CHANGE CONTROL      )
  SCLM07.DEV1.COBOL CL(DATASET ) VOL(C$US01)
  INSUFFICIENT ACCESS AUTHORITY
  FROM SCLM07.DEV1.** (G)
  ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
IEC150I 913-38,IFG0194E,CNGCTRL,ISPF,SLD9K45Q,E200,C$US01,SCLM07.DEV1.COBOL
ICH408I USER(CNGCTRL ) GROUP(APC     ) NAME(CHANGE CONTROL      )
  SCLM07.ACCOUNT.FILE CL(DATASET ) VOL(C$US01)
  INSUFFICIENT ACCESS AUTHORITY
  ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
IEC161I 040(056,006,IGGOCLFT)-002,CNGCTRL,ISPF    ISPFDEMO,SLD9K6EA,,,
IEC161I SCLM07.ACCOUNT.FILE
```

Figure 16-24 RACF violation when change control tries to edit SCLM07.DEV1.COBOL in SCLM

Again, if we check the EAC violations list we can see that there are two new entries in the list of violations, shown below in Figure 16-25.

```
------------------------------------------------------------------------------
                          Violation Selection                 Row 1 from 8
Command ===>                                                Scroll ===> PAGE

Enter a prefix value to limit the display of each column
    Date        Time      User      Data Set
    *           *         *         *

Enter '/' to view Violation details

    Date        Time      User      Data Set
_   2007-01-16  21:21:51  CNGCTRL   SCLM07.ACCOUNT.FILE
_   2007-01-16  21:21:51  CNGCTRL   SCLM07.DEV1.COBOL
_   2007-01-16  20:56:00  DEVELOP   SCLM07.DEV1.COBOL
_   2007-01-16  20:53:09  DEVELOP   SCLM07.DEV1.COBOL
_   2007-01-16  20:52:12  DEVELOP   SCLM07.DEV1.COBOL
_   2007-01-16  20:48:47  DEVELOP   SCLM07.DEV1.COBOL
_   2007-01-16  20:42:28  DEVELOP   SCLM07.DEV1.COBOL
_   2007-01-14  20:09:06  DEVELOP   SCLM07.DEV1.COBOL
****************************** Bottom of data ********************************
```

Figure 16-25 Dual violations for change control team

Selecting either one of the CNGCTRL violations, we can see the Violation Detail panel shown in Figure 16-26, showing a violation reason code of 07, which tells us that the user is not in the list of access rules defined for the SCLM07.DEV1.** EAC profile.

```
------------------------------------------------------------------------------
                              Violation Detail
Command ===>

Validation Details
   Profile used  . . : SCLM07.DEV1.**
   Application . . . :
   Function  . . . . :
   User or Group . . :
   Violation reason  : Reason code 07: The userid, connected RACF groups or *
   (all users) were not defined within the Profile. No Application matching
   was performed, as the user could not be granted access.

Violation Details
   Data set  . . . . : SCLM07.DEV1.COBOL
   Date  . . . . . . : 2007-01-16          Time  . . . . . . : 21:21:51
   User  . . . . . . : CNGCTRL             Group . . . . . . : APC
   Access required . : READ                Access granted  . : NONE

Enter '/' to select option
   /  Display program chain details
```

Figure 16-26 More detail on the violation showing reason code 07

There will be similar violations if, for example:

- The development team try to promote from TEST to PROD.
- The development team try to edit in TEST.
- The change control team try to edit something other that SCLM07.TEST.PACKAGE.

16.4 Additional considerations

In addition to setting up and using EAC as described above, there are some additional tips and considerations that can help you to use EAC successfully.

16.4.1 Violation reason code 10

Apart from the violations of 07 and 08 described above, the most common EAC violation, especially when initially setting up EAC, is a reason code 10. The most common reason that a user will get this violation is because there is a non-APF authorized load library in the execution path that the user is using.

By execution path we mean LPA, LINKLIST, STEPLIB, and ISPLLIB. Generally load libraries in LPA and LINKLIST are automatically APF authorized. So this normally means that there is a load library in STEPLIB or ISPLLIB that is not APF authorized.

For EAC to allow access, the first thing EAC checks is that the environment is not compromised in some way. So for an uncompromised environment, all load libraries must be APF authorized.

Using DDLIST (TSO ISRDDN) for a logged on userid, shown in Figure 16-27, we can list the ISPLLIB concatenation. In this example the logon procedure does not allocate STEBLIB. So the only place we need to check for non-APF authorized libraries is in the ISPLLIB.

```
                          Current Data Set Allocations            Row 1 of 13
Command ===>                                                    Scroll ===> PAGE

 Volume   Disposition Act DDname   Data Set Name   Actions: B E V M F C I Q
 C$USO1   SHR,KEEP   >    ISPLLIB  SCLM.PROJDEFS.LOAD
 C$USO1   SHR,KEEP   >             BZZ.SBZZLOAD
 C$USO1   SHR,KEEP   >             ISPF.ZOSR8.APARLOAD
 C$USO1   SHR,KEEP   >             ISPFLPA.ZOSR8.MLPALIB
 C$USO1   SHR,KEEP   >             ISPFLPA.ZOSR8.SISPLPA
 C$USO1   SHR,KEEP   >             ISPFLNK.ZOSR8.SISPLOAD
 $$SR78   SHR,KEEP   >             SYS1.DFQLLIB
 $$SR78   SHR,KEEP   >             SYS1.DGTLLIB
 $$SR78   SHR,KEEP   >             SYS1.SICELINK
 $$SR78   SHR,KEEP   >             SYS1.SCBDHENU
 $$SR78   SHR,KEEP   >             EOY.SEOYLOAD
 $D8109   SHR,KEEP   >             DB2.V810.SDSNLOAD
 C$USO1   SHR,KEEP   >             AZZ.V1R1.SAZZLOAD
------------------------- End of Allocation List ----------------------------
```

Figure 16-27 ISPLLIB concatenation

Each and every one of the data sets listed in the ISPLLIB must be APF authorized. This can also be checked through DDLIST (TSO ISRDDN) by typing the APF command. This will list all the APF authorized libraries. Compare the list of datasets in ISPLLIB and STEPLIB with what is in the APF list to ensure that all are present.

16.4.2 Breeze and EAC

In order to use Breeze with EAC, there are some rules that need to be defined, and one of the Breeze jobs needs to be modified for EAC to successfully allow access to Breeze resources.

Setting up a new application/function

There are a number of Breeze modules that need to be defined to EAC as High/Low program combinations for users to have update access on the Breeze database. To add a new application/function, go to option **4** from the EAC main menu, then type an **I** next to one of the existing Application/Functions to insert a new one. Figure 16-28 shows the new Application/Function that we have defined for Breeze. Notice in this case that we have left the function blank, as there are no sub-functions within the Breeze access that need to be split out.

The programs you are adding are:

BZZSAPD1	Required for Breeze to be able to Clear packages that exist at a higher level in the hierarchy
BZZSAPV1	Required so approvers can vote on packages
CIGS0001	One of the main Breeze update modules
CIGS0002	Another of the main Breeze update modules

Archived

```
------------------------------------------------------------------------------
                          Application Maintenance             Row 1 to 4 of 4
Command ===>                                                 Scroll ===> PAGE

  Application  BREEZE
  Function . .
  Data:
  Application to allow Breeze access



    Enter '/' to select option or overtype pairs

       High Program  Low Program
       BZZSAPD1      BZZSAPD1
       BZZSAPV1      BZZSAPV1
       CIGS0001      CIGS0001
       CIGS0002      CIGS0002
****************************** Bottom of data ********************************
```

Figure 16-28 New Application/Function pair for Breeze access

Setting up new profiles for Breeze

A new RACF profile and corresponding EAC profile need to be defined to protect the Breeze database. Create a RACF profile for your Breeze database with a UACC of READ, but no additional access rules. Then in EAC create a new profile that matches with rules that allow update to the required users or groups as shown in Figure 16-29.

```
------------------------------------------------------------------------------
                           Profile Maintenance                  Row 1 from 1
Command ===>                                                 Scroll ===> PAGE

  Profile:  BZZ.SBZZPKG.**
  Data:
  Breeze package database protection



    Enter a prefix value to limit the display of each column
        Application  Function     User/Group   Access
        *            *            *            *

    Enter '/' to select option or overtype rules

        Application +  Function +  User/Group  Access +
        BREEZE                     CNGCTRL     UPDATE
****************************** Bottom of data ********************************
```

Figure 16-29 EAC profile to protect Breeze database

Archived

In our example, Breeze only comes into play in the promotion from TEST to PROD. So the CNGCTRL group is the only group that require access, as they will do the voting as well. Additionally you could separate the Breeze function by have a BREEZE/VOTE application/function pair so that you could just give voting authority to other groups.

Changes to Breeze JCL procedures

In order for Breeze to perform the vote function from the browser, there is a JCL procedure referenced in the Breeze server job that is submitted by the browser started task. This is the BZZSMPKG member shown below in Figure 16-30.

```
//*****************************************************************
//*  SERVER JCL
//*  NOTE: STEPLIB DATASETS MUST BE APF-AUTHORIZED
//*****************************************************************
//CIGLISTN  EXEC  PGM=BZZLISTN,REGION=0M
//STEPLIB   DD    DSN=BZZ.SBZZLOAD,DISP=SHR
//*         DD    DSN=TCPIP.SEZATCP,DISP=SHR
//*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//CIGJAVA   DD    DSN=BZZ.V1R1.SBZZJAVA,DISP=SHR
//CIGLOG    DD    SYSOUT=*
//CIGOUT    DD    SYSOUT=*
//*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
//CIGINRDR DD  SYSOUT=(A,INTRDR),DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)
//CIGSMPKG DD  DSN=BZZ.V1R1.SBZZJCL(BZZSMPKG),DISP=SHR
//CIGSMPRT DD  DSN=BZZ.V1R1.SBZZJCL(BZZSMPRT),DISP=SHR
```

Figure 16-30 Breeze server job referencing the BZZSMPKG member

As shipped, this member is not an ISPF procedure. As such, EAC is going to have problems authorizing the vote program as EAC requires a valid ISPF environment to be in place in order to validate if a user is authorized to use a particular program to access a resource. In order to be able to successfully run the vote job the shipped version of BZZSMPKG needs to be replaced with a version that runs the vote program, BZZSAPV1, within an ISPF environment, for example from IKJEFT01. Example 16-4 shows some sample JCL to do this.

Archived

Example 16-4 Version of BZZSMPKG job when EAC is in use

```
//JC1      JOB (ACCT#),'NAME',CLASS=A,REGION=4096K,
//         MSGCLASS=H,MSGLEVEL=(1,1),
//JC1_USER,PASSWORD
//*  ------------------------------------------------------------------
//STEP010  EXEC PGM=IKJEFT01,REGION=300M,TIME=1439,DYNAMNBR=200
//STEPLIB  DD  DISP=SHR,DSN=BZZ.SBZZLOAD
//ISPPROF  DD  UNIT=VIO,DISP=(NEW,PASS),SPACE=(CYL,(10,10,10)),
//             DCB=(LRECL=80,BLKSIZE=19040,DSORG=PO,RECFM=FB),
//             DSN=                  TEMPORARY TABLE LIBRARY
//*
//ISPMLIB  DD  DSN=BZZ.SBZZMENU,DISP=SHR
//         DD  DSN=ISP.SISPMENU,DISP=SHR
//*
//ISPSLIB  DD  DSN=BZZ.V1R1.SBZZSENU,DISP=SHR
//         DD  DSN=ISP.SISPSENU,DISP=SHR
//*
//ISPPLIB    DD DSN=BZZ.V1R1.SBZZPENU,DISP=SHR
//           DD DSN=ISP.SISPPENU,DISP=SHR
//*
//ISPTLIB  DD  UNIT=VIO,DISP=(NEW,PASS),SPACE=(CYL,(10,10,10)),
//             DCB=(LRECL=80,BLKSIZE=19040,DSORG=PO,RECFM=FB),
//             DSN=                  TEMPORARY TABLE ENURARY
//         DD  DSN=ISP.SISPTENU,DISP=SHR
//*
//ISPTABL    DD UNIT=VIO,DISP=(NEW,PASS),SPACE=(CYL,(10,10,10)),
//           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)
//*
//ISPCTL1  DD DISP=NEW,UNIT=SYSDA,SPACE=(CYL,(1,1)),
//            DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)   TEMPORARY FILE
//*                                                 TAILORING DATASET
//SYSUDUMP DD  SYSOUT=*
//SNAPDUMP DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//CIGLOG   DD  DSN=&&CLOG,
//             DISP=(NEW,PASS),UNIT=SYSALLDA,
//             DCB=(RECFM=FBA,LRECL=133,BLKSIZE=1330),
//             SPACE=(CYL,(1,5))
//CIGRPT   DD   SYSOUT=*
//CIGIN DD   *
//SYSPRINT DD  SYSOUT=*
//ISPLOG   DD  DUMMY
//FLMMSGS  DD  SYSOUT=*
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
ISPSTART PGM(BZZSAPV1)
//*
```

Archived

16.4.3 Using multiple rule files

A useful side effect of EAC having its own set of profiles that provide the specific rules for SCLM access is that a second set of EAC rules can be defined. The EAC rules are stored in a VSAM file. When you, as the EAC administrator, are adding and changing profiles, you tell EAC through the settings panel (EAC option 1) which rule file to store the profiles in. This is shown in Figure 16-31.

```
------------------------------------------------------------------------------
                                  Settings                        Row 1 from 2
Command ===>                                                   Scroll ===> PAGE

Options
    Enter '/' to select option
         Confirm delete of Profile definition
         Confirm delete of Application Function definition
         Confirm autosave of Profile Rules updates
         Confirm autosave of Application updates
      /  Allow Administrator access

Specify Enhanced Access Control for SCLM Load Library
      SHSSLINK data set . . 'HSS.V1R1.SHSSLINK'

Current Rule File . . . . : 'HSS.V1R1.RULEFILE'

Specify Rule File
    Enter '/' to select option

     _   'HSS.V1R1.RULEFILE'
     **End**
```

Figure 16-31 EAC Setting panel

You can define a second set of access rules stored in a separate VSAM rule file that provide restricted access, for example. An example where this scenario could be used is during a weekend implementation or promotion of SCLM code to production. By using a restricted set of profiles, you can stop users from having update access while the implementation is running. Once complete you can then switch the rule files and load the original rules back into memory. shows the settings panel with multiple rule files to select from. To select a different rule file just enter an **S** next to the rule file you wish to activate as shown in Figure 16-32.

Note: You must still run the rule load job to load the rules from this file into memory as mentioned in 16.3.3, “Loading the new rules into memory” on page 405.

Archived

```
------------------------------------------------------------------------------
                                  Settings                        Row 1 from 2
Command ===>                                                   Scroll ===> PAGE

Options
    Enter '/' to select option
         Confirm delete of Profile definition
         Confirm delete of Application Function definition
         Confirm autosave of Profile Rules updates
         Confirm autosave of Application updates
      /  Allow Administrator access

Specify Enhanced Access Control for SCLM Load Library
      SHSSLINK data set . . 'HSS.V1R1.SHSSLINK'

Current Rule File . . . . : 'HSS.V1R1.RULEFILE'

Specify Rule File
    Enter '/' to select option

     _   'HSS.V1R1.RULEFILE'
     s   'HSS.RESTRICT.RULEFILE'
     **End**
```

Figure 16-32 EAC setting panel with multiple rule files

16.4.4 Using the help from the violation panel

The EAC panels have been written following the Common User Access® (CUA®) guidelines, and as such, the help panels drive in a particular way. One of the useful features of the help panels is accessing the extended help for violation reason codes. When you look at a violation through option **5** within EAC, you see the following panel as shown in Figure 16-33.

```
------------------------------------------------------------------------------
                           Violation Selection              Row 29 from 30
Command ===>                                                Scroll ===> PAGE

Enter a prefix value to limit the display of each column
    Date        Time      User      Data Set
    *           *         *         *

Enter '/' to view Violation details

    Date        Time      User      Data Set
_   2007-01-16  20:42:28  DEVELOP   SCLM07.DEV1.COBOL
_   2007-01-14  20:09:06  DEVELOP   SCLM07.DEV1.COBOL
****************************** Bottom of data ********************************
```

Figure 16-33 Violation Selection panel

Archived

If you select the violation you are interested in, you see the violation detail panel as shown in Figure 16-34.

```
------------------------------------------------------------------------------
                              Violation Detail
Command ===>

Validation Details
   Profile used  . . : SCLM07.DEV1.**
   Application . . . :
   Function  . . . . :
   User or Group . . :
   Violation reason  : Reason code 10: The program environment can be
   compromised owing to the presence of an unauthorized asynchronous task.


Violation Details
   Data set  . . . . : SCLM07.DEV1.COBOL
   Date  . . . . . . : 2007-01-16          Time  . . . . . . : 20:42:28
   User  . . . . . . : DEVELOP             Group . . . . . . : APC
   Access required . : READ                Access granted  . : NONE

Enter '/' to select option
   /  Display program chain details
```

Figure 16-34 Violation detail panel

Position your cursor underneath any of the fields on the panel and press PF1 for help and you will receive the help panel related to, not just the Violation detail panel, but the specific field you have requested information on. This is particularly useful for finding additional information on the violation reason code. Positioning the cursor under the violation information and pressing PF1 brings up the Violation Reason field help panel as shown in Figure 16-35.

Archived

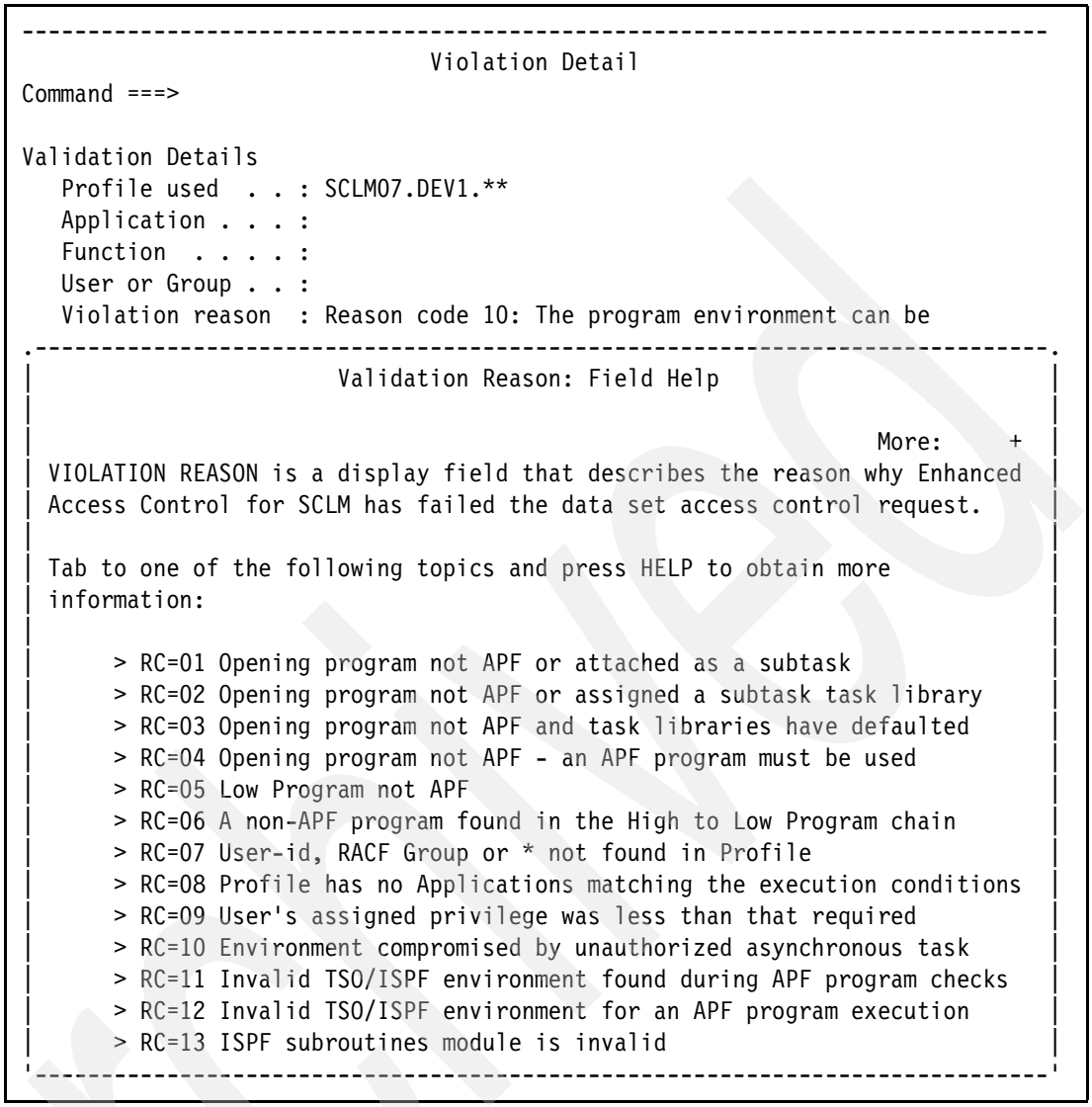


Figure 16-35 Violation reason field help panel

Archived

Then, by again positioning the cursor on the particular violation you are interested in and then pressing PF1 again, you can see the detailed help text for that particular violation.
Figure 16-36 shows the additional information for a violation reason code 10 after PF1 has been pressed.

```
------------------------------------------------------------------------------
                               Violation Detail
Command ===>

Validation Details
   Profile used  . . : SCLM07.DEV1.**
   Application . . . :
   Function  . . . . :
   User or Group . . :
   Violation reason  : Reason code 10: The program environment can be
.------------------------------------------------------------------------------.
|                          Violation Reason Code 10                            |
|                                                                              |
|                                                                More:     +   |
| RC=10 Environment compromised by unauthorized asynchronous task              |
|                                                                              |
| The Validation Routine checks and withholds access if the SCLM program       |
| environment appears compromised.  During this checking, an unauthorized      |
| asynchronous task was found in the program environment.  Therefore the       |
| data set access request was denied.  This type of problem cannot be          |
| resolved by changes to the Profile access rules; nor is it caused by the     |
| installation of the Enhanced Access Control for SCLM software.  The          |
| problem is specific to the program execution conditions at the time of the   |
| data set access request.                                                     |
|                                                                              |
| This problem may occur in an online TSO/ISPF environment, where other       |
| tasks (possibly in split sessions) are present.  One of these tasks has      |
| the capability of interrupting or compromising the SCLM environment.         |
| Closing the split session or logging off then onto TSO may resolve the       |
| problem.  The problem may be intermittent, depending upon the mix of tasks   |
| in the TSO/ISPF environment.  If the problem persists, contact your IBM      |
| representative for help.                                                     |
|                                                                              |
'------------------------------------------------------------------------------'
```

Figure 16-36 Help for Violation reason code 10

Archived

Part 5

SCLM Advanced Edition: Developing with the SCLM Developer Toolkit

SCLM Developer Toolkit V2.1 combines Eclipse-based plug-ins and SCLM functions to provide SCM services for Eclipse-based development environments. It extends the currently available SCLM/WebSphere Developer for System z interface to provide a workstation-based portal to SCLM. Also, through the SCLM Developer View in the SCLM Developer Toolkit, access is provided to SCLM projects and members and SCLM functions and services from the IBM Eclipse environment.

In addition, SCLM Developer Toolkit supports the development of Java and Java 2 Platform Enterprise Edition (J2EE) applications using the existing Eclipse and WebSphere tooling, with storage and management of those components via SCLM on z/OS.

In this part of the book, we describe the following topics:

- Using SCLM Developer Toolkit for traditional mainframe development
- Using SCLM Developer Toolkit for Java/J2EE application development
- Deployment using SCLM Developer Toolkit

© Copyright IBM Corp. 2007. All rights reserved.

Archived

17

Merge Tool

Merge Tool is a product that is sold as part of the SCLM Advanced Edition. A Merge utility belongs in the suite of Software Change Management, but as an add-on tool, Merge Tool is not fully integrated with SCLM. In this section we look at how Merge Tool processing can be performed in tandem with SCLM using the product as it ships. However we also look at a sample utility that utilizes standard ISPF techniques to fully integrate the Merge Tool into SCLM at a member by member level.

Archived

© Copyright IBM Corp. 2007. All rights reserved.

17.1 Merge Tool usage out of the box

As SCLM uses the ISPF editor, it is possible to use the Edit compare command to compare your current source with source code further up the hierarchy. However, the Merge Tool is designed to provide the facility to compare two changed source members with a common base.

For example, suppose that your source is in the PROD level of the hierarchy. There is an emergency change required to keep production up and running, so the support programmer brings a version of the source down from the EMER level of the hierarchy. Meanwhile, during normal development work, another application programmer is working on the same source member in DEV1.

Prior to the new development version being promoted, the developer needs to merge in the changes from the EMER level. The developer could just compare his development source with the EMER, but this does not give the whole picture. The developer needs to see the implications of the changes in relationship to the version of the source in the PROD library.

This section is provided as a brief introduction to Merge Tool so that the sample that is presented in the section can be understood in context. The Merge Tool guide, *IBM Merge Tool for z/OS and z/OS User's Guide,* SC27-1694, contains a good explanation of all the Merge Tool functions along with scenarios that explain the markup tags in the work file very well.

17.1.1 Using Merge Tool to generate work and merge files

Merge Tool is simple to use, but the disadvantage is that, because it is not integrated with SCLM, this means that to compare and merge SCLM members, this has to be done to a library outside of SCLM's control. The Merged source can then be copied into SCLM and saved as normal.

To begin using the Merge Tool, you need to set up your libraries and preferences. Once you start Merge Tool, enter a `/` next to the **`Set Libraries`** option. Press Enter and you will be presented with a panel so you can add the libraries where your source resides, as well as indicating where you want the work and merge files to be created. This is shown in Figure 17-1.

In this example, the File 1 value is the normal development group. The File 2 value is the emergency library. The Base library is a concatenation of where the base source member resides. In our example that is TEST followed by BASE. You can see in the example that the input files are under SCLM control, but the output files are not.

Any of the output files that do not exist will be created by the Merge Tool.

Archive

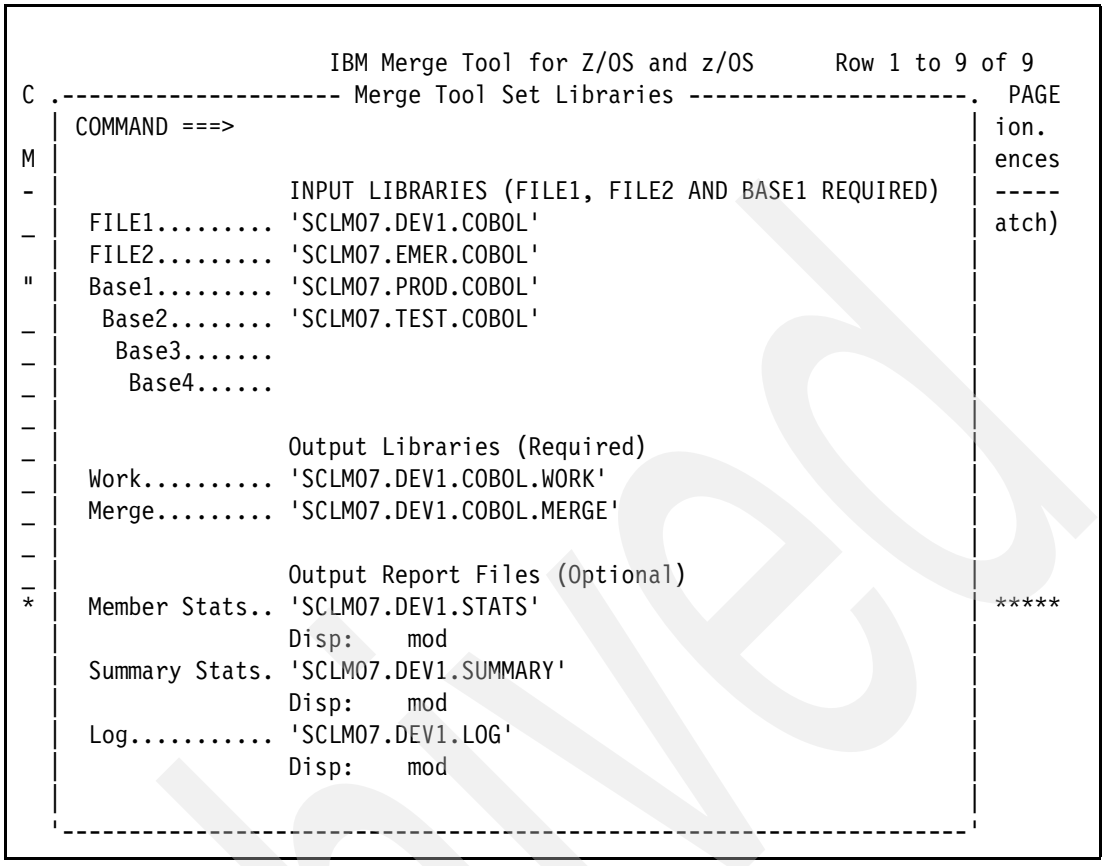


Figure 17-1 Merge Tool "Set Libraries" option

Merge Tool also needs to know the format of the files to enable it to correctly be able to do the compares. This is done through the **Source Type** option. There are number of default types to chose from such as COBOL, PL/I and ASM. Or if you are comparing a type not covered by the defaults then you can use column specifications to define the area of comparison. This is shown in Figure 17-2.

Archived

```
                        IBM Merge Tool for Z/OS and z/OS      Row 1 to 9 of 9
C .-------------------- Specify Source Type --------------------. ll ===> PAGE
  |                                                              | ect option.
M | Command ===>                                                 |  Preferences
- |                                                              | ------------
_ |     Specify Type:   COBOL   (COBOL, ASM, JCL, DATA, PL1)     | nd 2. Batch)
  |                                                              |
" |          COLUMNS FOR COBOL ARE  7:72                         |
_ |                                                              |
_ |       or                                                     |
_ |                                                              |
_ |   Specify compare column start        and end                |
_ |                                                              |
_ |                                                              |
_ |                                                              |
_ |                                                              |
_ '--------------------------------------------------------------'
******************************** Bottom of data ********************************
```

Figure 17-2 Merge Tool "Source Type" option

Next we populate the list of members that we want to work with. By entering *** i**n the Member field, all members that are in File 1 will be listed with information as to whether File 2 and Base members exist for them. This is shown in Figure 17-3.

```
                        IBM Merge Tool for Z/OS and z/OS      Row 1 to 3 of 3
Command ===> ____________________________________________  Scroll ===> PAGE
                                                 Enter "/" to select option.
Member: ________  _ Append to list _ Set Libraries _ Source Type _ Preferences
------------------------------------------------------------ --------------------
__ Group Action          _ View Reports   1 Exec Mode ( 1. Foreground 2. Batch)
    File1                File2    Base     Have  Have   Source
"/" Member    Action     Member   Member   Work  Merge  Type
__  RDBKC01               YES      ------                COBOL__
__  RDBKC02               YES      ------                COBOL__
__  STARTAPP              YES      YES                   COBOL__
******************************** Bottom of data ********************************
```

Figure 17-3 Merge Tool member display

We now want to generate work and merge files. The work file is Merge Tool's own source file that contains the source compares from all three libraries along with some mark-up tags to tell you the results of the compare. It is good practice to generate the work file first to confirm that the merge is going to do what you expect it to do. The generate work (**GW**) and generate merge (**GM**) commands can be performed on individual members or by doing a group action for all members. Next to the STARTAPP program we perform a **GW** command to generate the work file. Once the Merge Tool has finished, the main panel is updated to show that we now have a work file. Next enter the **EW** command next to the STARTAPP program to edit the work file as shown in Figure 17-4. Alternatively you can view the work file with the **BW** command.

```
 IBM Merge Tool for Z/OS and z/OS      Row 1 to 3 of 3
Command ===> ____________________________________________  Scroll ===> PAGE
                                                Enter "/" to select option.
Member: ________  _ Append to list _ Set Libraries _ Source Type _ Preferences
------------------------------------------------------- --------------------
__ Group Action          _ View Reports   1 Exec Mode ( 1. Foreground 2. Batch)
    File1                File2    Base     Have  Have   Source
"/" Member    Action     Member   Member   Work  Merge  Type
__  RDBKC01              ------   ------                COBOL__
__  RDBKC02              ------   ------                COBOL__
EW  STARTAPP             YES      YES       YES         COBOL__
****************************** Bottom of data ********************************
```

Figure 17-4 Editing the generated work file

You are put into ISPF edit on the workfile where you can see the results of the merge. The results from this action are shown in Figure 17-5 on page 426. The work file contains information on which files were used to generate the work file. in columns 1 through 7 there are tags that define what has occurred during the merge. In the example below we see the lines that have been added or removed by file 2, the SCLM07.EMER.COBOL, denoted by the +2 and -2 tags. Similarly the lines that have been modified in file 1, SCLM07.DEV1.COBOL are marked with the +1 and -1 tags. For example there is a change to line 003400 where **Move "Q" to Input-name** has been changed to **Move "Q" to Input-name** . Merge Tool shows this as removing a line and adding a line.

Once you are happy with the changes in the work file, you press PF3 to save the member and then issue the **GM** (Generate Merge) command to create the merge file based on the contents of the work file. Based on our settings above, the Merge file is stored outside of SCLM control in SCLM07.DEV1.COBOL.MERGE.

17.1.2 Adding merge file back into SCLM

The reason it is best practice to generate the merge files outside of SCLM is because having Merge Tool generate them directly within SCLM invalidates the accounting information that SCLM stores in its VSAM accounting files. Of course it would be possible using SCLM's Migrate function to then migrate the newly created source and therefore update the account information.

However by creating the Merge File directly, you could also copy the file over an existing file without meaning to, or if you have Enhanced Access Control (EAC) in place, Merge Tool itself may not have the authority to update the development source library where you are going to be working on the source.

The best practice, once you have a merge file created outside of SCLM, is to go into edit on the member in the development group, as you normally would in SCLM. Then use the edit COPY command to copy in the generated merge file into the member you are working on. This way you keep all processing under the control of SCLM, and have a means of cancelling out of ISPF if you think the merge file is not what you were expecting.

So in our example we go into edit on SCLM07.DEV1.COBOL(STARTAPP) as shown in Figure 17-5. SCLM starts an edit session. We delete all the lines in the member using the **d999999** line command, then issue the **COPY 'SCLM07.DEV1.COBOL.MERGE(STARTAPP)'** command on the command line to copy in the member.

```
------------------------------------------------------------------------------
EDIT       SCLM07.DEV1.COBOL.WORK(STARTAPP) - 01.00        Columns 00001 00072
Command ===>                                                 Scroll ===> PAGE
****** ***************************** Top of Data ******************************
000001 .----- **** IBM MERGE TOOL FOR z/OS AND z/OS         WORKFILE ****
000002 .----- BASELINE: SCLM07.PROD.COBOL(STARTAPP)
000003 .----- FILE 1  : SCLM07.DEV1.COBOL(STARTAPP)
000004 .----- FILE 2  : SCLM07.EMER.COBOL(STARTAPP)
000005 .----- ....+....1....+....2....+....3....+....4....+....5....+....6....+.
000006       000100* -------------------------------------------------
000007       000200*    IDE Sample Program
000008       000300* -------------------------------------------------
000009       000400 Identification Division.
000010       000500 Program-ID.  StartApp.
000011       000600
000012       000700 Data Division.
000013       000800 Working-Storage Section.
000014       000900
000015       001000 COPY STARTCPY.
000016       002000
000017       002100 Procedure Division.
000018       002200
000019       002300     Initialize Program-pass-fields
000020       002400                Program-other-fields
000021       002500                Program-flags.
000022       002600
000023 . +2  002700*    Move "ITSO Redbooks team" to Input-name
000024 . -2  002700     Move "ITSO Redbooks team" to Input-name
000025       002800     Perform until Loop-done
000026 . +2  002900         Display " "
000027 . +2  003000         Display "Enter a name or Q to quit:"
000028 . +2  003100         Move Spaces to Input-name
000029 . +2  003200         Accept Input-name
000030 . -2  002900*        Display " "
000031 . -2  003000*        Display "Enter a name or Q to quit:"
000032 . -2  003100*        Move Spaces to Input-name
000033 . -2  003200*        Accept Input-name
000034       003300         IF Input-name = Spaces
000035 . +1  003400           Move "E" to Input-name
000036 . -1  003400           Move "Q" to Input-name
000037       003500         End-IF
000038       003600
```

Figure 17-5 Example of Merge Tool workfile

17.2 Integrating the Merge Tool with SCLM

Out-of-the-box the Merge Tool requires some manual SCLM processes, as described above, to bring the merge members into SCLM. But using some simple ISPF edit macro techniques along with SCLM services and REXX to call the Merge Tool functions, it is possible to call merge tool functions directly from a member in SCLM, thus integrating the Merge Tool into the SCLM editor, much the same way that the edit compare command can be called.

Merge Tool does not have a documented API, however the merge process can be executed in a batch mode by allocating the required files and passing the list of members to the process in a SYSIN type file. This method of calling the Merge Tool can be invoked in a foreground mode by using REXX to allocate the files, build the SYSIN file, and then call the merge process.

17.2.1 Integrated Merge Tool in action

Instead of invoking the Merge Tool to do the compares and create the work and merge files, we go into normal SCLM edit on the member that we know needs to have other changes merged into it. SCLM brings up the editor, and on the command line we enter the MERGE command as shown in Figure 17-6.

```
 File  Edit  Edit_Settings  Menu  Build  SCLM  Utilities  Test  Help
-------------------------------------------------------------------------------
EDIT       SCLM07.DEV1.COBOL(STARTAPP) - 01.03            Columns 00001 00072
Command ===> merge                                          Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 000100* ------------------------------------------------
000200 000200*    IDE Sample Program
000300 000300* ------------------------------------------------
000400 000400 Identification Division.
000500 000500 Program-ID.  StartApp.
000600 000600
000700 000700 Data Division.
000800 000800 Working-Storage Section.
000900 000900
001000 001000 COPY STARTCPY.
002000 002000
002100 002100 Procedure Division.
```

Figure 17-6 Invoking the sample merge command

Immediately a pop-up panel will appear requesting some information that Merge Tool requires. The supplied sample will determine much of this information based on the information about the member. The following fields are required, as shown in Figure 17-7.

Base file

This is the base file for the comparison. The sample looks up the hierarchy from the development member you are working on and places the first found member into this field. You can change this value if you want a different file to be used as the base file.

File 1

This is defaulted to the current development file; that is, the file you are currently editing.

File 2

Initially this field is blank, as SCLM does not know which other file you wish to use in the compare. Once entered, the value is stored in your ISPF profile and is reused whenever you use this option. In the example below this has been set to SCLM07.EMER.COBOL, which is the emergency level of our hierarchy. This would be a common scenario.

Archived

Options - Edit work file

This option places you into edit on the work file so you can verify that the compare is giving the results you expect. This also allows you to make any changes you deem necessary.

Options - Merge into SCLM

This option will take the generated merge file and replace the contents of the member you are editing in SCLM with the merge file. If you have selected not to edit the work file, the merge file will be generated straight away. You still have the option to not replace the member contents or even cancel out of the edit if you change your mind.

Source type

This is the Merge Tool specified option that allows Merge Tool to correctly run the compare. The sample exec determines the SCLM language of the member, then based on that sets the source type.

Start Col/End Col

The source type can be overridden by specifying a start column and an end column.

```
------------------------------------------------------------------------------
EDIT       SCLM07.DEV1.COBOL(STARTAPP) - 01.03            Columns 00001 00072
Command ===> merge                                          Scroll ===> CSR
****** **************************** Top of Data ****************************
000100 000100 .-------------------- SCLM Merge Options --------------------.
000200 000200 |                                                             |
000300 000300 | Merge Tool Data sets                                        |
000400 000400 |    Base (Next location where member found in Hierarchy)     |
000500 000500 |        ===> SCLM07.PROD.COBOL                               |
000600 000600 |                                                             |
000700 000700 |    File1 (Current Development File)                         |
000800 000800 |        ===> SCLM07.DEV1.COBOL                               |
000900 000900 |                                                             |
001000 001000 |    File2 (Enter 2nd Merge File)                             |
002000 002000 |        ===> SCLM07.EMER.COBOL                               |
002100 002100 |                                                             |
002200 002200 | Options (enter '/' to select)  Source type   : COBOL        |
002300 002300 |  / Edit Work File               or Start Col :              |
002400 002400 |  / Merge into SCLM               and End Col :              |
002500 002500 |                                                             |
002600 002600 | Press Enter to process or PF3 to cancel                     |
002700 002700 '-------------------------------------------------------------'
002800 002800     Perform until Loop-done
```

Figure 17-7 Reviewing and specifying the Merge Toll options

Pressing Enter will invoke the Merge Tool. With the setting we have specified above, the work file will be generated and the sample exec will place us in edit on the generated work file, which is stored in a temporary data set, as shown in Figure 17-8.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
EDIT       DOHERTL.CIGWORK(STARTAPP) - 01.00              Columns 00001 00072
Command ===>                                                 Scroll ===> CSR
****** ***************************** Top of Data ******************************
000001 .---- **** IBM MERGE TOOL FOR z/OS AND z/OS        WORKFILE ****
000002 .---- BASELINE: SCLM07.PROD.COBOL(STARTAPP)
000003 .---- FILE 1  : SCLM07.DEV1.COBOL(STARTAPP)
000004 .---- FILE 2  : SCLM07.EMER.COBOL(STARTAPP)
000005 .---- ....+....1....+....2....+....3....+....4....+....5....+....6....+.
000006       000100* ------------------------------------------------------
000007       000200*    IDE Sample Program
000008       000300* ------------------------------------------------------
000009       000400 Identification Division.
000010       000500 Program-ID.  StartApp.
```

Figure 17-8 Temporary data set containing the work file

You can make any required changes. Then press PF3 to end and exit out of the temporary member. At this point the process will display a pop-up confirmation panel if you have chosen to perform the merge and store the file back in SCLM, as shown in Figure 17-9.

```
--------------------------------------------------------------------------------
EDIT       SCLM07.DEV1.COBOL(STARTAPP) - 01.03              Columns 00001 00072
Command ===> merge                                           Scroll ===> CSR
****** ***************************** Top of Data ******************************
000100 000100 .------------- SCLM Merge Confirmation -------------.
000200 000200 |                                                   |
000300 000300 | Do you want to generate the merge file and        |
000400 000400 | replace the SCLM member (Y/N) ===> N              |
000500 000500 |                                                   |
000600 000600 |                                                   |
000700 000700 '---------------------------------------------------'
000800 000800 Working-Storage Section.
000900 000900
001000 001000 COPY STARTCPY.
002000 002000
002100 002100 Procedure Division.
```

Figure 17-9 Pop-up panel to confirm the replacement of the SCLM member

Selecting **Y** to replace the member in SCLM will do just that. The current contents are deleted and the merge file you have just generated is placed in the member you are working on. At this time you still have the option to cancel out of the edit session and not replace the member if the merge file looks incorrect.

The advantage of using this sample method over the standard Merge Tool is that all the actions are done within the member in SCLM, so therefore under the control of SCLM. Also, the file settings and options settings are mostly derived for you. Obviously this is an interactive member by member option, so the multiple member processing and batch facility in Merge Tool do not come into play.

17.2.2 Explanation of the sample code to perform the function

The sample code to perform this function comes in the form of a single REXX procedure, two ISPF panels, and an ISPF messages member. These will need to be added to the ISPF libraries allocated in your logon procedure that is used for your SCLM development. The parts are as follows:

- REXX member - MERGE
- ISPF Panel member - SCLMMRG1
- ISPF Panel member - SCLMMRG2
- ISPF Message Member - SCLM00

However, here is a brief explanation of the processing involved in the REXX procedure:

1. Get the dataset name and member using the Edit macro statement.
2. Get various SCLM variables from the profile pool.
3. Perform an SCLM ACCTINFO service call for the current member to derive the language and “real” data set name. SCLM has the ability to have longer than 3 level dataset names even though on the service the data set is made up of project.group.type.
4. Determine the next group in the hierarchy using the SCLM NXTGRP service call.
5. Perform an SCLM ACCTINFO service call to search for the member in the hierarchy starting at the next group above the current development group.
6. Set the Source type based on the members SCLM language
7. Display the pop-up panel to allow the user to change any required options.
8. Using the variables from the panel allocate the required data sets that Merge Tool requires. This requires determining the record length of the current source file and then allocating the work file to have an LRECL of 7 greater than this.
9. Set up the CIGIN file to contain the member being processed and the source type option
10. Call the AZZMERGE exec to create either the work file or the work and merge file, depending on what options the user has selected.
11. If the option to edit the workfile was selected, then call the ISPF edit service for the temporary file. If the option to copy the merge file to SCLM was selected as well then call the AZZMERGE exec again to generate the merge file from the work file that was just edited.
12. If the option to copy the merge file to SCLM, then display the confirmation panel.
13. If the action is confirmed, use edit macro commands to first delete the contents of the SCLM member, and then copy in the merge file member.

Archived

18

Open Systems basics

The term “Open Systems” refers to an effort to provide standardization and interoperability between systems, particularly UNIX® systems. As traditional mainframe processes move forward into the open systems world, many mainframe systems programmers and application programmers are unfamiliar with many of the aspects of the world of distributed programming, and the tools used in the development of distributed applications.

For systems programmers there is a need to understand the basics of the z/OS UNIX File System, as they have to install new products that are progressively increasing their utilization of the z/OS UNIX file system.

For application developers who are used to traditional 3270 systems and associated tools such as ISPF, it is sometimes a difficult transition to use more Integrated Development Environments or IDEs.

In this chapter we introduce some of the basics of the open systems world such as:

- What is the z/OS UNIX file system?
- What are Java and J2EE?
- What are Eclipse, Rational® Application Developer (RAD), and WebSphere Developer for System z (WD/z)?

Next we explain some of the basics behind these terms so that when they are used throughout the rest of this book, you will have a better understanding of the concepts.

© Copyright IBM Corp. 2007. All rights reserved.

18.1 What is the z/OS UNIX file system?

The z/OS UNIX file system is the IBM UNIX offering on z/OS that was created to provide a new platform on System z to compete with other UNIX platforms. The z/OS UNIX file system is much the same as any other UNIX system in that it has a directory and file structure the same as other UNIX systems (which are in fact similar to PC systems). It has TCP/IP to connect to other systems. It has a set of system calls that perform functions that are similar to the functions other UNIX systems provide. It allows users to telnet or login to a shell (the UNIX command-line interface) to interact with the system. It has certain utilities that the UNIX user expects to find in the work environment.

18.1.1 The IBM implementation of UNIX on z/OS

IBM did not port someone else's UNIX to z/OS. z/OS UNIX is not an emulation, a subsystem, or an LPAR; it is not sitting on top of z/OS (as in a layering effect that would suggest bad performance). IBM looked at the spec1170 specifications that say, “these are what the UNIX interfaces are” and implemented those specifications directly into the z/OS system as OpenEdition services, which now are the z/OS UNIX services. The OpenEdition z/OS Shell and Utilities, with many of the Korn shell features, were first to take advantage of the new z/OS UNIX file system in z/OS.

To manage z/OS UNIX, IBM wanted to use z/OS system programming and operational skills. They wanted those who understand and manage S/390® availability, reliability, and security to continue to manage this new system in ways familiar to them. Therefore, there are traditional z/OS methods to manage the hierarchical file system, and SMP/E management of product code and service updates. However, UNIX programmers who expect UNIX interfaces will find those interfaces, unaware that, under the covers, traditional z/OS things happen. For example, z/OS UNIX services operate in z/OS virtual storage areas called address spaces. Both the z/OS system programmer and the UNIX application developer can be comfortable in this new environment.

There are many services in the z/OS system, such as z/OS services and z/OS UNIX services. All of these services run on z/OS. Existing z/OS applications continue to run, unaffected by the existence of UNIX services and applications that use the z/OS UNIX services. All the applications can benefit from the reliability, availability, and security of the S/390 platform.

A common misconception is that application programmers have a choice of either running UNIX programs that use UNIX services and file systems or running z/OS programs that use z/OS services and z/OS data sets. You do not have to choose between z/OS UNIX and z/OS. For example, if you are developing a UNIX application that needs to call DB2, you do not have to pass a request to a z/OS program that then extracts data from DB2 and passes data back to the UNIX program. You can call DB2 directly from UNIX. And, if you telnet into the shell, you have a standard UNIX interactive command-line interface like the one you are familiar with. There is no wall between UNIX and z/OS and there are no “sides”; you do not run on the “UNIX side” or the “z/OS side.” z/OS is a powerful blend of z/OS UNIX file system and z/OS.

18.1.2 Basic UNIX concepts

z/OS system programmers now installing, maintaining, and debugging z/OS UNIX are faced with new responsibilities that require learning a new vocabulary and new concepts. Perhaps they now must interact with UNIX or TCP/IP computer personnel from whom they have been isolated previously. Differences between the familiar z/OS world and the new UNIX world may seem cultural and philosophical, as well as technical. This section tries to bridge the gap.

Kernel and the shell

The UNIX operating system has two distinct components, the kernel and the shell:

- The kernel is part of the BCP element of z/OS; it sends instructions to the processor, schedules work, manages I/O, and tracks processes, open files, and shared memory, among other things. Other parts of the operating system or applications request the kernel's services using assembler callable services (called syscalls). No work gets done in z/OS UNIX without involving the kernel.
- The shell, the interactive interface with the kernel, has a programming language of its own. The shell is a command interpreter. The shell's work consists of programs and shell commands and utilities run by shell users. You can think of the shell running with the kernel as TSO/E running with z/OS (after all, TSO/E is the original shell of z/OS.)

A shell script consists of shell commands packaged in one file; a shell script is similar to an z/OS command list or a REXX exec.

Accessing the shell

There are several ways for you to access the shell. The most common are:

- The OMVS command, issued by a logged-on TSO/E user ID from a 3270 terminal or a workstation running a 3270 emulator, provides those familiar with TSO/E a familiar 3270-type interface.
- Use of the TSO ISHELL command. This is more of an ISPF menu style system to list directories and work with UNIX files.
- From z/OS 1.8 onwards there is more UNIX support directly in ISPF. ISPF option 3.17 provides a similar interface to ISHELL but uses more ISPF style constructs.
- The telnet or rlogin commands, issued (without a login proc) from a workstation running TCP/IP.

  When you use telnet or rlogin to access z/OS UNIX, you have a standard UNIX interface, not a 3270-type interface. You work in character-at-a-time mode (each character processed as you type it). You can use the vi editor.

The shell has some extended functions that have no z/OS equivalents; for example:

- Pipes (redirecting output of a program to the input of another)
- Redirection (directing a data flow)

Daemons

A daemon is like an z/OS started task — it is a long-running task that runs in the background and is not associated with any particular user. It starts work (or performs the work itself) on behalf of a user request, such as an rlogin request on an IP port, or a user shell script run at a scheduled time each day. Daemons usually run authorized and can issue authorized functions, such as changing the identity of the user associated with a process. You can think of the VTAM® started task as a daemon.

Hierarchical File System (HFS)

If you have used UNIX before, you are probably familiar with file systems that are hierarchical in nature. In z/OS, system programmers need to understand this file system; they have new terms to learn and new procedures to follow, including:

- Allocating and mounting HFS data sets
- Applying service to the programs and data in the HFS data sets.

In z/OS UNIX, all data and programs are in a hierarchical file system (HFS). You can think of this file system as an upside down tree with a root at the top and many leaves, that is files, at

the bottom. This file system is similar to the file structures in the Windows platform. It is a collection of files and directories.

z/OS programs and data are delivered to you in z/OS data sets and in this new file structure. After the installation is complete, more directories and files will be created for use by application developers.

z/OS has implemented the file systems in the following way. It uses z/OS data sets of a type called “HFS”, a type created for z/OS UNIX. System programmers create and maintain the HFS data sets. These data sets can only be used through z/OS UNIX. This use is no different from how a z/OS program would use a VSAM data set.

The way to make an HFS data set accessible is by mounting it. After you finish with a data set, you can unmount it. The z/OS UNIX mount is not the same as the traditional volume mount; internally, z/OS allocates and opens a data set. For the unmount, z/OS closes the data set and deallocates it.

18.1.3 z/OS UNIX security

In z/OS, z/OS security extends to include z/OS UNIX. To protect the system from programs and programs from other programs, every resource runs in an address space and every task in that address space runs under a security environment managed by RACF, or a comparable security program.

All access control decisions for z/OS UNIX are made by RACF, unlike other UNIX systems. In z/OS UNIX, RACF knows users by a numeric ID, called a UID. Additionally, groups the users belong to are known by group IDs (GIDs). For example, if everyone within a department needs to use a certain set of common files, directories, or devices, that department would be a group and have a GID. A user's UID and GID are stored in RACF's security data base.

When a user wants to access a file, RACF matches the requester's UID and GID against security information associated with each file, namely:

- The file's owner, represented by the owner's UID.
- Group owner, represented by the owning group's GID.
- Permission bits, which describe the read, write, and execute ability for owner, group, and “others” (all users). The permission bit is known by a three-digit number. For example, permission bit 755 is a common one, as shown in Figure 18-1, looks like this, where r stands for read, w stands for write, and x stands for execute.

```
r w x  r w x  r w x
1 1 1  1 0 1  1 0 1
   7      5      5
   3      3      3
   3      3      all users permission
   3      group permission
   owner permission
```

Figure 18-1 Permission bits in UNIX

- By matching the user's UID and GID against this security information, RACF determines who should be allowed to read, write, and execute the file. In this case the permission bit 755 means that the owner can read the file, write to the file, and execute the file; members of the owning group can read and execute the file, as can all users. The owner can write to the file; no one else can.

Superuser authority

Just like all UNIX systems, the installation defines certain system administrators as superusers, having UID(0), who can change the contents of any file, manage processes, and perform other administrative activities. This can be restricted based on RACF profiles. When not doing activities that require superuser authority, system administrators can change to user authority, which permits access to his or her own files and other files, according to the permission bits.

Do not confuse superuser authority with z/OS supervisor state. Being a superuser does not increase the ability to do things in z/OS. Remember the point about “no wall between z/OS and UNIX”? Well, where security is concerned, the wall is actually there.

18.1.4 Utilizing z/OS UNIX

IBM offers familiar ways to use z/OS UNIX, whether your background has been with UNIX or with z/OS. Workstation users can copy data from z/OS data sets into the file system for use at the host or, through z/OS Network File System, at the workstation. They can also upload ASCII-encoded files to the host, where they can be translated into EBCDIC encoding for the host. The same application program can access both traditional z/OS data sets and the files in the HFS. The program can invoke callable services from Assembler applications and subroutines, in addition to the C/C++ language applications. And, here are more ways:

- Interactive users can choose between the familiar interface of the UNIX shell or command interface and the traditional TSO/E panel or command interface. To use the HFS, you can use TSO/E commands, shell commands, or the panel interface of ISPF.
- To edit HFS files, you can use the ISPF full-panel editor or a UNIX editor, such as vi or ed, available in the shell.
- You can write z/OS job control language (JCL) that includes UNIX shell commands.
- Using z/OS UNIX extensions to REXX, you can run REXX programs from TSO/E, batch, the shell, or a C/C++ program.
- You can run a non-interactive shell command or a shell script from the TSO/E READY prompt and display the output at your terminal; you can run an interactive shell command through the OMVS command.
- You can use the BPXBATCH interface (with familiar JCL statements) to run the shell and its utilities in the background.

Workstation users can be connected to TSO/E and the shell through 3270 emulation or through TCP/IP. An installation can customize logging-on so that a user who prefers to work with a UNIX-like interface is put directly into the familiar UNIX shell. From there, the user can choose to switch to the TSO/E environment (through ISPF panels or TSO/E commands). An interactive user can log on through the TSO/E interface and switch to the shell when desired.

18.1.5 Guidelines for installing products into the z/OS UNIX file system

There are a number of packaging guidelines that IBM products follow as they install code into the z/OS UNIX file system. There are certain guidelines for where parts of a product are installed to protect the delivered code but also to ensure any local tailoring is not compromised by upgrades to the product. During the installation of SCLM Developer Toolkit and SCLM Administrator Toolkit you will see certain directories being used for certain things. So let us look at some of the directory locations in the z/OS UNIX file system to understand what they are used for.

/usr/lpp

This directory is where product code is normally installed. It is also possible that if you have a service or maintenance procedure, then you may precede the /usr/lpp directory with a prefix director such as /apc/SCLMDT/usr/lpp/SCLMDT/. For example, when you set up your SMP/E job, it will assume that you are installing into /usr/lpp and will create the products own directory structure after that, /SCLMDT/bin/. But during SMP/E setup you can tell SMP/E that you are using a path prefix.

Normally you will give a product it's own HFS data set to contain the product code and you will mount this file at a certain directory location. Let us assume we have a path prefix of /apc/SCLMDT/usr/lpp rather than just /usr/lpp. When doing the install we make the root directory Read/Write (RDWR) and create the SCLMDT/usr/lpp directory. We then mount a HFS file at this directory location and again make it Read/Write, for the duration of the install. In our SMP/E job we tell SMP/E our path prefix is /apc/SCLMDT/usr/lpp. The SMP/E jobs will then create the required directories under here, so /usr/lpp/SCLMDT/bin. The full path will be /apc/SCLMDT/usr/lpp/SCLMDT/bin. The SMP/E APPLY job will then copy the product files to this directory location.

Once the install is complete you unmount and mount the directory as Read-only (READ). This is part of the IBM scheme to keep local customer data separate from shipped code. On z/OS, /usr/lpp/ is supposed to contain only code shipped by IBM (or whoever). This way, after install is complete, many systems within a sysplex can share the same physical copy of /usr/lpp.

Each system in a sysplex will have its own copy of its system-specific code (like **/etc** and **/var**, and others).

This is different than on some other platforms, which allow local customer data to also live within /usr/lpp. If you are porting a product from a platform which expects to write a local data into /usr/lpp, then consider using a symlink to point from **/usr/lpp/productname/config.file** to **/etc/productname/config.file.**

This can usually be done without any change to the ported code.

To see what HFS files are mounted at what directory locations, issue the following operator command:

```
D OMVS,F
```

In the system log, you will see a list of files returned with their mount points and whether they are read or write as shown in Figure 18-2.

Archive

```
TYPENAME   DEVICE ----------STATUS----------- MODE  MOUNTED    LATCHES
HFS            18 ACTIVE                      RDWR  12/08/2006  L=28
  NAME=AUZ.V310.OMVS.HFS                            15.39.13    Q=0
  PATH=/apc/sclmat/V310
HFS            16 ACTIVE                      READ  12/08/2006  L=27
  NAME=WD4Z.V700.OMVS.HFS                           15.39.13    Q=0
  PATH=/apc/wd4z700/usr/lpp/wd4z
HFS            15 ACTIVE                      READ  12/08/2006  L=26
  NAME=IXM.V190.OMVS.HFS                            15.39.13    Q=0
  PATH=/apc/txml190
HFS            13 ACTIVE                      READ  12/08/2006  L=24
  NAME=WEBSPHER.V601.SBBOHFS                        15.39.13    Q=0
  PATH=/apc/WebSphere.V601/usr/lpp/zWebSphere/V6R0
HFS            12 ACTIVE                      READ  12/08/2006  L=23
  NAME=JAVAOMVS.V142.UK07860.HFS                    15.39.13    Q=0
  PATH=/apc/java142-UK07860/usr/lpp/java
```

Figure 18-2 Results from D OMVS,F command as shown in SYSLOG

/etc

/etc contains local data that must be preserved from one release to the next. This is configuration data, setup info, the your local data. To avoid re-configuring everything when IBM or other customers ship new code, this data must be kept separate from the code that ships. So it is separate from the shipped code in /usr/lpp or pathprefix/usr/lpp.

For example, SCLM Developer Toolkit after SMP/E installation provides a job to set up the /etc directory. A directory called /SCLMDT/CONFIG is created under /etc and contains such things as the http configuration files. Generally these file will not change once set up is complete, and must be preserved between IPLs.

/var

/var contains data that is created during the normal running of a product, that must be preserved from one IPL to the next. One example of this type data is lists of files queued to a printer.

Ideally, the product will create and populate its /var directories on the fly. This makes packaging, installing, and documenting the product easier for everyone. Actions may still be needed to migrate /var from one release to another, or between systems in a sysplex.

Again using SCLM Developer Toolkit as an example, after SMP/E installation a job is run to set up the /SCLMDT/WORKAREA and /SCLMDT/LOGS directories under the /var directory. The data in these directories is temporary and is only valid for the action that is taking place. So for SCLMDT this data could actually be lost between IPLs, but that may not always be the same for all products.

18.2 What is Java? What is J2EE?

Java/J2EE is just another form of application development. There are many resources on the Web that help to explain all the terminology on the Java/J2EE world, but this section will give a brief outline of some of the terms you will come across in the last two parts of this book.

Java is a programming language, however, it is unique in that code written in Java will work on any platform that has a Java Virtual Machine. When you compile a Java program, it does not compile into native code, but instead into Java bytecode, that is interpreted at runtime by the JVM™ (Java Virtual Machine). You can think of the JVM as a platform on its own, bytecode as its native instruction set. Java has a compiler like any other high level language, called javac. It is invoked, passing it a number of parameters to assist the compile. Java source goes in, class files come out. Class files are like the OBJ or load modules to COBOL source.

J2EE is a specification and rich set of Java APIs for developing modern Enterprise class applications. The purpose of J2EE is to factor out the common problems of developing things like Web based services, such as load balancing, persistence, etc. J2EE allows programmers to focus on the business logic, data model, and user interface, and forget about the complexities of managing the interactions of the three and worrying about the server and its performance. While J2EE is a kind of standard, multiple vendors provide different implementations of application servers that support J2EE applications, such as WebSphere Application Server (WAS), TomKat, JBoss.

Let us look at some of the other terms that are associated with Java/J2EE development:

J2EE Stands for “Java 2 Platform, Enterprise Edition” and it defines the standard for developing component-based multi-tier enterprise applications.

JAR Java Archive. A file format for storing information for or about Java programs.

WAR Web Archive. Much the same as a jar, except it relates to Web type applications.

EJB™ Stands for “Enterprise Java Beans”. Java Beans are reusable objects, like subroutines. EJBs take this a step further and are designed to be platform independent. But these are basically a type of JAR file.

EAR Enterprise Archive. This is just a deployment container. It will contain jar’s, war’s and additional files that are going to be deployed to a server.

To explain these in more depth, an important aspect of project development, particularly in J2EE projects, is that a number of different forms of application executables can be created. Java project executables are often packaged as JAR, WAR, or EAR files.

JAR files typically relate to standard Java applications or Enterprise Java Beans (EJB).

WAR files are created for Web resources. Typically these are composed of Java servlets, JSPs, and HTML files. WAR applications are often the front end of Web-based applications. These applications may reference other JARs such as EJBs for specific services. Each WAR file contains a web.xml file. This describes the composition of the WAR application in terms of Java, HTML, and the library services that it uses.

EAR files represent enterprise applications. These applications are composed of JAR and WAR files. In J2EE language, the creation of the EAR file is the assembly of its constituent JAR, and WAR files. This method of assembly allows EAR applications to be created which are in effect made up of specific components (JAR/WAR). The actual composition of the application is described in the application.xml file. The EAR file itself is not a standalone executable object. The EAR file must be installed in a J2EE compatible application server such as WebSphere Application Server (WAS). The installation of the EAR file is referred to as deployment. A deployed EAR application can be accessed via the WAS environment. Deployment is a separate process from that of the build. Deployment involves the physical installation of the EAR application.

When developing J2EE applications it is therefore possible that it will involve the development of a number of separate components such as WAR and JAR files. These files are then assembled together into an EAR file. The EAR file is then ready for deployment into a J2EE container (for example, WAS) for operation.

18.3 What are Eclipse, Rational Application Developer, and WebSphere Developer for System z?

SCLM Developer Toolkit and SCLM Administrator Toolkit run as plug-ins in the Eclipse framework. Eclipse is an Interactive Development Environment that provides tools to enable you to develop Java applications and more. It is also Open Source. IBM has its own version of Eclipse which IBM products use to ship with their products. For example, you can install SCLM Developer Toolkit as a stand-alone product and it will plug-in to the Eclipse that ships with it.

Rational Application Developer (RAD) is an IBM product that extends Eclipse with visual construction development to help Java developers rapidly design, develop, assemble, test, profile and deploy high quality Java/J2EE, Portal, Web, Web services, and SOA applications.

WebSphere Developer for System z (WD/z) is a product that further extends the functionality of Eclipse and RAD to add many features such as:

- Traditional development in an IDE of COBOL and PL/I
- High-level Enterprise Generation Language,
- The z/IDE, a portal to z/OS to allow you to work with your z/OS data sets and members
- Remote debugging capabilities for z/OS applications so that they can utilize the power of the IDE debugger

Much more that would be out of the scope of this book.

What this section will do, is give you an introduction to the IDE, be it Eclipse, RAD or WD/z and some of the terminology used when working in the Eclipse IDE. More information can be found in the on-line help in any of these products.

18.3.1 Workbench basics

When you start up an Eclipse IDE you are prompted to provide a workspace to start up. On first startup, the path will look something like this, depending on the installation path:

```
c:\Documents and Settings\<user>\IBM\rationalsdp7.0\workspace
```

Where <user> is the Windows user ID you have logged in as.

The workspace, a Windows directory, is a private work area created for the individual developer, and it holds the following information:

- Environment metadata, such as configuration information and temporary files.
- A developer's copy of projects that they have created, which includes source code, project definition, or config files and generate files such as class files.

Resources that are modified and saved are reflected on the local file system. Users can have many workspaces on their local file system to contain different project work that they are working on or differing versions. Each of these workspaces may be configured differently, since they will have their own copy of metadata that will have configuration data for that workspace.

Archived

Important: Although workspace metadata stores configuration information, this does not mean that the metadata can be transferred between workspaces. In general, we do not recommend copying or using the metadata in one workspace, with another one. The recommended approach is to create a new workspace and then configure it appropriately.

Additionally, if you are seeing some strange results, for example, if you have opened up an old workspace with a newer Eclipse, it may be a good idea to switch to a brand new workspace.

Eclipse allows you to open more than one Workbench at a time. It opens another window into the same Workbench, allowing you to work in two differing perspectives. Changes that are made in one window will be reflected back to the other windows, and you are not permitted to work in more that one window at a time. That is, you cannot switch between windows while in the process of using a wizard in one window. Opening a new Workbench in another window is done by selecting **Window** → **New Window**, and a new Workbench with the same perspective will open in a new window.

You can also start multiple Eclipse IDEs thus allowing you to work in different workspaces. When you are asked the name of your workspace just select a different one. For example, you may use different workspaces for different versions of your applications.

The default workspace can be started on first startup of the Eclipse IDE by specifying the workspace location on the local machine and selecting the check box **Use this as the default and do not ask again**, as shown in Figure 18-3.

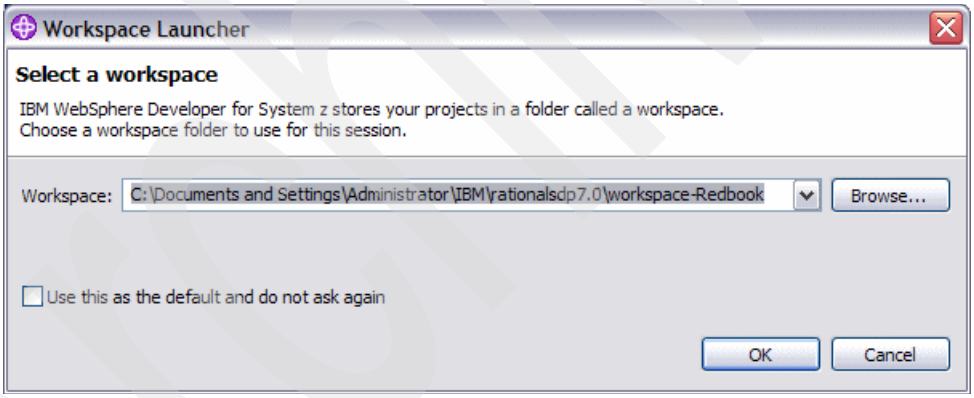


Figure 18-3 Setting the default workspace on startup

This will ensure on the next startup of The Eclipse IDE that the workspace will automatically use the directory specified initially, and it will not prompt for the workspace in the future.

If you do set the check box on this startup panel to not ask again, there is a procedure to re-initiate this, described as follows:

1. Select **Window** → **Preferences** → **General** → **Startup and Shutdown**.
2. .Check the **Prompt for workspace on startup** check box as shown in Figure 18-4, click **Apply**, and then click **OK**.

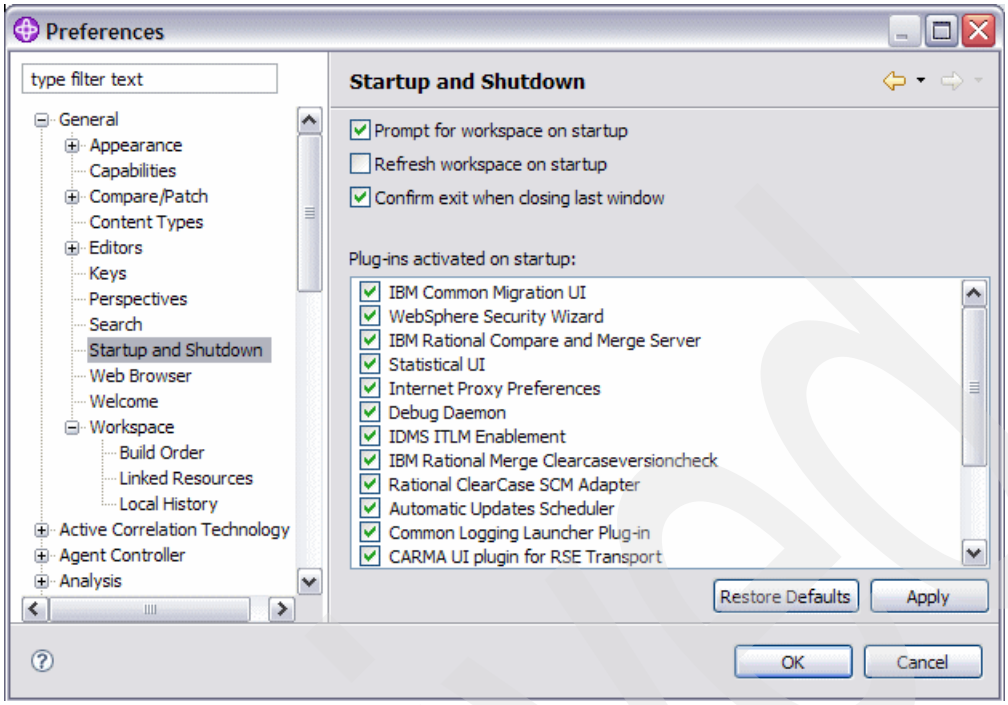


Figure 18-4 Setting the prompt dialog box for workspace selection on startup

On the next startup of Eclipse the workspace prompt dialog will appear asking the user which workspace to load up.

The other way to enforce the use of a particular workspace is by using the **-data <workspace>** command-line argument on Eclipse startup, where <workspace> is a path name on the local machine where the workspace is located and should be a full path name to remove any ambiguity of location of the workspace.

Tip: On a machine where there are multiple workspaces used by the developer, a shortcut would be the recommended approach in setting up the starting workspace location. The target would be "`<Eclipse Install Dir>\eclipse.exe`" `-data <workspace>`.

By using the -data argument you can start a second instance of Eclipse that uses a different workspace. For example, if your second instance should use the NewWorkspace workspace folder, you can launch Eclipse with this command (assuming that the product has been installed in the default installation directory):

```
"C:\Program Files\IBM\SDP70\eclipse.exe" -data C:\temp\tempspace
```

18.3.2 Perspectives, views, and editors

Eclipse supports a role-based development model. It does so by providing several different perspectives on the same project. Each perspective is suited for a particular role and provides the developer with the necessary tools to work on the tasks associated with that role.

Integrated development environment (IDE)

An integrated development environment is a set of software development tools such as source editors, compilers, and debugger, that are accessible from a single user interface.

In Eclipse, the IDE is called the *Workbench*. Eclipse's Workbench provides customizable perspectives that support role-based development. It provides a common way for all members of a project team to create, manage, and navigate resources easily. It consists of a number of interrelated views and editors.

Views provide different ways of looking at the resource you are working on, whereas editors allow you to create and modify the resource. Perspectives are a combination of views and editors that show various aspects of the project resource, and are organized by developer role or task. For example, a Java developer would work most often in the Java perspective, a Web designer would work in the Web perspective and a z/OS developer would work in the z/OS projects view if they were using WD/z.

Several perspectives are provided in the different versions of Eclipse, and team members also can customize them, according to their current role of preference. You can open more than one perspective at a time, and switch perspectives while you are working with Eclipse. If you find that a particular perspective does not contain the views or editors you require, you can add them to the perspective and position them to suit your preference.

Before describing the perspectives, we take a look at Eclipse's help feature.

Eclipse online help

Eclipse's online help system provides access to the documentation for all the plug-ins you have installed, and lets you browse and print help content. It also has a full-text search engine and context-sensitive help.

Eclipse provides the help content in a separate window, which you can open by selecting **Help** → **Help Contents** from the menu bar as shown in Figure 18-5.

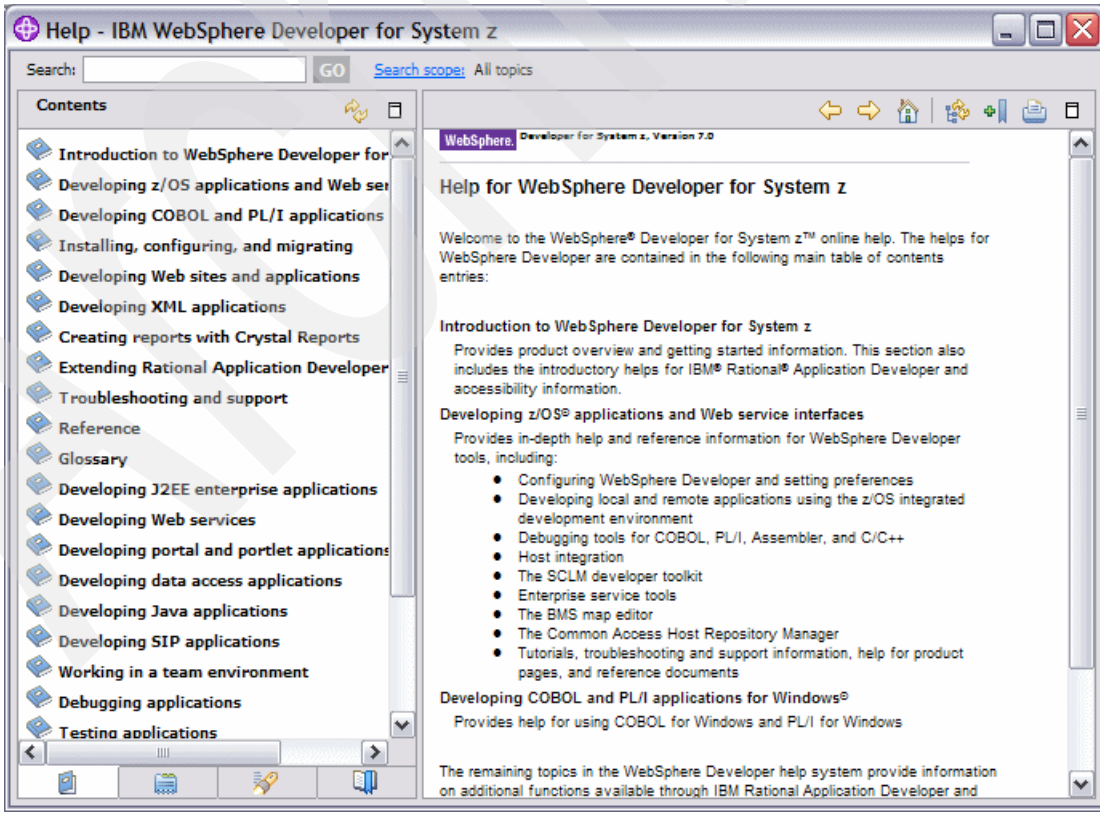


Figure 18-5 Help window

In the help window you see the available books in the left pane and the content in the right pane. When you select a book in the left pane, the appropriate table of contents opens up and you can select a topic within the book. When a page is selected, the page content is displayed in the right pane.

You can navigate through the help documents by using the Go Back and Go Forward buttons in the toolbar for the right pane. There are other buttons in this toolbar:

- Show in Table of Contents: Synchronizes the navigation frame with the current topic, which is helpful if you have followed several links to related topics in several files, and want to see where the current topic fits into the navigation path.
- Bookmark Document: Adds a bookmark to the Bookmarks view in the left pane.
- Print Page. Prints a page.
- Maximize: Maximizes the left pane to fill the whole help window.

Figure 18-6 shows the help window with the table of contents for the book, *Using the SCLM developer toolkit*.

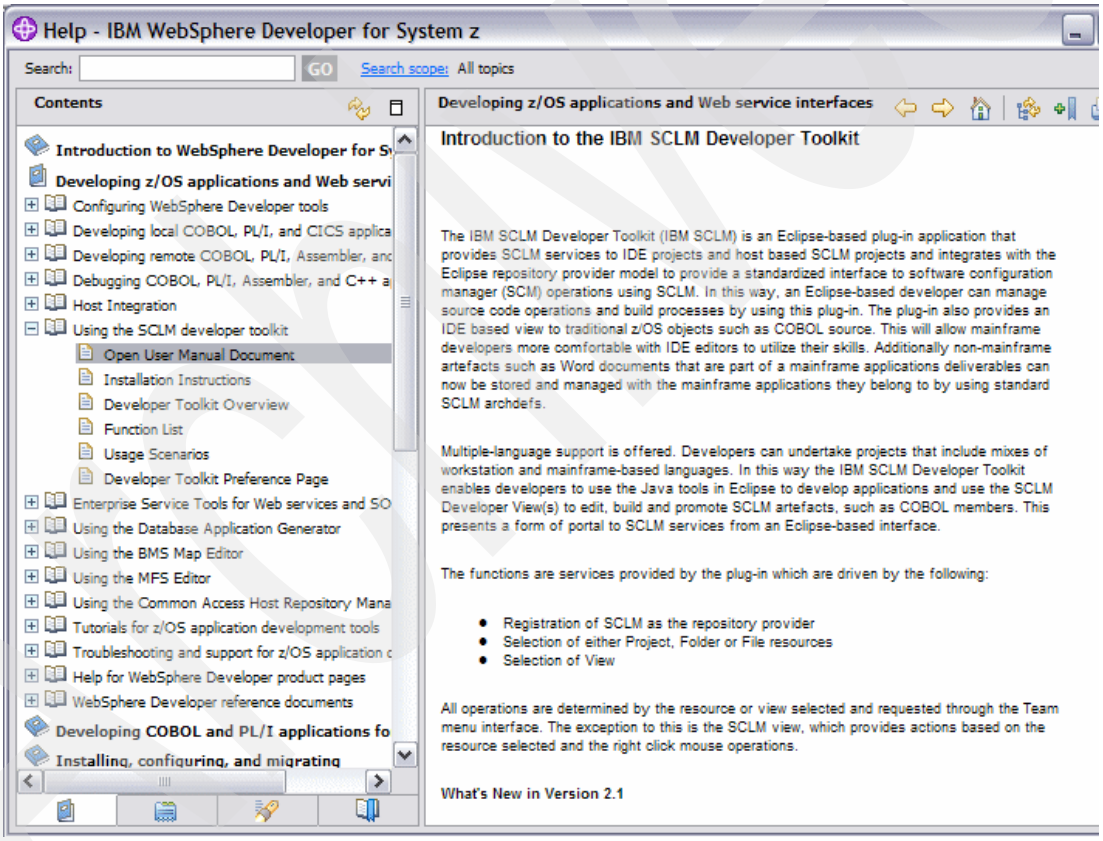


Figure 18-6 Workbench help

The Search field allows you to do a search over all books by default. The **Search Scope** link opens a dialog box where you can select a scope for your search. If you have previously defined a search list, this list of books and topics can be selected from the dialog, allowing you to choose from particular selections of books that you have found useful in the past. This is shown in Figure 18-7.

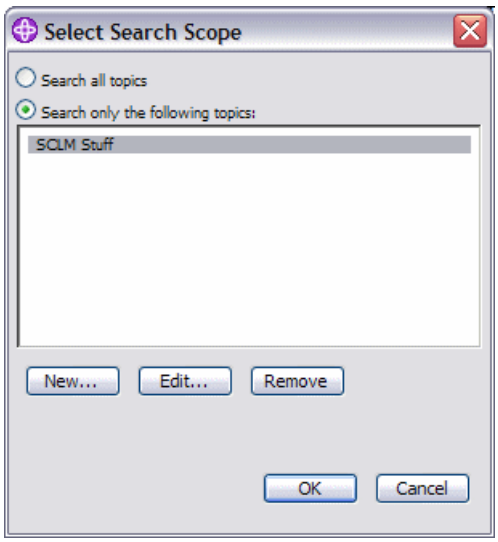


Figure 18-7 Select Search Scope dialog for help

To create a search list, click the **New...** button and enter a name for the list, or click **Edit...** if you wish to modify an existing search list. In either case, a dialog similar to the one shown in Figure 18-8 allows you to make your selection of books and topics.

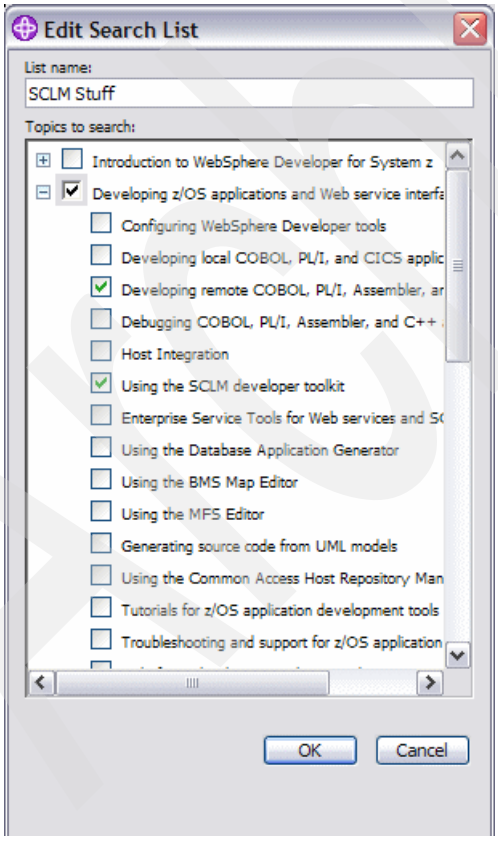


Figure 18-8 New/Edit Search List dialog for help

Archive

Note: The first time you search the online help, the help system initiates an index-generation process. This process builds the indexes for the search engine to use, which will probably take several minutes. Unlike WebSphere Studio Application Developer, the index is not rebuilt when the product is opened with a different workspace. It is also possible to prebuild the index: See the help entry under **Navigating and customizing the workbench → Extending the workbench (advanced) → Extending the platform → Reference → Other reference information → Pre-built documentation index**.

Enter your search expression in the Search field and click **Go** to start your search.

Perspectives

Perspectives provide a convenient grouping of views and editors that match the way you use Rational Application Developer. Depending on the role you are in and the task you have to do, you open a different perspective. A perspective defines an initial set and layout of views and editors for performing a particular set of development activities (for example, EJB development or profiling). You can change the layout and the preferences and save a perspective that you can have customized, so that you can open it again later. We will see how to do this later in this chapter.

Views

Views provide different presentations of resources or ways of navigating through the information in your workspace. For example, the Navigator view displays projects and other resources that you are working as a folder hierarchy, like a file system view. Rational Application Developer provides synchronization between different views and editors. Some views display information obtained from other software products, such as database systems or software configuration management (SCM) systems.

A view might appear by itself, or stacked with other views in a tabbed notebook arrangement. A perspective determines the initial set of views that you are likely to need. For example, the Java perspective includes the Package Explorer and the Hierarchy views to help you work with Java packages and hierarchies.

Editors

When you open a file, Rational Application Developer automatically opens the editor that is associated with that file type. For example, the Page Designer editor is opened for .html, .htm, and .jsp files, while the Java editor is opened for .java and .jpage files.

Editors that have been associated with specific file types will open in the editor area of the Workbench. By default, editors are stacked in a notebook arrangement inside the editor area. If there is no associated editor for a resource, Rational Application Developer will open the file in the default editor, which is a text editor.

Perspective layout

Many of Eclipse's perspectives use a similar layout. Figure 18-9 shows the layout of the SCLM Developer Toolkit perspective. Other perspectives have many of the same views such as the Editor, Navigator and Properties views. There are also views that are specific to a single perspective, such as the SCLM Explorer view.

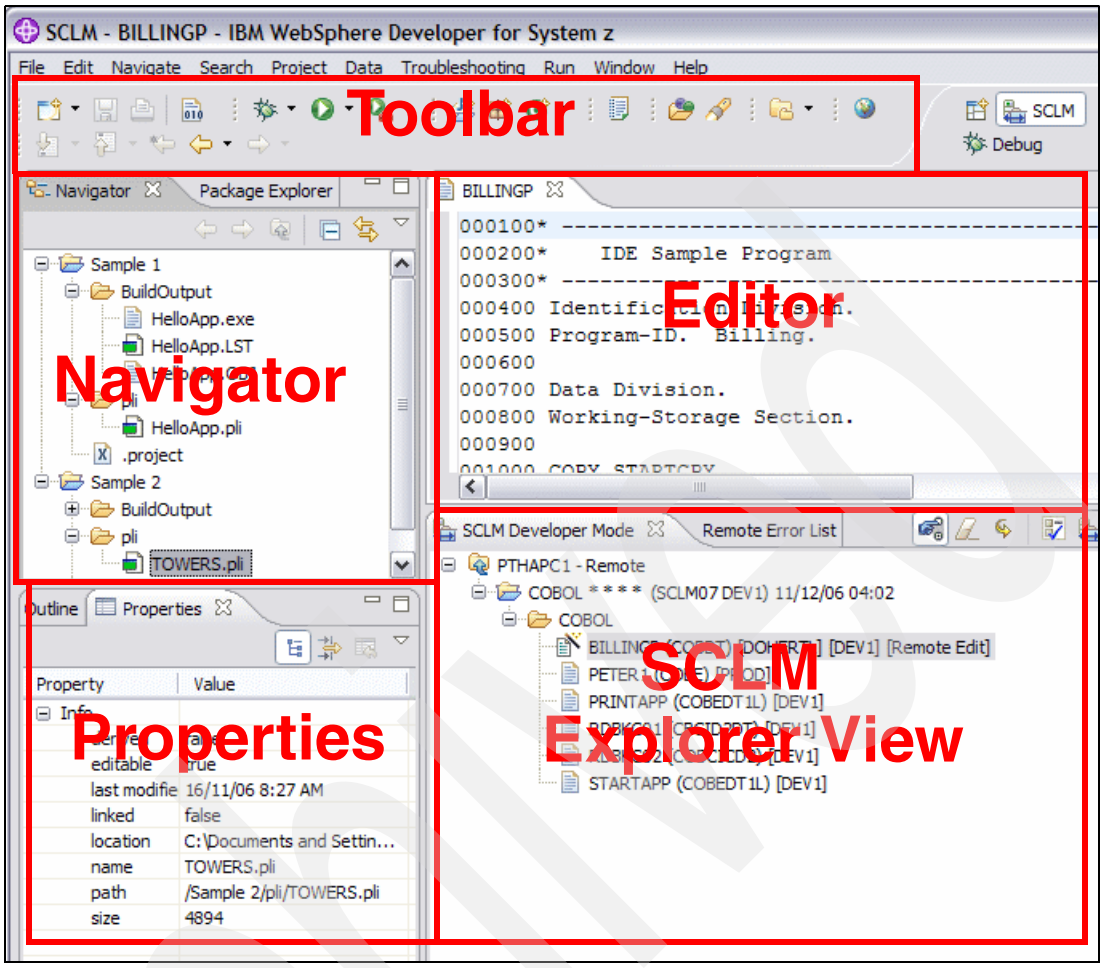


Figure 18-9 Perspective layout

On the left side are views for navigating through the workspace and checking the properties of members or files. In the middle or top right of the Workbench is larger pane, usually the source editor or the design pane, which allows you to change the code and design of files in your project. Different perspectives may have the properties in a different place or may have additional views such as a tasklist. You can also change the size of each of the panes to suite the way you use the IDE.

The content of the views is synchronized. For example, if you change a value in the Properties view, the update will automatically be reflected in the Editor view.

Switching perspectives

There are two ways to open another perspective:

- Click the Open a perspective icon ( ) in the top right corner of the Workbench working area and select the appropriate perspective from the list.
- Select **Window** → **Open Perspective**.

In either case, there is also an **Other...** option, which displays the Select Perspective dialog, as shown in Figure 18-10. Select the required perspective and click **OK**.

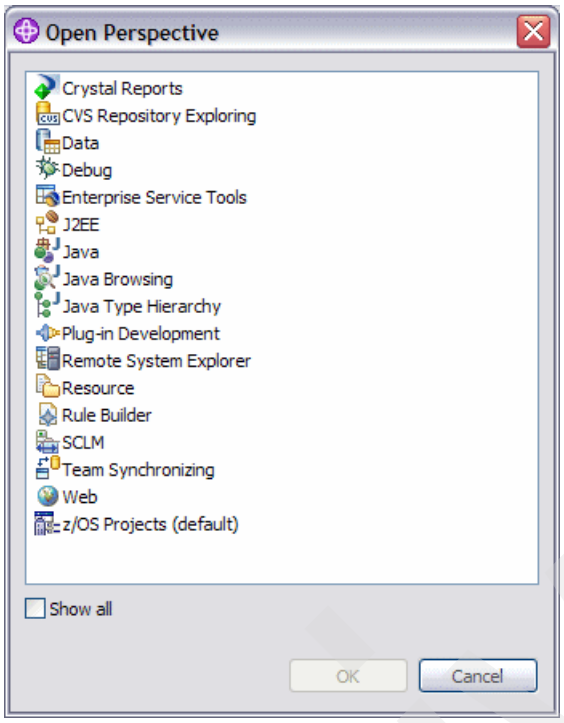


Figure 18-10 Select Perspective dialog

A button appears in the top left corner of the Workbench (an area known as the shortcut bar) for each open perspective, as shown in Figure 18-11. To switch to a particular perspective, click the appropriate button.

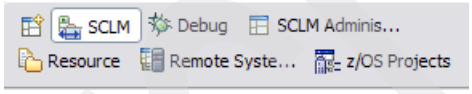


Figure 18-11 Buttons to switch between perspectives

Tips:

- The name of the perspective is shown in the window title area along with the name of the file open in the editor, which is currently at the front.
- To close a perspective, right-click the perspective's button on the shortcut bar (top right) and select **Close**.
- To display only the icons for the perspectives, right-click somewhere in the shortcut bar and deselect **Show Text**.
- Each perspective requires memory, so it is a good practice to close perspectives when not being used to improve performance.

Archived

Specifying the default perspective

Depending on the version of Eclipse you are starting there will be a different default perspective, for example, RAD uses the J2EE perspective as default, and WD/z uses the z/OS projects perspective as default. This can be changed using the Preferences dialog, for example, if you use WD/z but want the SCLM perspective as default:

1. From the Workbench, select **Window** → **Preferences**.
2. Expand **General** and select **Perspectives**.
3. Select the perspective that you want to define as the default, and click **Make Default**.
4. Click **OK**.

Context menus

Context menus are pop up menus that contain a list of options that are relevant in the context of the item, or node, being processed. They are activated by clicking the right mouse button on the item, and the menu ill then pop up. Figure 18-12 shows the context menu for opening a project in the Developer Toolkit SCLM view. It was activated by right clicking on the SCLM node.

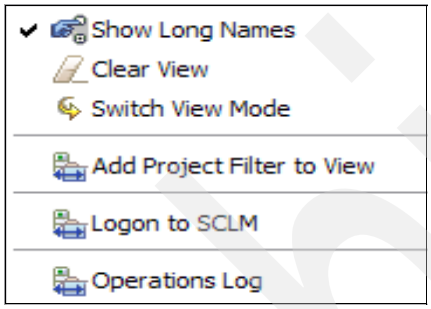


Figure 18-12 Sample context menu

More information

For more detailed information on the perspectives you are likely to work in, see the online help for the Eclipse plug-in that you are using. Additional information on many of the standard plug-ins can be found in the Redbooks publication, *Rational Application Developer V6 Programming Guide*, SG24-6449.

Archived

19

SCLM Developer Toolkit configuration

In this chapter we describe some of the configuration options with SCLM Developer Toolkit that relate to both mainframe development and Java/J2EE development. We look at the setup of installation and SCLM project specific configuration items. We also describe setting up your Integrated Development Environment (IDE) and setting Eclipse and Developer Toolkit preferences to better utilize the features in the IDE.

Archived

© Copyright IBM Corp. 2007. All rights reserved.

19.1 Overview

SCLM Developer Toolkit is split into two parts:

- Using SCLM for traditional mainframe development through an Eclipse IDE
- Using SCLM on z/OS to store and manage your Java/J2EE and other open systems development applications

The advantage of using SCLM Developer Toolkit for both mainframe and distributed development is that all of your code, for your full enterprise application, is stored and managed in one place, on z/OS. The security, stability, and backup/recoverability of the z/OS system is one of the reasons that many customers are looking for this in an SCM solution.

Additionally, many newer developers are not as familiar with the z/OS interface for software development. However, the z/OS platform is as strong as ever. To better utilize the skills of these developers as they begin to work in the marketplace, we can provide them with tools that they have been used to using during their training. To this end, providing them with an IDE to allow them to do their work will shorten the time they need to become productive. They no longer have to learn all aspects of the z/OS platform in one go.

Coupled with SCLM Developer Toolkit other z/OS functions are provided through the z/IDE available with WebSphere Developer for System z (WD/z). So editing non-SCLM data sets and members is also possible as well as submitting jobs, checking on the status of completed batch jobs among other tasks such as utilizing editors with context sensitive help.

Using WD/z users can also develop their COBOL or PL/I programs in isolation on their workstation, using the available debugger, before moving their code into a secure SCLM environment on the mainframe. At which point normal SCLM Edit, Build, and Promote development can be used, in addition to utilizing the remote debugging facilities provided by Debug Tool and WD/z. A more detailed discussion on debugging with SCLM, including using the remote debugger, was covered in Chapter 14, “Debugging and fault analysis with SCLM” on page 301.

19.2 Setting up SCLM Developer Toolkit and your SCLM projects (Administrator task)

Depending on the type of Eclipse you have installed, SCLM Developer Toolkit into will vary the connection method used. If you are running WD/z, then the connection protocol is Remote Systems Explorer (RSE) as provided with WD/z. Any other Eclipse environment that you install into will activate the HTTP server connection. Once SCLM Developer Toolkit is installed and the required transport mechanism has been configured, your users will be able to start working with their SCLM projects right away. Developer Toolkit uses standard SCLM services to list and work with members so there are no administrator specific tasks that need to be performed.

However, there are some project specific tasks that can be performed to tailor the behavior of SCLM Developer Toolkit.

19.2.1 ISPF.conf

In order for SCLM Developer Toolkit to perform the SCLM services, it must first set up an ISPF environment. The ISPF.conf file stored by default in **`/etc/SCLMDT/CONFIG/ISPF.conf`** and defines the required ISPF allocation as shown in Example 19-1.

Example 19-1 Required ISPF data sets as specified in ISPF.conf

```
* The libraries beginning BZZ.* are for the Breeze product and are
* included to show how multiple data sets are added to the concatenations.
* These should be removed if the Breeze product is not installed.

sysproc=BZZ.SBZZCLIB,ISPF.ZOSR8.SISPCLIB
ispmlib=ISPF.ZOSR8.SISPMENU
isptlib=ISPF.ZOSR8.SISPTENU
ispplib=ISPF.ZOSR8.SISPPENU
ispslib=BZZ.SBZZSENU,ISPF.ZOSR8.SISPSENU,ISPF.ZOSR8.SISPSLIB
ispllib=BWB.V2R1.SBWBLOAD,BZZ.SBZZLOAD,ISPF.ZOSR8.APARLOAD


* OPTIONAL:
* Include below your own additional user exec for ISPF allocation.
* The allocation exec will be passed the following parameters
* PROJECT, PROJECT DEFINITION, USERID.
* This can add allocation refinement by project or userid .
* A sample exec job is found in 'BWBISPF2'.
* If required remove the * below & change HLQ.test.exec to user defined
* data set.

allocjob = SCLM.PROJDEFS.REXX(SCLMALOC)
```

You should specify any libraries that contain members that might be required by the SCLM processes. For example, if you have a build translator or exit that uses the ISPLNK call method, then the member executed must be in the ISPF concatenation, so the PDS containing the member should be specified here. Additionally, if you are running batch builds, the FLMLIBS member that specifies the ISPF data sets for batch builds should be available in the ISPSLIB concatenation as SCLM Developer Toolkit uses standard file tailoring to include the required SCLM skeletons.

If you have a requirement to have SCLM project specific allocations, then this is also possible. SCLM Developer Toolkit provides, in the ISPF.conf, the ability to call an additional exec to perform allocations. This call is also shown above in Example 19-1. There is a sample version of this file provided in the supplied SBWBSAMP library called BWBISPF2.

This file can then make any changes necessary for project specific allocations. It should be noted that at the time this exec is called the ISPF environment has not been set up yet, so you are not able to use the LIBDEF service to make the alteration to the allocation. You can however reallocate the already allocated ISPF DDs by checking the current allocation and then reallocating with your data sets in front. An example of this is shown in Example 19-2.

Example 19-2 Sample REXX to reallocate SYSPROC, ISPSLIB and ISPMLIB

```
/* REXX */
   parse arg $args
   parse var $args PROJ PROJDEF USERID
   say '***************************** '
   say 'Running DT allocation job'
   say 'PROJ    = 'PROJ
   say 'PROJDEF = 'PROJDEF
   say 'USERID  = 'USERID
   say '***************************** '
```

Archived

```
/* Work out current allocations using LISTA */
   x = outtrap('out.','*','concat')
   "lista status"
   lc = rc
   x = outtrap('off')

/* Specify DDNAMES you need to reallocate   */
   DDS = 'ISPPLIB ISPSLIB'
   DD# = 0
   DSN. = ''

   I = 2
   Do While I <> OUT.0
      If Substr(OUT.I,1,1) = ' ' Then
      Do
         Parse var OUT.I DDN . 12 DISP
         If DDN = '' Then
            DSN.DD# = DSN.DD# "'"DSN"'"
         Else
         Do
            DD# = Wordpos(DDN,DDS)
            DSN.DD# = "'"DSN"'"
         End
      End
      Else
         Parse var OUT.I DSN .

      I = I + 1
   End

/* Perform allocations                      */
Select
  When (PROJ = 'SCLM07') Then
  Do
     Address TSO
     "ALTLIB ACTIVATE APPLICATION(EXEC) DATASET('SCLM07.PROJDEFS.REXX')"
     If rc = 0 Then
        Say 'SYSEXEC : SCLM07.PROJDEFS.REXX concatenated'

     "FREE  FILE(ISPSLIB)"
     "ALLOC FILE(ISPSLIB) DATASET('SCLM07.PROJDEFS.SKELS',"DSN.2") SHR REUSE"
     If rc = 0 Then
     Do
        Say 'ISPSLIB : Reallocated with following concatenation -'
        Say "'SCLM07.PROJDEFS.SKELS',"DSN.2
     End

  End
  When (PROJ = 'SCLM01') Then
  Do
     Address TSO
     "ALTLIB ACTIVATE APPLICATION(EXEC) DATASET('SCLM01.PROJDEFS.REXX')"
     If rc = 0 Then
        Say 'SYSEXEC : SCLM01.PROJDEFS.REXX concatenated'
     "FREE  FILE(ISPMLIB)"
```

Archived

```
     "ALLOC FILE(ISPMLIB) DATASET('SCLM01.PROJDEFS.MESSAGES',"DSN.1") SHR REUSE"
     If rc = 0 Then
     Do
        Say 'ISPMLIB : Reallocated with following concatenation -'
        Say "'SCLM01.PROJDEFS.MESSAGES',"DSN.1
     End
     "FREE  FILE(ISPSLIB)"
     "ALLOC FILE(ISPSLIB) DATASET('SCLM01.PROJDEFS.SKELS',"DSN.2") SHR REUSE"
     If rc = 0 Then
     Do
        Say 'ISPSLIB : Reallocated with following concatenation -'
        Say "'SCLM01.PROJDEFS.SKELS',"DSN.2
     End
  End
  Otherwise
     NOP
End

Exit
```

19.2.2 TRANSLATE.conf

The TRANSLATE.conf file provides keywords to determine how code is stored within SCLM.

The TRANSLATE.conf member must exist and must have values set for the ASCII and EBCDIC code pages. If you are only going to be looking at your traditional SCLM types, such as COBOL, then nothing else needs to be set in the TRANSLATE.conf.

If you are going to utilize the ability to store ASCII files, such as Word documents, directly in SCLM, then there is a provision to specify the name of the language definition that files being binary transferred and stored are saved with. This is the TRANLANG keyword.

Additionally, the SCLM language definitions name also controls whether long name files are converted to suitable valid short hostnames to store in SCLM. This long to short name mapping is controlled by the SCLM long/short name translate file. The keyword to specify if certain languages should be translated from long to short names is LONGLANG.

For example, you have a number of Word documents that you want to store in SCLM. In this case, files of this type cannot be edited on the mainframe. So there is no point in translating them to EBCDIC and they should just be stored in ASCII. Perform the following steps:

- Create an SCLM Language translator based on the sample BWBTRANJ in the SBWBSAMP library. Basically this translator is nothing more than a FLMLANGL macro as shown below in Example 19-3.You could call it BINARY (as this is how it will be stored), or if you want to be specific, call it DOC.

Example 19-3 Example of SCLM Translator for Word documents

```
*********************************************************************
*        WORD DOCUMENT LANGUAGE DEFINITION FOR SCLM                 *
*********************************************************************
         FLMLANGL    LANG=DOC,VERSION=1.0
*
```

- Create an SCLM type in the hierarchy to store these binary objects. For this example create a type of DOC with a DCB of RECFM(VB) and LRECL(256).

- As this file has a long name on the workstation, such as InstallGuide.doc, you need to make sure this is mapped to a generated short name in the SCLM PDS where you will store it. Therefore create a LONGLANG entry for the BINARY or DOC language, whatever you have called it. For example, **LONGLANG BINARY**.
- Add a TRANLANG entry for the BINARY or DOC language, whatever you have called it. For example, **TRANLANG BINARY**.
- When the Word document is added to SCLM you assign it a language of DOC. SCLM Developer Toolkit will check the TRANSLATE.conf and see that there is a LONGLANG keyword for the language of DOC, so will generate a short name for the file to be stored in the DOC library. As there is also a TRANLANG keyword for the language of DOC SCZLM Developer Toolkit will not do an ASCII to EBCDIC translation when the member is stored. When the member is checked out in SCLM it will be transferred down to the Eclipse workspace without any EBCDIC to ASCII conversion. Word is then started to work on the file. When changes are complete the Word document is checked back into SCLM and again no ASCII to EBCDIC translation takes place.

Example 19-4 below shows an example TRANSLATE.conf member.

Example 19-4 Sample TRANSLATE.conf member

```
* -------------------- CODEPAGE SECTION ----------------------
*
* Below change ASCII and EBCDIC codepages
* used in ASCII/EBCDIC client/host text translation.
* ISO8859-1 is the default ASCII codepage used.
* IBM-1047 is the default EBCDIC codepage used.
*
CODEPAGE ASCII = ISO8859-1
CODEPAGE EBCDIC = IBM-1047
*
* ------------ ASCII to EBCDIC TRANSLATION SECTION -----------
*
* Below add SCLM Language type for no ASCII/EBCDIC
* translation to the host (File will be binary transferred).
* If files were ASCII text they will remain in that ASCII codepage.
* ASCII/EBCDIC translation will take place for files by default.
*
TRANLANG JAVABIN
TRANLANG J2EEBIN
TRANLANG J2EEOBJ
TRANLANG TEXTBIN
TRANLANG BINARY
TRANLANG DOC
TRANLANG XLS
*
* ------------ LONG/SHORT NAME TRANSLATION SECTION -----------
*
* Longname to shortname translation implies the longname file on the
* Client (including directory package structure) will be mapped to
* a valid host member name of 8 characters and stored in SCLM using
* this translated host short name.
* No translation involves the Client file already being in host
* short name format (8 characters or less) and stored as is.
*
* Add Language types for longname to shortname translation
```

Archive

```
* using the keyword LONGLANG.
*
LONGLANG JAVA
LONGLANG J2EEPART
LONGLANG JAVABIN
LONGLANG J2EEBIN
LONGLANG J2EEOBJ
LONGLANG DOC
LONGLANG XLS
*
```

19.2.3 Site and project configuration files

During product installation a directory will be created under the CONFIG directory to hold SCLM project specific settings. By default, that directory is **/etc/SCLMDT/CONFIG/PROJECT**.

As an SCLM Administrator you can create project specific configuration files in this directory where SCLM Developer Toolkit, prior to every operation will check to see if there are settings that need to be applied. You can also set up site specific values that will be applied to all SCLM projects and then if you wish, override these on a project by project basis. Additionally, the installation specific configuration values in the TRANSLATE.conf file can also be overridden on a project by project basis.

The search order is as follows:

1. Combination of programmed defaults and **/etc/SCLMDT/CONFIG/TRANSLATE.conf**

   Certain parameters are defaulted programmatically if they are not set in the SITE or project files. Others are set in the TRANSLATE.conf member.

2. **/etc/SCLMDT/CONFIG/PROJECT/SITE.conf**
3. **/etc/SCLMDT/CONFIG/PROJECT/*SCLMProject*.conf** where ***SCLMProject*** is the project name of your SCLM project.

The following settings can be set in either the SITE.conf, a project specific config file or left as default:

- **BUILDAPPROVER/PROMOTEAPPROVER**

  This option is set to BREEZE is you want to activate the IBM Breeze for SCLM panel options on the promote panel in SCLM Developer Toolkit

- **CCODE**

  If your site or projects standards dictate that a change code must be entered when adding or editing a member, this option should be set to **Y**. Any panel in SCLM Developer Toolkit that has a change code field specified on the panel will make entry mandatory.

- **FOREGROUNDBUILD/FOREGROUNDPROMOTE**

  Many sites do not allow foreground compiles for performance reasons. To this end these options restricts SCLM Developer Toolkit users to batch builds and promotes

- **BATCHBUILDn/BATCHPROMOTEn/BATCHMIGRATEn**

  These options allow you to specify separate default jobcards for the batch build, promote and migrate options.

- **BUILDSECURITY/PROMOTESECURITY/DEPLOYSECURITY**

  These options cause the additional SAF/RACF security to be invoked. This also provides the facility to invoke the build, promote or deploy action using a surrogate userid.

- **ASCII/EBCDIC**

  These parameters allow you to override at a project level, the default ASCII and EBCDIC codepages specified in the TRANSLATE.conf. This means that on a single installation it is possible to run multiple SCLM projects that contain members that use different codepages. For example, during bidirectional testing we used two separate SCLM projects, one to hold Hebrew source and the other to hold Arabic source. By using the project configuration file we had separate code pages for both.

- TRANLANG/NOTRANLANG

  These options add or remove TRANLANG keywords that have been set previously in the TRANSLATE.conf

- LONGLANG/NOLONGLANG

  These options add or remove LONGLANG keywords that have been set previously in the TRANSLATE.conf

Example of a project specific configuration file

Example 19-5 below is an example of a project specific configuration file. It could also be set up as a site specific configuration file. By default this will be stored in **/etc/SCLMDT/CONFIG/PROJECT/SCLM07.conf.**

Example 19-5 Project specific SCLM Developer Toolkit option - SCLM07.conf

```
* Project - SCLM07
* Below are a number of project specific options used to
* determine the behaviour of the Eclipse front-end.
*
* These will override the SITE.CONF file
*
*
* SCM Approver processing applies to this project?
BUILDAPPROVER=BREEZE
PROMOTEAPPROVER=BREEZE
*
* Change Code entry on check-in is mandatory?
CCODE=Y
*
* Foreground or On-line builds/promotes allowed for this project?
FOREGROUNDBUILD=N
FOREGROUNDPROMOTE=N
*
* Batch Build default jobcard
BATCHBUILD1=//SCLMBILD JOB (#ACCT),'SCLM BUILD',CLASS=A,MSGCLASS=X,
BATCHBUILD2=//        NOTIFY=&SYSUID,REGION=512M
BATCHBUILD3=//*
BATCHBUILD4=//*
*
* Batch Promote default jobcard
BATCHPROMOTE1=//SCLMPROM JOB (#ACCT),'SCLM PROMOTE',CLASS=A,MSGCLASS=X,
BATCHPROMOTE2=//        NOTIFY=&SYSUID,REGION=128M
BATCHPROMOTE3=//*
BATCHPROMOTE4=//*
*
* BUILD Security flag for SAF/RACF security call/Surrogate ID switch
BUILDSECURITY=N
```

```
* PROMOTE Security flag for SAF/RACF security call/Surrogate ID switch
PROMOTESECURITY=N
* J2EE DEPLOY security flag for SAF/RACF security call/Surrogate ID switch
DEPLOYSECURITY=N
*
* Arabic Codepage overrides
*
ASCII=UTF-8
EBCDIC=IBM-420
*
* Project specific TRANLANG and LONGLANG entries
*
TRANLANG=DOC
LONGLANG=DOC
```

The ASCII/EBCDIC codepages will ensure that code transferred to and from the Eclipse IDE will be translated correctly between the Arabic codepages. The SCLM Language definition of DOC will be added to default TRANLANG and LONGLANG specified in the TRANSLATE.conf file.

Common site settings versus project specific setting

The settings in the SITE config file and project specific config files can be set up in a number of ways due to the flexible nature of specifying these settings. However, it is worth looking at the settings in such a way that you can consider some of them common settings across all SCLM projects and some of the settings as project specific. This way you can set up the SITE.conf with the common settings and therefore not duplicate them in the project specific files. So for example, if all of your SCLM projects enforce batch builds, Breeze support and change code entry then these could be placed in the SITE.conf file. But due to the actual nature of the specific SCLM project the default jobcards and additional TRANLANG/LONGLANG keywords could all be placed in a project specific config file.

19.3 Setting up your IDE environment (user task)

There are SCLM Developer Toolkit specific preferences that can be modified, but additionally there are preferences that can be set that are done in conjunction with other standard Eclipse preferences. These are things that the individual user will do on their workstation. Before starting to work in SCLM Developer Toolkit you should review the default preferences. This is done by going to **Window** → **Preferences** → **Team** → **SCLM Preferences** in the Eclipse IDE. However, if you want to jump straight in to using SCLM Developer Toolkit, you can review your preferences at any time, the defaults set are the most commonly used and will most likely be adequate for your needs.

Main preferences panel

The main preferences panel is shown in Figure 19-1.

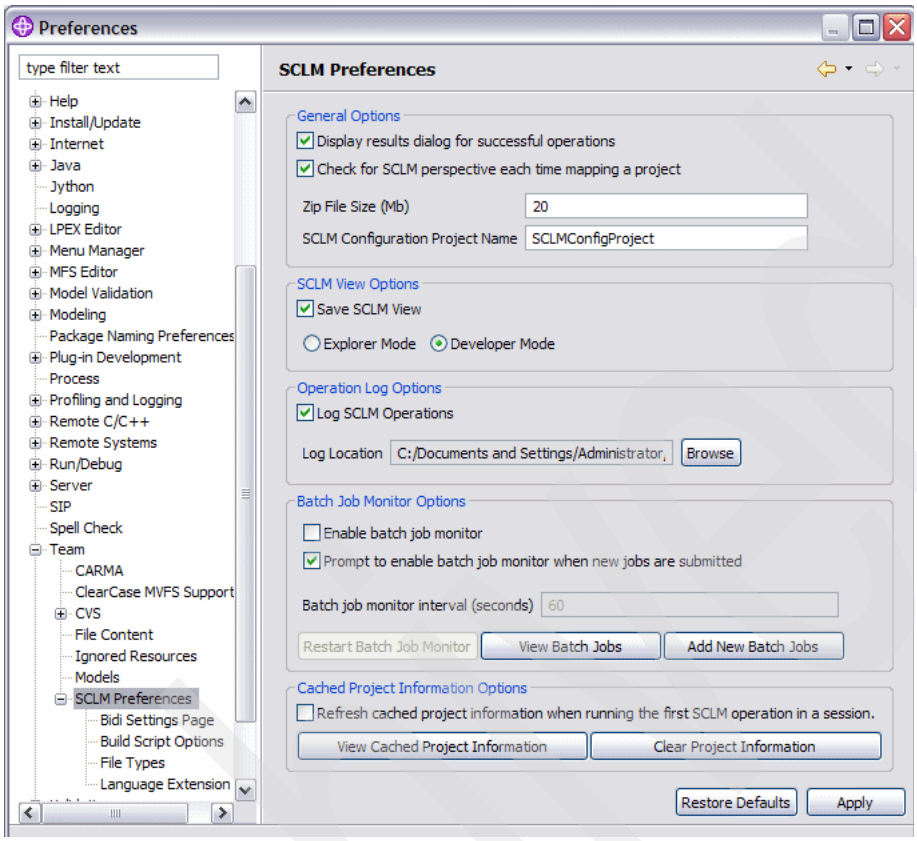


Figure 19-1 SCLM Preferences

The SCLM Preferences page allows you to specify the following settings:

- **Display results dialog for successful operations**

  The SCLM service operations return a dialog with a return code and process description. It is possible to suppress this dialogue by checking the return dialog selection. If this is checked, all process returns will be presented from the Eclipse Task list.

- **Check for SCLM perspective each time mapping a project**

  Enable the SCLM perspective each time SCLM is mapped to a project in the IDE view.

- **Zip File Size**

  The maximum size of the zip file for multi-file transfers, which enables a user to specify the upper limits of the zip file. On some systems, an unlimited amount causes a file truncation error in transmission. If the transfer is greater than the set amount, multiple files are used.

- **SCLM Configuration Project Name**

  The name of the configuration project being used. This is a temporary project that SCLM Developer Toolkit creates in the IDE View to facilitate using the functions of Eclipse for items in the SCLM View. The user does not need to work directly with this project.

- **Save SCLM View**

  Whether to save the SCLM view between sessions.

- **Explorer Mode/Developer Mode**

  Which SCLM View mode to use. Developer Mode provides a view based on type, similar to SCLM 3.1 is ISPF. Explorer Mode provides a full listing of SCLM members by group, type and member.

- **Log SCLM Operations**

  Whether SCLM operations are logged, which enables the viewing of operations in terms of parameters passed and SCLM output. Even if you chose not to show a return code dialog for successful operations, they will still be logged if this box is checked

- **Log Location**

  The location of the SCLM log file.

- **Enable Batch Job Monitor**

  Whether the batch monitor is active. The batch monitor is a daemon that periodically checks for jobs listed in the Batch Job List (click **View Batch Jobs**). If a job is in the list, one it completes, the Batch Job Monitor will pop up a message dialog to the user telling them the job has finished.

- **Prompt to enable batch job monitor....**

  If the Monitor is turned off, you will be prompted with a dialog to turn it on when you submit a batch job.

- **Batch job monitor interval**

  How often the batch job monitor polls checking for job status.

- **Batch Monitor buttons**

  The current jobs being monitored can be listed and deleted if necessary. Also, the monitor can be restarted, for example, when a more frequent time interval is specified.

- **Refresh cached project information**

  The complete list of cached project information can be viewed and cleared if required. SCLM Developer Toolkit will automatically retrieve project information for projects being accessed that it does not have information for. Additionally, the project information can be selected to be refreshed when the first SCLM operation is performed in that session.

Language extension preference page

One of the powerful things about eclipse is the ability to associate various editors with file extensions. For example you may have a third party editor that you like to use for editing HTML. SCLM utilizes this feature by using the Language Extension preferences.

Obviously a normal member in a PDS is a standard 8 character member name, and as such, it does not have a file extension. You can, however, associate an SCLM language with a file extension. So when a member is edited in the IDE using SCLM Developer Toolkit, as it is transferred to the workstation, the language is checked and matched against the language extension table. If an extension is found, the Eclipse file types table is checked to see which editor is to be opened with the specified extension.

For example, members of COBOL language can have a .cbl extension. In this way editor preferences can be selected based on remote system file types.

Archived

To add an extension, click the **New** button on the Language Extension preference panel and enter the SCLM Language and the extension that it will relate to. A common example is to associate a COBOL language, COBE, to a .cbl extension. This can be seen in Figure 19-2.

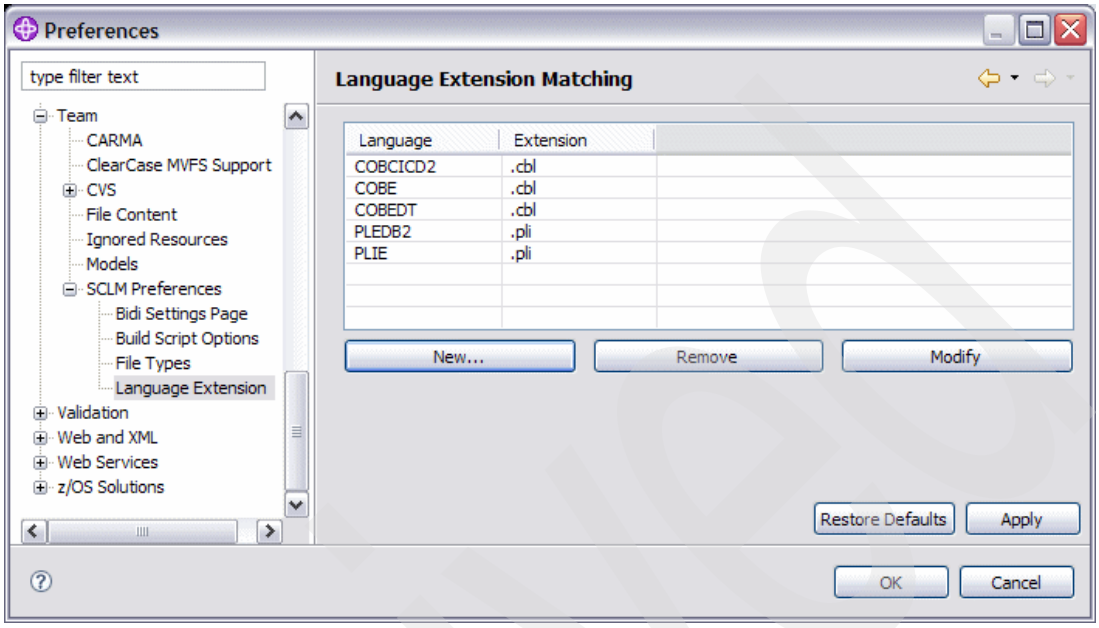


Figure 19-2 SCLM Preferences - Language Extension matching

By checking the associations that *.cbl has in the base eclipse file types, you can associate more advanced editors to work on your code. Select **Windows** → **Preferences** → **General** → **Editors** → **File Associations** to see a listing of file extensions. If the one you want is not displayed, such as *.cbl, you can add it by clicking the **Add** box next to the File Types window. Highlight the language you have just added and click the **Add** button next to the Associated Editors. From here you can select an internal Editor or an External program.

The completed dialog for the .cbl extension is shown in Figure 19-3.

For example, if you are using WebSphere Developer for System z, the LPEX editor is installed as a plug-in. You can associate this, if it is not done so already, with the .cbl extension. This way when you edit your SCLM COBOL program, the LPEX editor is used instead of a text editor. This can give you the look and feel of a z/OS editor, such as ISPF or XEDIT, as well as color coding to assist in development.

Another example is to associate the .doc extension with Microsoft® Word, By default this will not be set up in Eclipse. Add the *.doc extension as detailed above. Then click the **Add** button next to the Associated Editors and select the External program radio button. Finally select the Microsoft Word program from the list. This is shown in Figure 19-4.

With this example it is not necessary to associate an SCLM language with the extension. This is because with SCLM Developer Toolkit, you will associate Word documents with a language type that will invoke long to short name translation. So SCLM itself will know that Word documents stored in SCLM have a long name and as such will open these files with their long names, at which point Eclipse will use the file extension to open the correct editor.

Additional file type associations can be added in for *.xls, *.pdf. Also if you have any additional normal members in SCLM that you wish to open with a special editor, add entries in the Language Extension preferences panel.

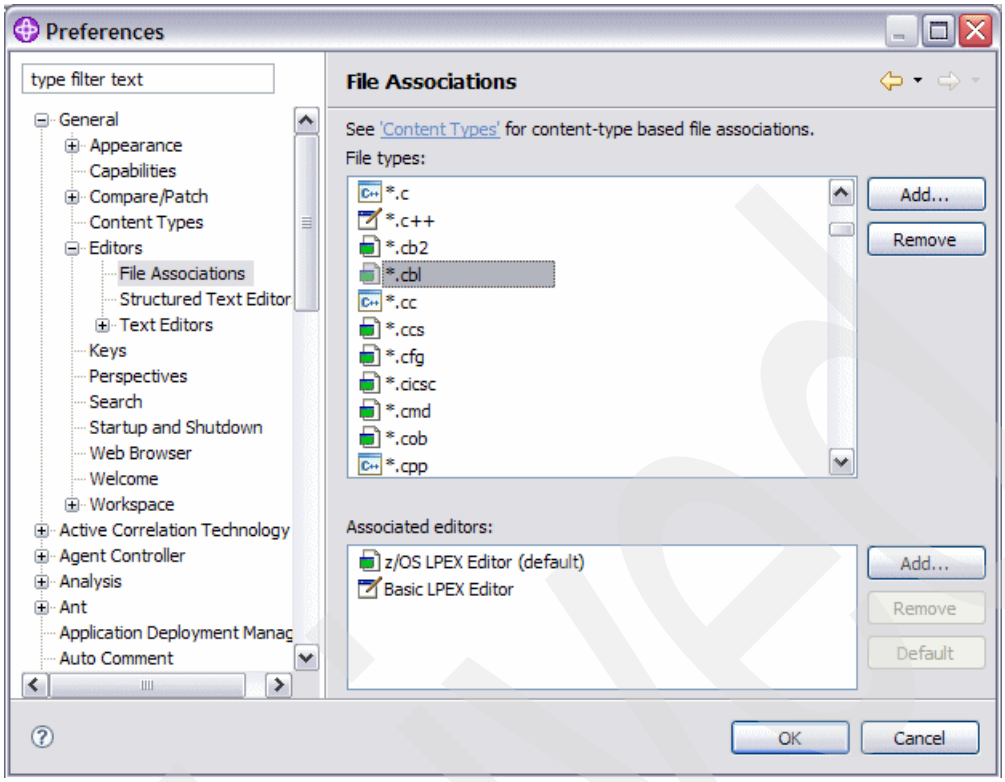


*Figure 19-3 Eclipse - File extension to Editor associations for *.cbl*

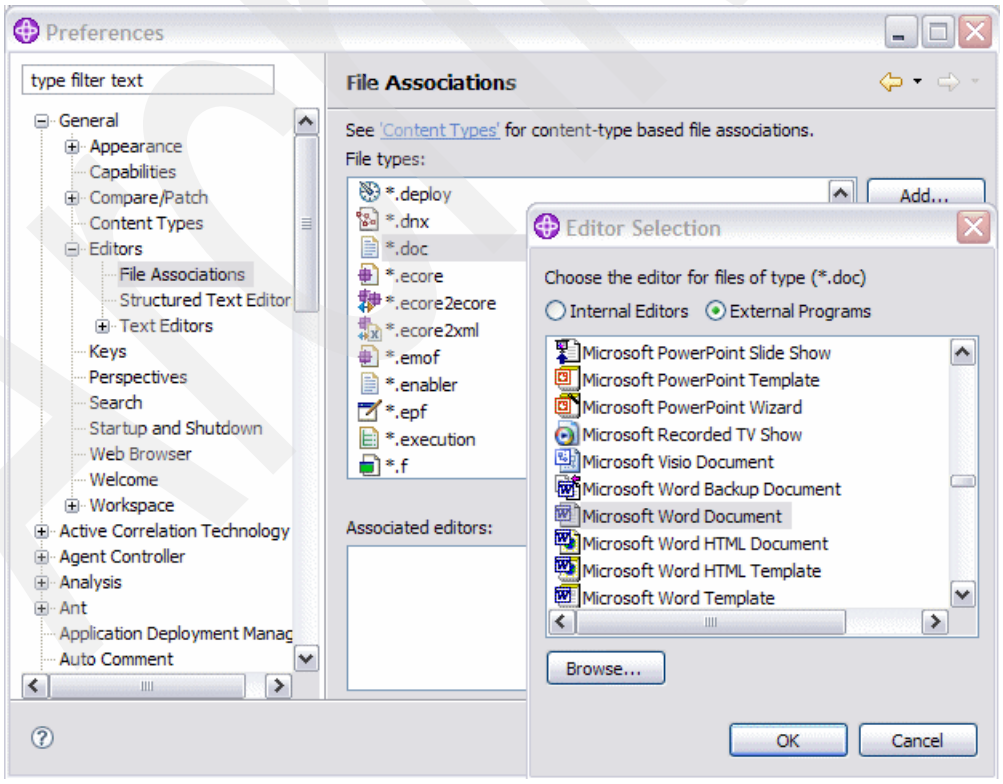


*Figure 19-4 Eclipse - File extension to Editor associations for *.doc*

File type preferences page

Due to the way SCLM stores members, it is necessary to describe a Type and Language for each type of file. To automate this function, the IDE preferences can be set so that these properties are set at file creation time based on the file type (extension). If you are migrating many COBOL or PL/I applications developed in WD/z to SCLM on the mainframe, you can tell SCLM what Type and Language to use for those extensions as default. This will save time associating groups of members with a type and language if you have a standard set for these values.

For Java programs, due to the way that each Java project needs to be stored in a separate SCLM type, the type definition will need to be changed when the Java parts are added to SCLM. However by setting a file type you can set the language so that it does not need to be set for each member.

To add a file type, click the **New** button and add the details for file extension and associated SCLM properties in the New Pattern page. These values can also be modified at a later time. By defining all of the SCLM properties, any files added to SCLM via the IDE view will have the correct type and language settings already set.

An example of this dialog can be seen here in Figure 19-5.

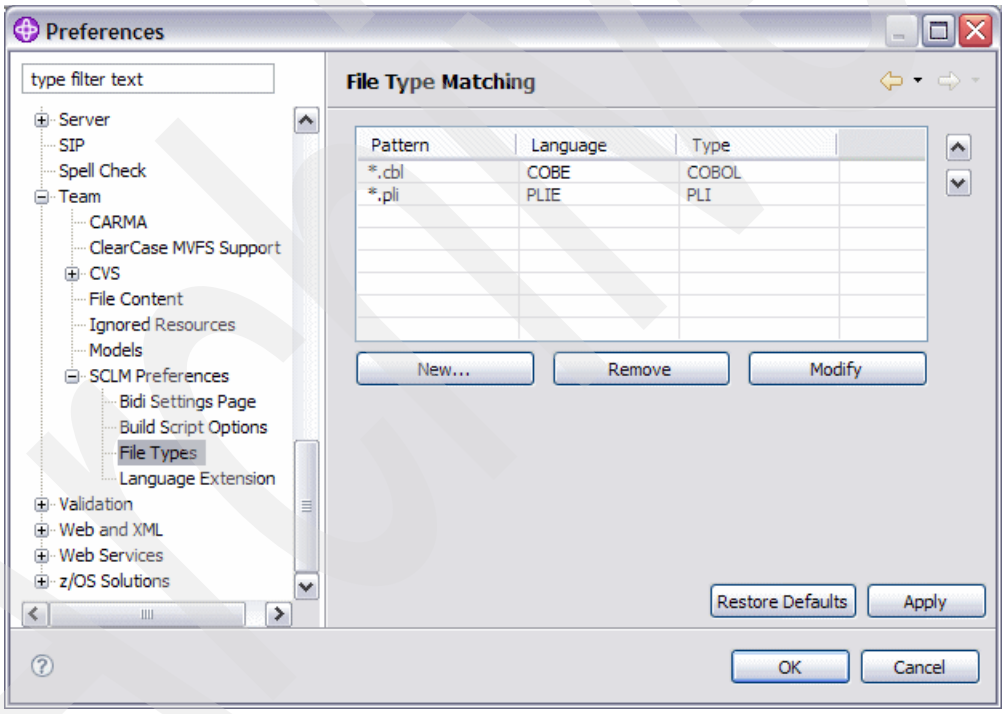


Figure 19-5 SCLM Preferences - File Type matching

20

SCLM Developer Toolkit for mainframe development

In this chapter we describe how to use SCLM Developer Toolkit for mainframe development. We start by using the SCLM view and show the similarities with the z/OS SCLM interface. We then look at how, utilizing SCLM Developer Toolkit's ability to store long name file names, we can use aspects of workstation development along with traditional z/OS development.

Archived

© Copyright IBM Corp. 2007. All rights reserved.

20.1 Working with the SCLM explorer view

This section describes a typical development scenario of editing, building, and promoting a traditional COBOL application using SCLM Developer Toolkit. We also look at some of the additional options such as versioning and running database content reports (DBUTIL).

When you first start Eclipse, depending on whether that Eclipse is IES, RAD, or WD/z for example, you may have to open the SCLM perspective. To do this, go to **Select Window** → **Open Perspective** → **Other...**. and select SCLM from the list displayed. You then see the SCLM perspective as shown in Figure 20-1.

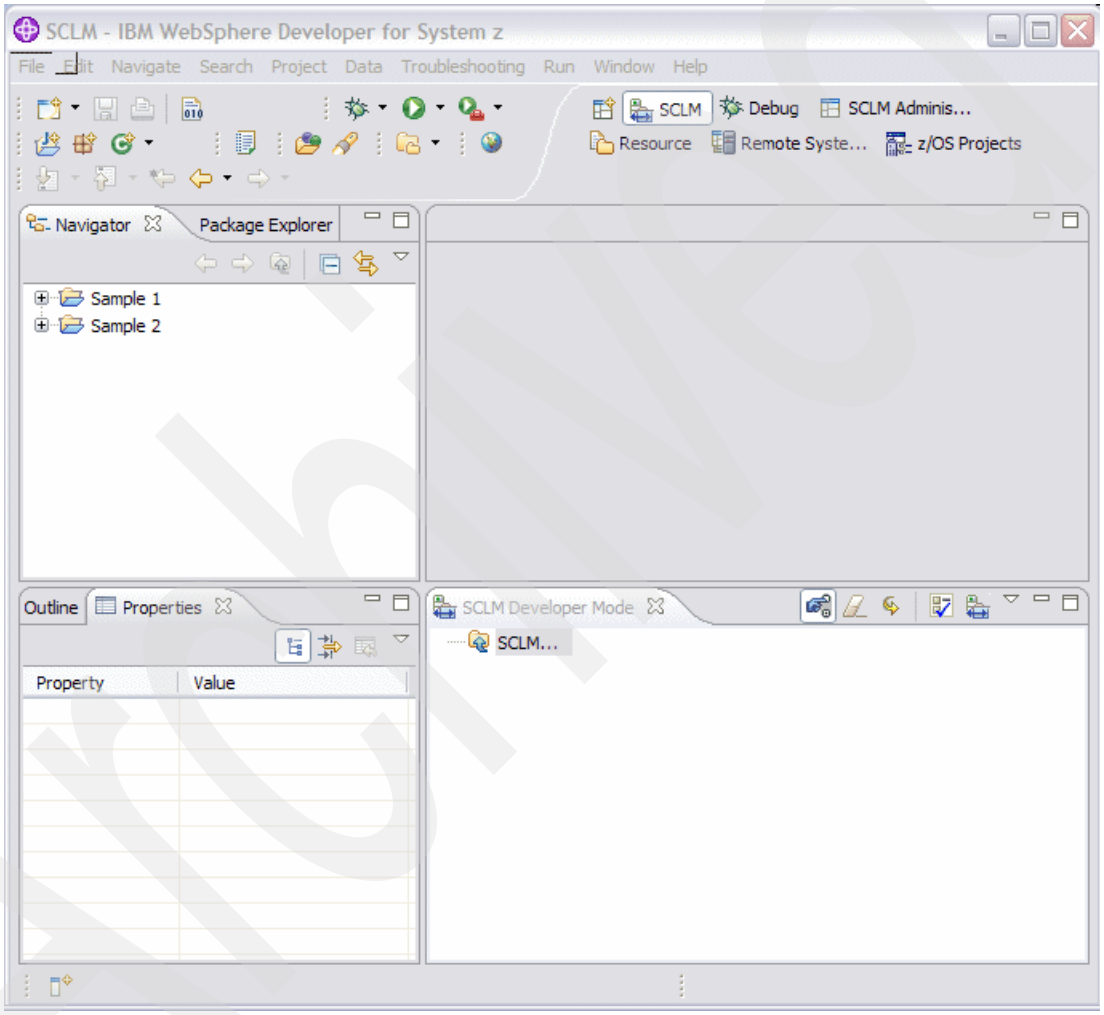


Figure 20-1 Initial SCLM Developer Toolkit panel

20.1.1 Before you begin

Before we start looking at using Developer Toolkit to work with your z/OS members, we should draw attention to the Operations Log. We looked at the options relating to this in the preferences section previously. Every operation performed by Developer Toolkit returns a log. You can suppress the log in most instances if the return code is zero, but the logs will still be stored locally on your workstation and can be viewed if required.

The operations log is available by clicking the **Mode** , or by right-clicking on any node and selecting the **Operations Log** option. The operations log dialog will be displayed where you see the entries stored by date. Expand the date node in the tree and you see there is a log entry for each operation that was performed. If the operation failed, then there is a red cross next to the operation. Information that is available in the operations log includes:

- SCLM messages and reports, such as build messages and reports
- Output from user exits, such as CCVFY or build and promote user exits
- SCLM Developer Toolkit connection information
- Trace information for each action that may be useful in debugging if there is a problem

The operations log can be seen in Figure 20-2.

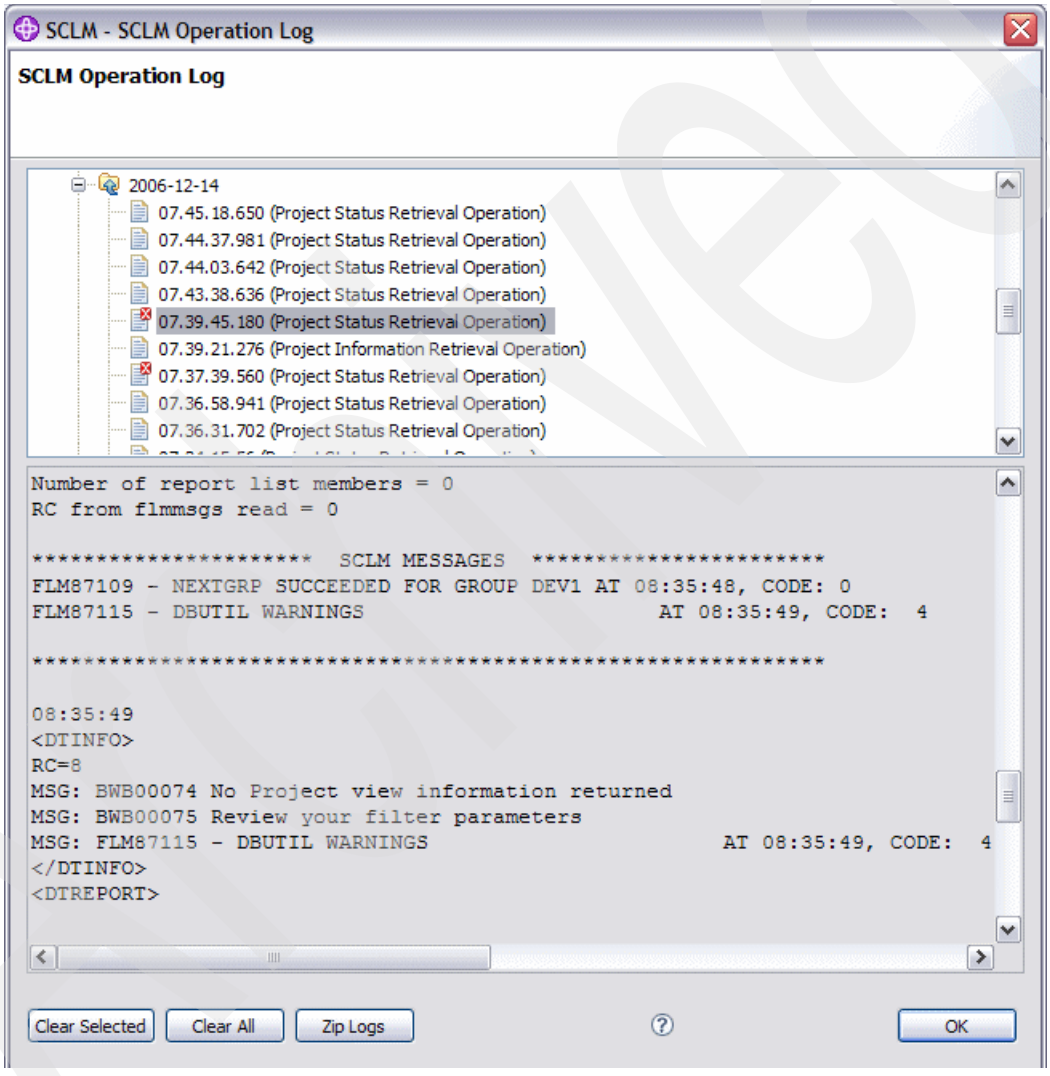


Figure 20-2 Operations Log

Additionally, you can zip up the logs, for example, to send to IBM if support is required.

20.1.2 Connecting to z/OS

Depending on the flavor of Eclipse you have installed SCLM Developer Toolkit into will vary the connection method used. If you are running WD/z then the connection protocol is Remote Systems Explorer (RSE) as provided with WD/z. Any other Eclipse environment that you

install into will activate the HTTP server connection. Start by double-clicking the **SCLM...** node or right-click and select **Add Project Filter to View.**

If you are not already connected, you will be presented with the connection dialog. If you are connecting via WD/z, this will include a list of already configured RSE connections as shown in Figure 20-3. Select the connection you want to use from the list, and enter the SCLM project you want to add to the Explorer View. If you have not logged in, you will be asked for your Mainframe userid and password.

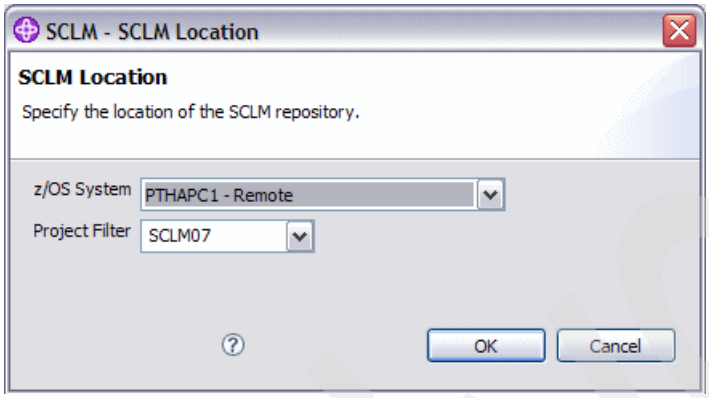


Figure 20-3 Specifying location of SCLM project through RSE connection

If you are connecting using another Eclipse such as RAD, you are prompted for the IP address and port number of the http server that Developer Toolkit is using. As with the RSE connection, you must also enter the SCLM project that you are interested in using. This location specification is shown in Figure 20-4.

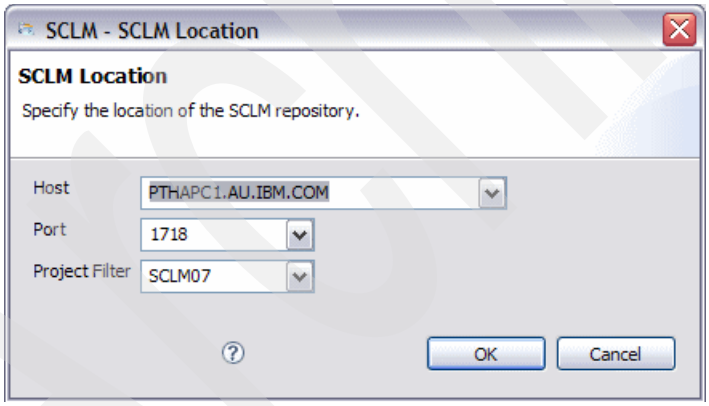


Figure 20-4 Specifying location of SCLM project through HTTP connection

20.1.3 Populating the explorer view

Once connected to z/OS, we can now populate the view. The SCLM Explorer view has had a node added to it for the project we specified as shown in Figure 20-5.

Figure 20-5 Populating the project node

Double-click the project node, SCLM07 in the above example, or right-click and select **Populate Project Filtered View** from the displayed context menu. The first time you work with a specific project or an alternate project, SCLM Developer Toolkit needs to go and collect information about the project. This is to store in cache in your workspace so that subsequent actions on that project are performed quicker. The cached information is used to propagate drop-down lists for example and includes information such as:

- Group, type, language, and authcode information from SCLM
- ASCII/EBCDIC codepage from TRANSLATE.conf or the site/project configuration files
- TRANLANG/LONGLANG information from TRANSLATE.conf or the site/project configuration files
- Other site and project specific configuration information from the site/project configuration files

So whenever a project new to you in this workspace is accessed, the following progress dialog shown in Figure 20-6 is seen.

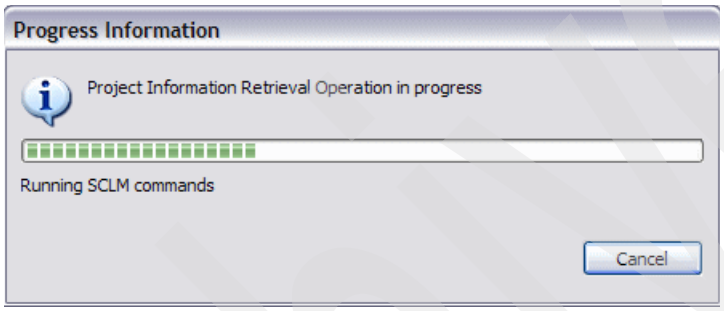


Figure 20-6 Project Information Retrieval in progress

Once project information is retrieved, the project information panel shown in Figure 20-7 is displayed for you to enter an alternate project if required. This panel already has the group pull-down populated for the base SCLM project. If you enter an alternate project, then project information will be retrieved from cache, or from the host if this is the first time you have referenced the alternate. On this panel you will also enter your default development group. This panel is very similar to the ISPF SCLM main panel.

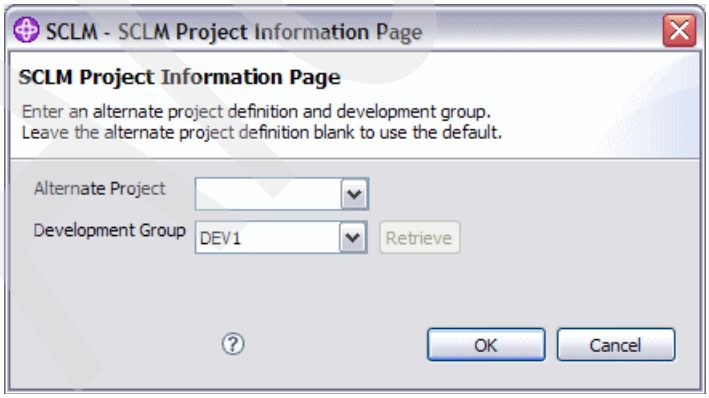


Figure 20-7 Project Information to enter your development group

Click **OK** and you are now shown the filters page. This panel is where we now decide what information we want to retrieve from SCLM. There are two ways to retrieve information from SCLM, either by entering certain filter values, or by populating by an ARCHDEF. Let us look at each of these population methods in turn.

Populating by filter

Use this option if you want to bring up a list of members based on certain filter criteria. You can use wild cards on all of the available fields. You can also use the retrieve buttons to retrieve type and language information from the cached project information. Figure 20-8 shows the filter dialog filtering on a type of COBOL and members beginning RDBK*.

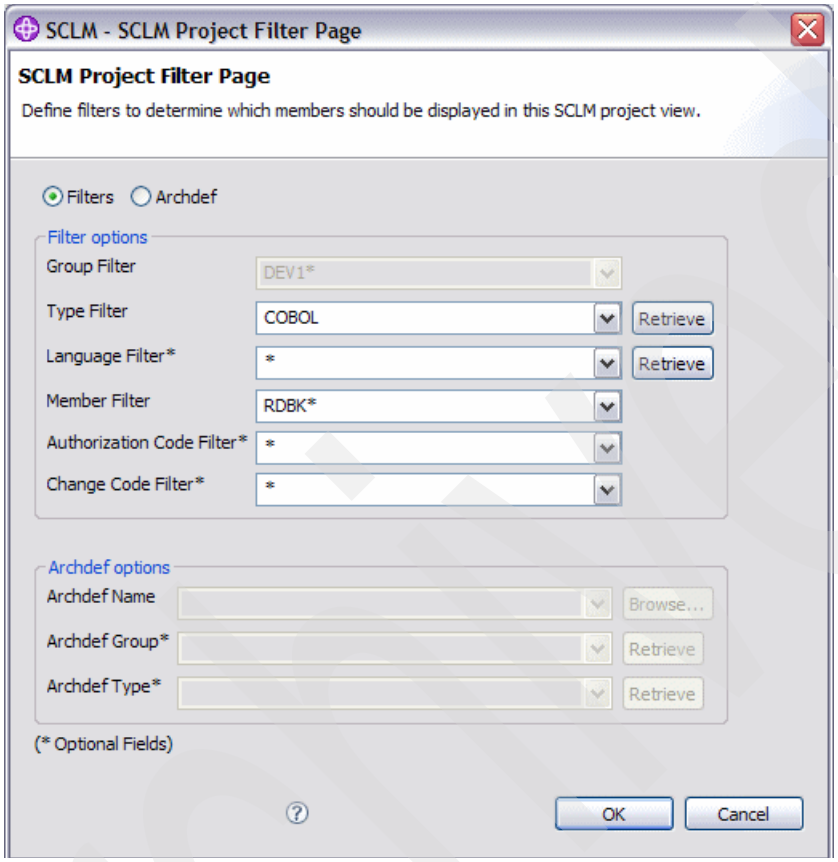


Figure 20-8 Project Filter panel

When **OK** is clicked, the request is sent to Developer Toolkit back-end services where the list of members matching the criteria is retrieved and returned to the IDE. The SCLM Explorer view is then populated with the list of members. Depending on what mode you have chosen, Explorer or Developer (which is the startup default), the member list will be displayed slightly differently.

For the example above, the member list returned is shown in Figure 20-9. The filters used are shown in the project note under the connection node. Also the time the population was performed is shown. This is because this view is a snapshot at a point in time. It is possible that another developer may be editing, building and promoting members included in your filter scope. If this is the case it is good practice to refresh the list occasionally. The results shown below are shown in Developer mode, so there is a type node, in the example showing COBOL. Below the type node, the members are shown, showing the member name, the member's language, and the group where the member resides.

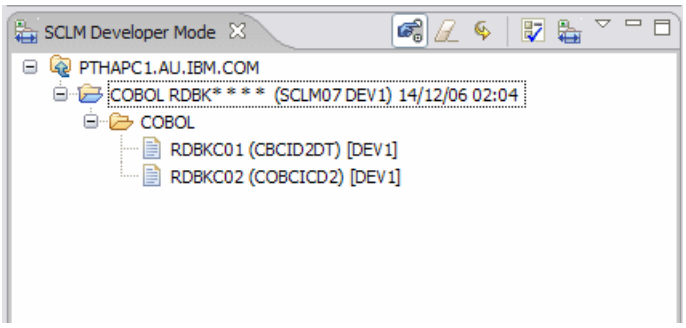


Figure 20-9 Resulting member list from populating by filter

Populating by ARCHDEF

Alternatively the SCLM Explorer view can be populated by specifying an ARCHDEF. This feature will read through the ARCHDEF and the SCLM build map, if available, to provide a list of all members that are included in the ARCHDEF. The **ARCHDEF** radio button at the top of the filter panel is selected and this will activate the ARCHDEF options, and deactivate the normal filter options. If you know the name of the ARCHDEF member name you can enter it in the ARCHDEF name field. The group and type can be left blank as the group will default to your development group and the type to a type of ARCHDEF. You can change these if you want to display the contents of an ARCHDEF that resides higher in the hierarchy, or in a different SCLM type, such as PACKAGE. Populating by ARCHDEF is shown in Figure 20-10.

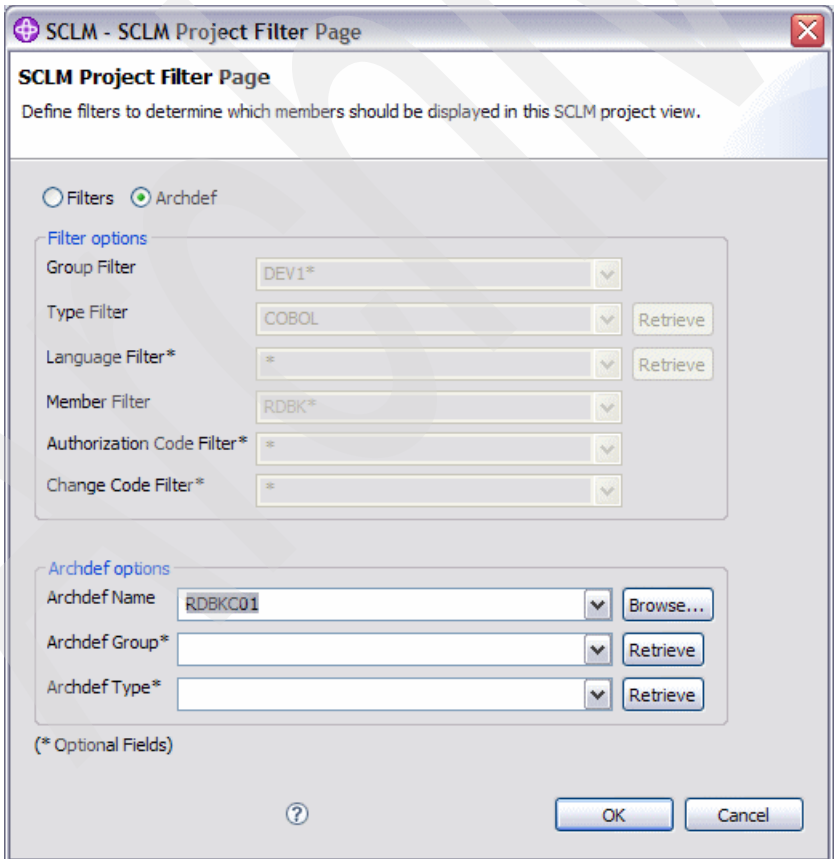


Figure 20-10 Populating by ARCHDEF RDBKC01

If you do not know the name of the ARCHDEF, you can click the **Browse** button and that will bring up an SCLM Member selection page. Click the **Refresh Members** button and an SCLM Project Filter Page is displayed. Enter the group, type and any other filter patterns that will help you find the ARCHDEF you are looking for. Click **OK** to begin the search. Figure 20-11 shows the filter panel used to find the ARCHDEF you are looking for.

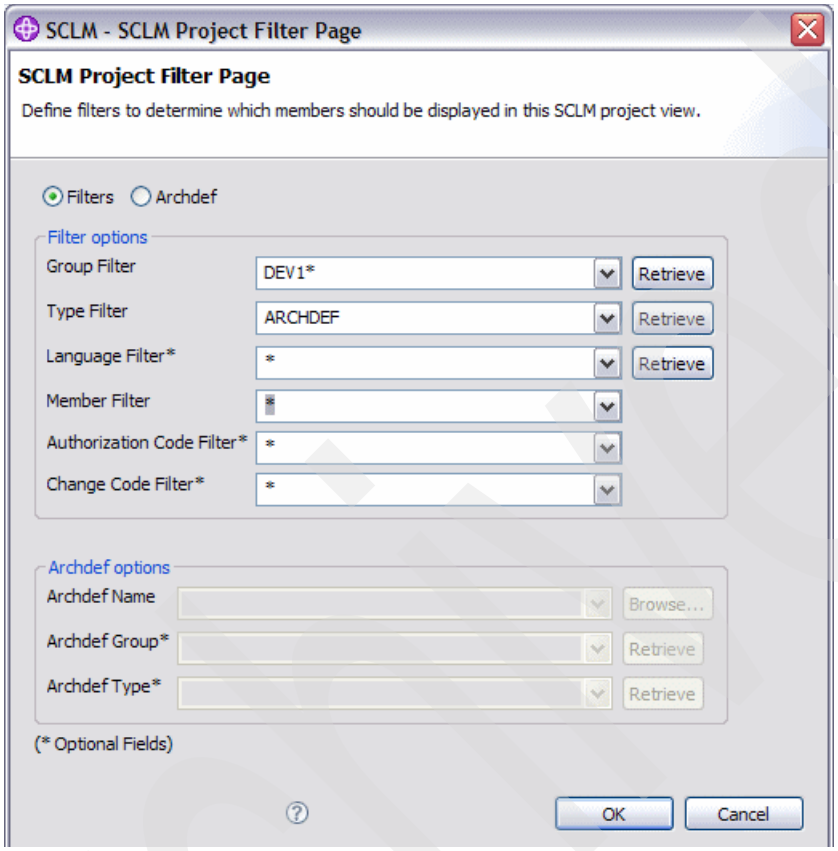


Figure 20-11 Filter panel used to search for ARCHDEFs

Once complete, the SCLM Member Selection Page will be populated with the groups and members that matched your criteria shown in Figure 20-12.

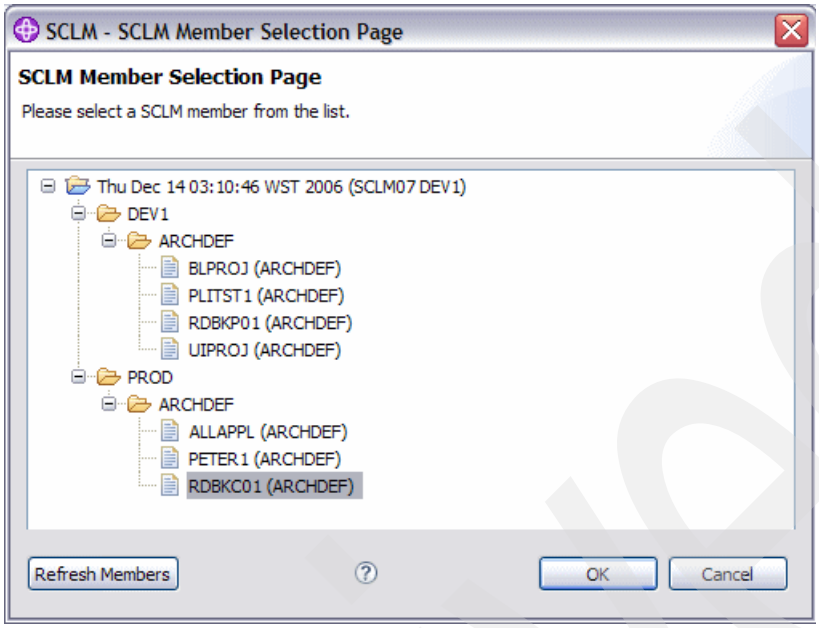


Figure 20-12 Member selection panel showing ARCHDEFs meeting filter criteria

By then selecting the ARCHDEF you are interested in and clicking the **OK** button, the ARCHDEF 0ptions on the original Filter page are now populated as shown in Figure 20-13.

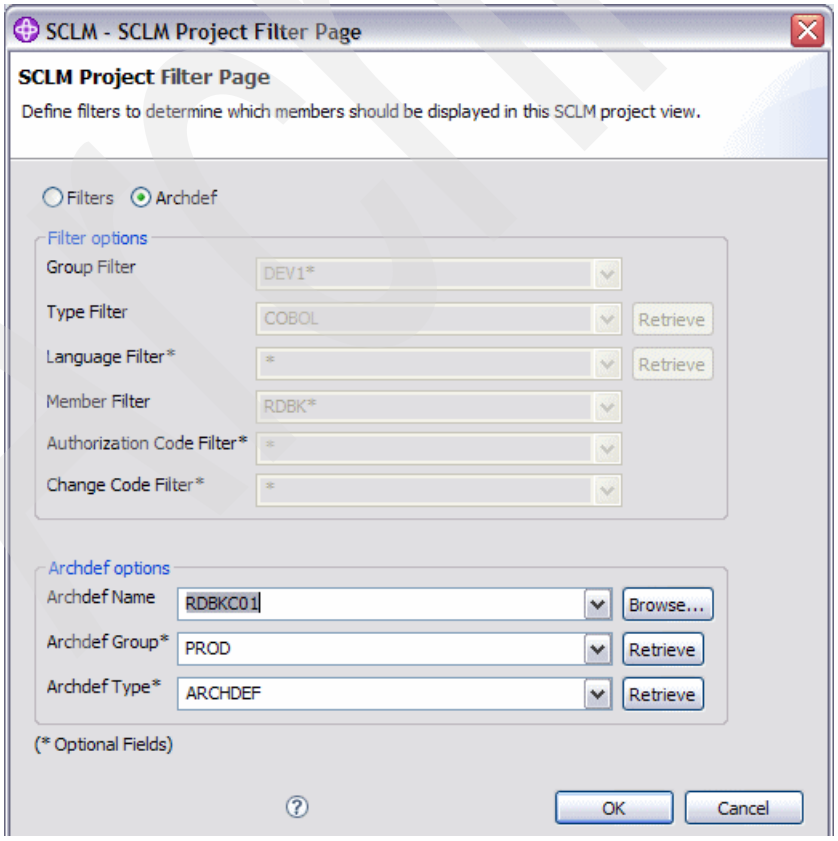


Figure 20-13 Populating by ARCHDEF after ARCHDEF has been searched for

When you click **OK**, the request is sent to SCLM and the list of members contained in your selected ARCHDEF is displayed. The information displayed is the same as populating by filter except all types and members that are included in the ARCHDEF are populated in the view. This is shown in Figure 20-14.

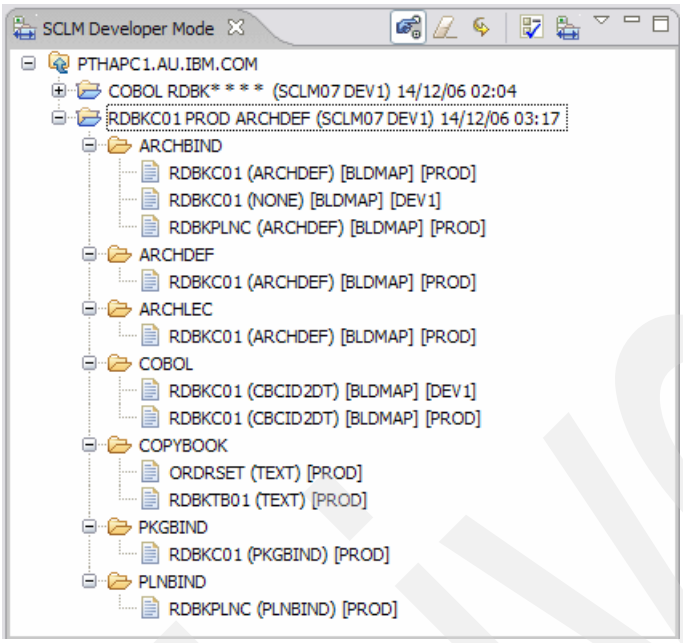


Figure 20-14 Resulting member list from populating by ARCHDEF

The example above shows a fairly typical scenario in that the ARCHDEF we needed to use is already at the PROD level in the hierarchy. However, when SCLM is populating, it looks from your development group, searching up through the hierarchy. The member list shown above has members from both DEV1 and PROD, as some of the components contained in this ARCHDEF are being worked on in DEV1 already.

Populating by ARCHDEF and then working on the populated list is similar to using the Unit of Work processing in SCLM (Option 3.11).

Viewing multiple projects in the view

It is possible to view multiple SCLM projects in the view, or multiple views of the same SCLM project. If you wanted to limit your view to just show the COBOL, COPYBOOK, and ARCHDEF types, then you can set up three views. You may want to make code changes to the COPYBOOK and COBOL library, but want to have the ARCHDEF in the view to enable you to build the high level ARCHDEF that contains the full scope of your application.

To add additional project nodes to the view, just double-click the connection node, or right-click at select the **Add Project Filter to View.** In the example above, this is the node that shows PTHAPC1.AU.IBM.COM. You can then enter filters for each of the additional projects, such that your project list is made up of a combination of filters and populated by ARCHDEF.

Figure 20-15 shows five different filters. There are separate project filter views for the SCLM07 project, listing types COBOL, COPYBOOK, and ARCHDEF that match certain criteria. There is a project view for the SCLM07 project that is populated by ARCHDEF. Also, there is a project view for the SCLM10 project, listing the ARCHDEF type.

Archived

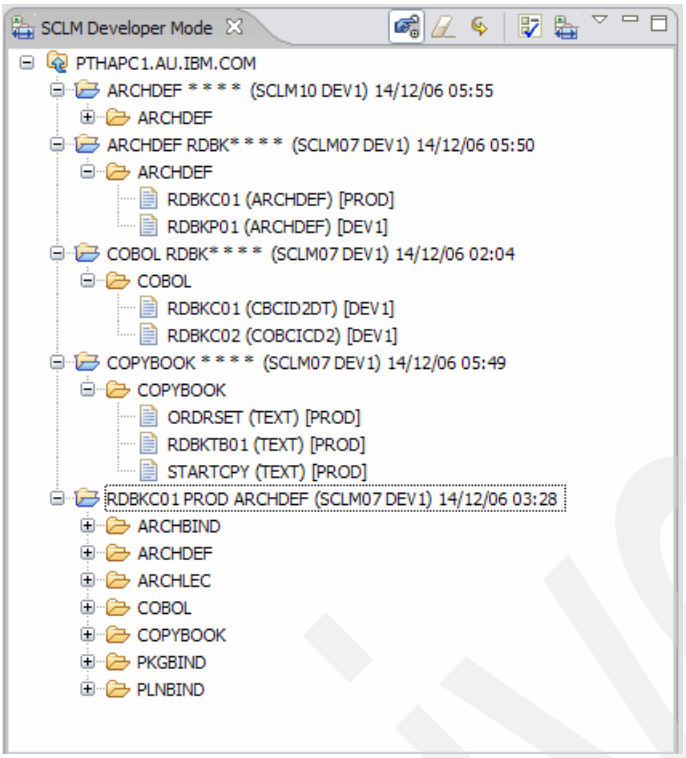


Figure 20-15 Listing multiple project views in the same View

Each of these views can then be refreshed independently of the rest by right-clicking the project line to show the context menu and selecting the **Refresh Project Filtered View** option.

Developer mode versus Explorer mode

There are two modes that the SCLM Explorer view can be displayed in, Developer mode and Explorer mode. The functions for SCLM are common across each of these views, but the group, type, and member information is shown differently across the two modes. You can switch between the two modes by clicking the **Switch View Mode** , or by right-clicking on any node in the tree and selecting the **Switch View Mode** option.

Developer mode

Developer Mode provides a view of SCLM members based on type, similar to SCLM Option 3.1. We saw from the previous examples that the nodes are separated by type. Then within each type the members are listed. Decorators show the group location of the member next to the member. As members are promoted through the hierarchy the decorator will change to show the new location. If you are using the SCLM view for the purpose of code development, building, and promoting, then we recommend that you use Developer Mode view. This provides the suite of functions that support this type of usage.

Explorer mode

Explorer Mode provides a full listing of SCLM members by group, type, and member, and is more suited for exploring the contents of the hierarchy. The display is different, as there is a group node now placed in the view, so the members are shown in the groups where they are found. Developer mode will always start looking up the hierarchy from your specified development group, this cannot be changed. Explorer mode allows you to search any groups individually, all groups, group patterns, or by hierarchical group search, depending on how the group is specified. Table 20-1 shows some examples.

Archived

Table 20-1 Explorer mode group values

Group Value	Description
DEV1	Displays all members matching the filter from the DEV1 group
DEV*	Displays all members matching the filter from groups that begin withe the name DEV
DEV1*	Displays the first found member in the hierarchy starting at the DEV1 group and looking up. This is because SCLM knows that the part of the group name before the wildcard is actually also a group
*	Displays all members matching the filter from all groups in the project

The results, using the above values and searching for member BILLINGP, can be seen in Figure 20-16.

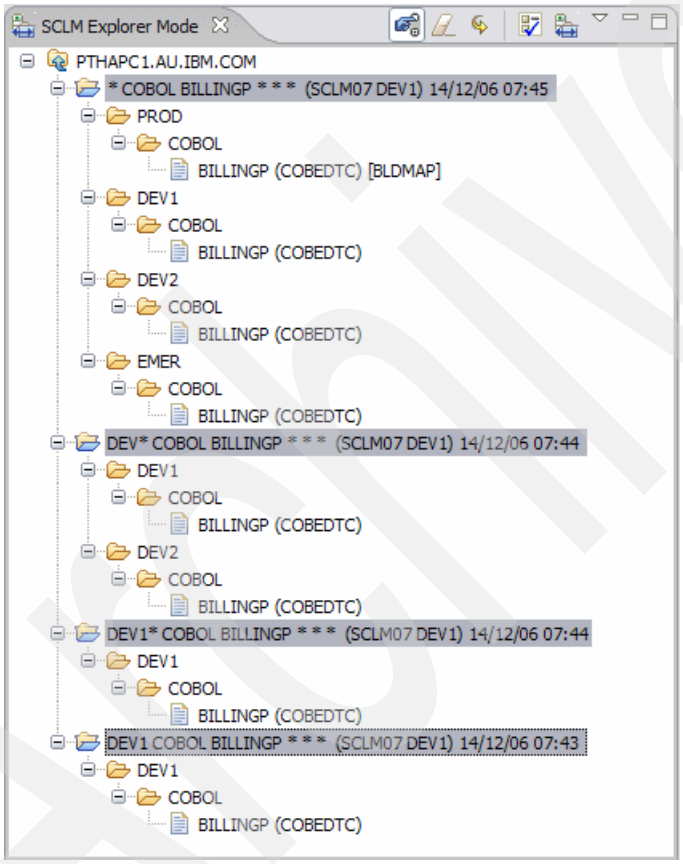


Figure 20-16 Explorer mode using different group specifications

20.1.4 Editing members

Now that you have populated the view, you can begin to work with the members. Previously, we looked at associating the SCLM language with a workstation extension so as to utilize the power of different editors. The Edit/Check-out function provides an SCLM check-out service for the selected member. The SCLM member contents are copied to the workstation and are edited locally. However, the member is locked in SCLM to stop other users from editing it. Check-in copies the edited code back to SCLM and releases the lock on the member.

Archived

Double-clicking a member will open the member in a browse mode. This means that the member is not locked in SCLM, so it is not checked out.

To begin editing, right-click to bring up the context menu and select the **Edit / Check out** option. Which editor opens in the editor pane will depend on what language extension mapping you have set. Also, you can maximize the editor pane so that you are editing in full panel, thus giving you more real estate to work with. Without any language extension mapping, the default text editor will open. Editors like LPEX will color-code based on language syntax, to provide visual aids while you are coding. The editor panel can be seen in Figure 20-17.

Note: If you are using the LPEX editor that comes with WD/z and you are familiar with the feel of the ISPF editor, or maybe XEDIT, you can change the way the editor interacts to some extent. Go to **Window** → **Preferences** → **LPEX Editor** and change the Editor profile to one that suits the editor you are used to.

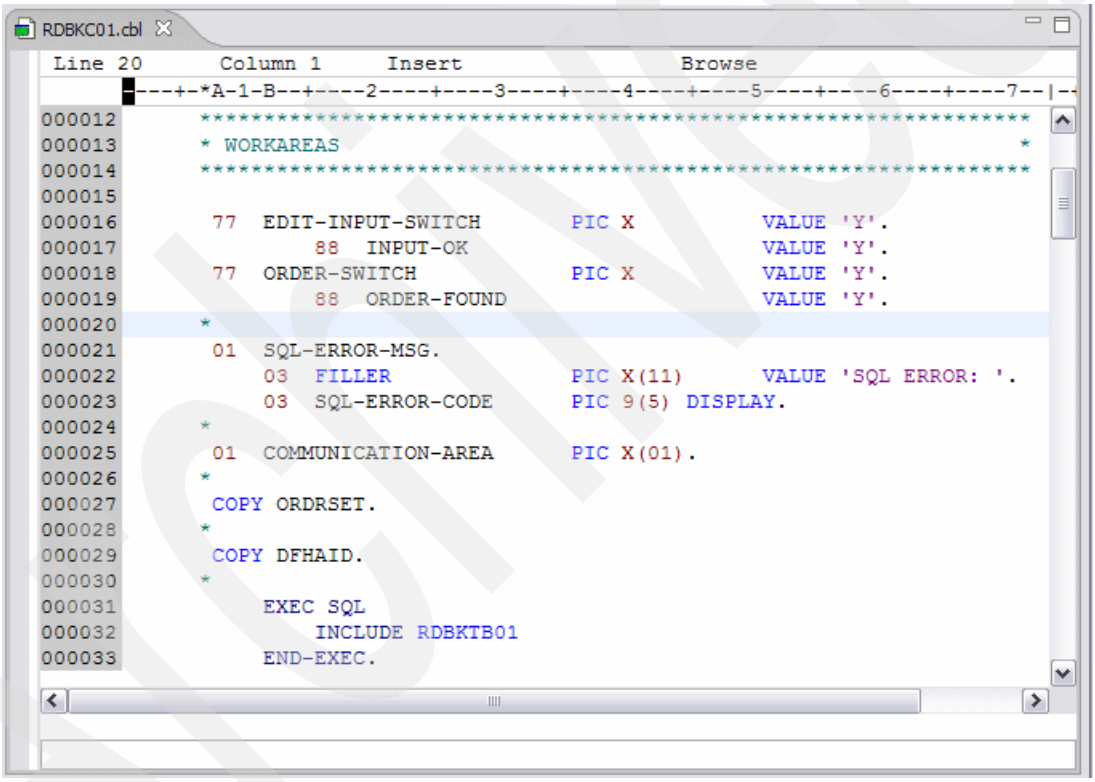


Figure 20-17 Editor pane, editing a COBOL program

Once you are finished editing the member, go back to the SCLM view and right-click the member name to bring up the context menu. Select the **Check In** option and the member will be copied back to SCLM on z/OS, saved, and the lock will be released. You can change the language or add a change code for the member at this time. Alternatively, if you do not want to save your changes, select the **Cancel Edit / Unlock** option from the context menu.

If your site uses any CCVFY or CCVER user exits, these will be called prior to the edit session beginning. If there are requirements on entering change codes, for example, then any messages sent from these exits will be written to the operations log.

SPROF processing

SPROF in ISPF is an SCLM Edit macro command, so this is not available via the IDE. If you need to change member details such as the language, or add a change code, there are two ways this can be done in Eclipse.

Right-click the member and select the **Update Member Details** option. The panel shown in Figure 20-18 will be displayed.

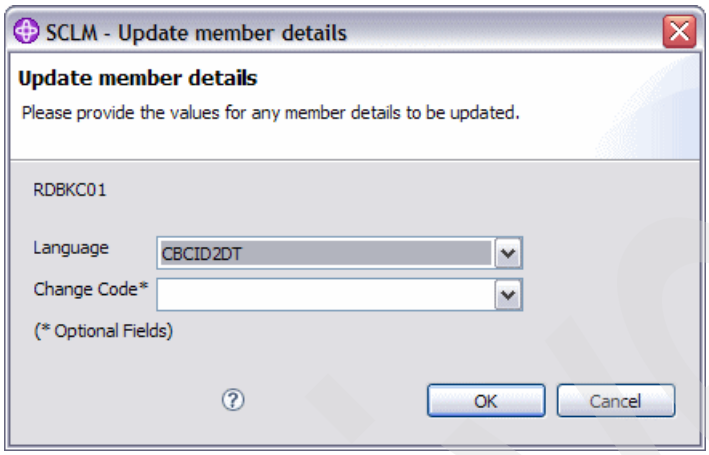


Figure 20-18 Update Member Details panel

You can select a different language from the pull-down list, for example, if you want to use a debug language. Also, you can add a change code.

The other way to change member details is to use the **Update/Reset Account Info** context menu item when you right-click the member. This dialog works slightly differently than the Update member details, as it allow you to select multiple members from the SCLM View to process the action against some or all of the members. By selecting multiple members from the list and then right-clicking against the select list, the panel shown in Figure 20-19 will be displayed. By selecting some or all of the members and then clicking the **Update Selected** button, you can then change the language for the selected members.

Archived

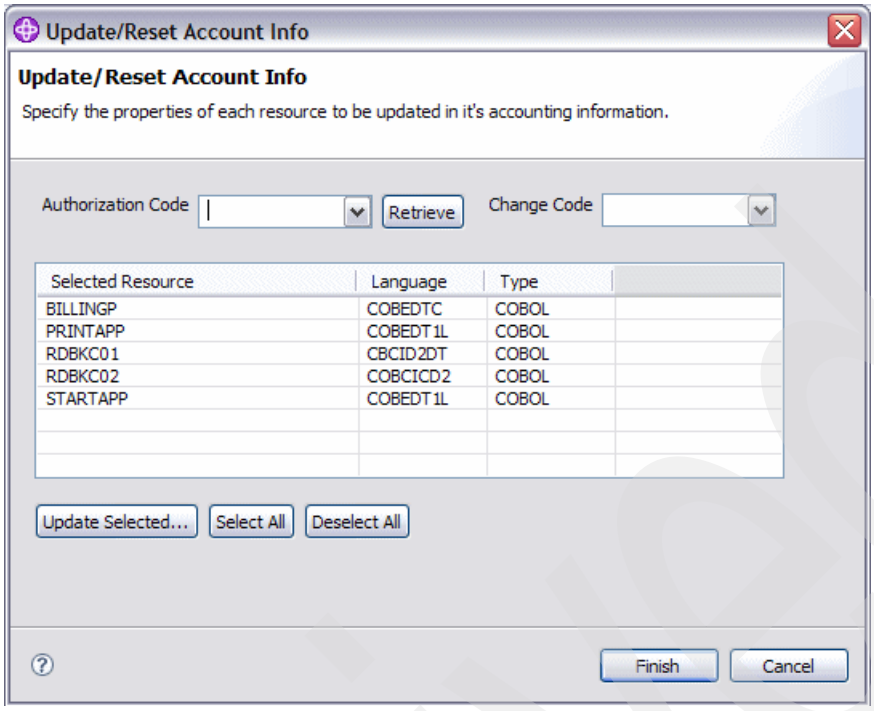


Figure 20-19 Update/Reset Account Info panel

Viewing accounting information

To see information that is stored about the member in SCLM, there are two options. When a member is populated into the member list in the SCLM view, certain member details are stored for that member. These can be viewed by just single-clicking the member to activate it as the selected member. Then look at the properties pane, by default, to the lower left side of the Eclipse window. The properties pane for member RDBKC01 can be seen in Figure 20-20.

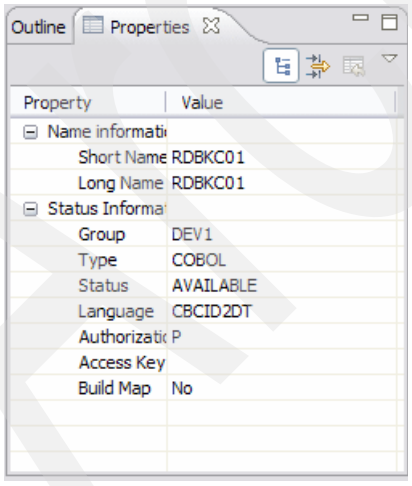


Figure 20-20 Member properties pane

You can get additional information on the member by right-clicking on the member name and selecting the **View / Refresh SCLM Status** option in the displayed context menu. This option will retrieve all of the account information for the member, including the Include lists, Change code lists, and the build map, if it exists. The full accounting information returned for member RDBKC01 can be seen in Figure 20-21.

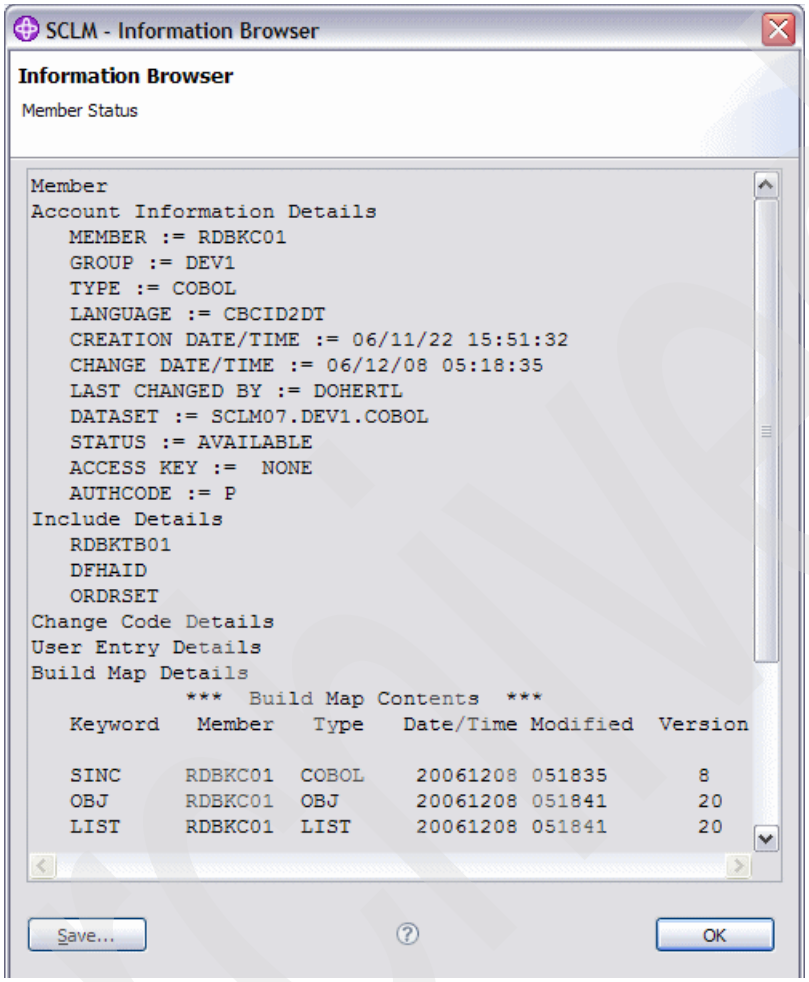


Figure 20-21 Account and build map information for selected member

20.1.5 Building members

The build process in Developer Toolkit is basically the same as if you were running a build through the SCLM ISPF interface on z/OS. The same options are available in the IDE as are available in z/OS. The build services provided in Developer Toolkit can be invoked either on-line or in batch mode. In on-line mode, the build messages and report are sent directly back to the log. In batch mode, Developer Toolkit provides a batch job monitor to inform you when the job you have submitted ends.

To invoke the build of a member, source, or ARCHDEF, right-click the member to bring up the context menu and select the **Build** action. The panel shown in Figure 20-22 is displayed. Enter the required build options, or take the defaults, and click the **OK** button to invoke the build.

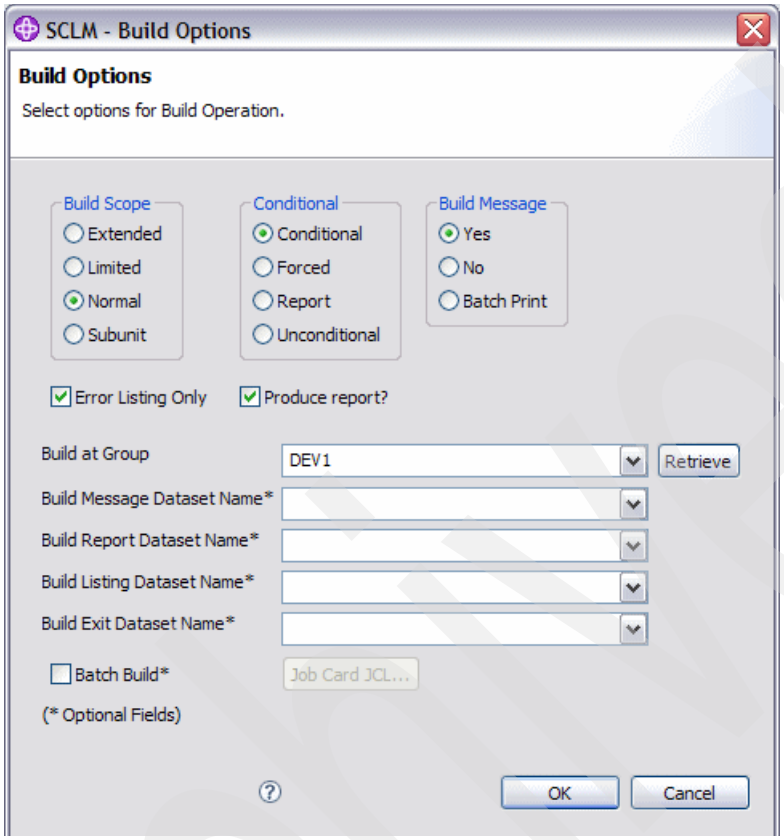


Figure 20-22 Build panel

If your project administrator has restricted on-line builds through the site/project configuration, as documented in 19.2.3, “Site and project configuration files” on page 455, then you will be informed through a message as shown in Figure 20-23 that you have to use a batch build.

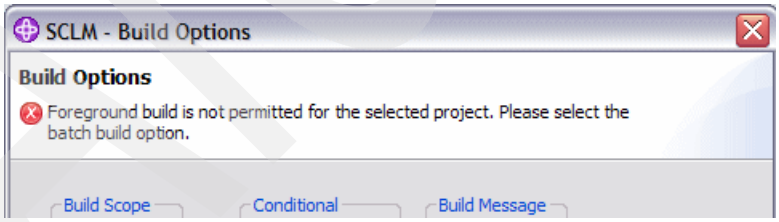


Figure 20-23 Foreground builds have been restricted through Site/Project config files

If you are running in on-line mode and you have selected the preference to always return the log on completion, you will see a completion dialog. If you click the **Details** button, the dialog box is expanded to show you build messages and a list as shown in Figure 20-24.

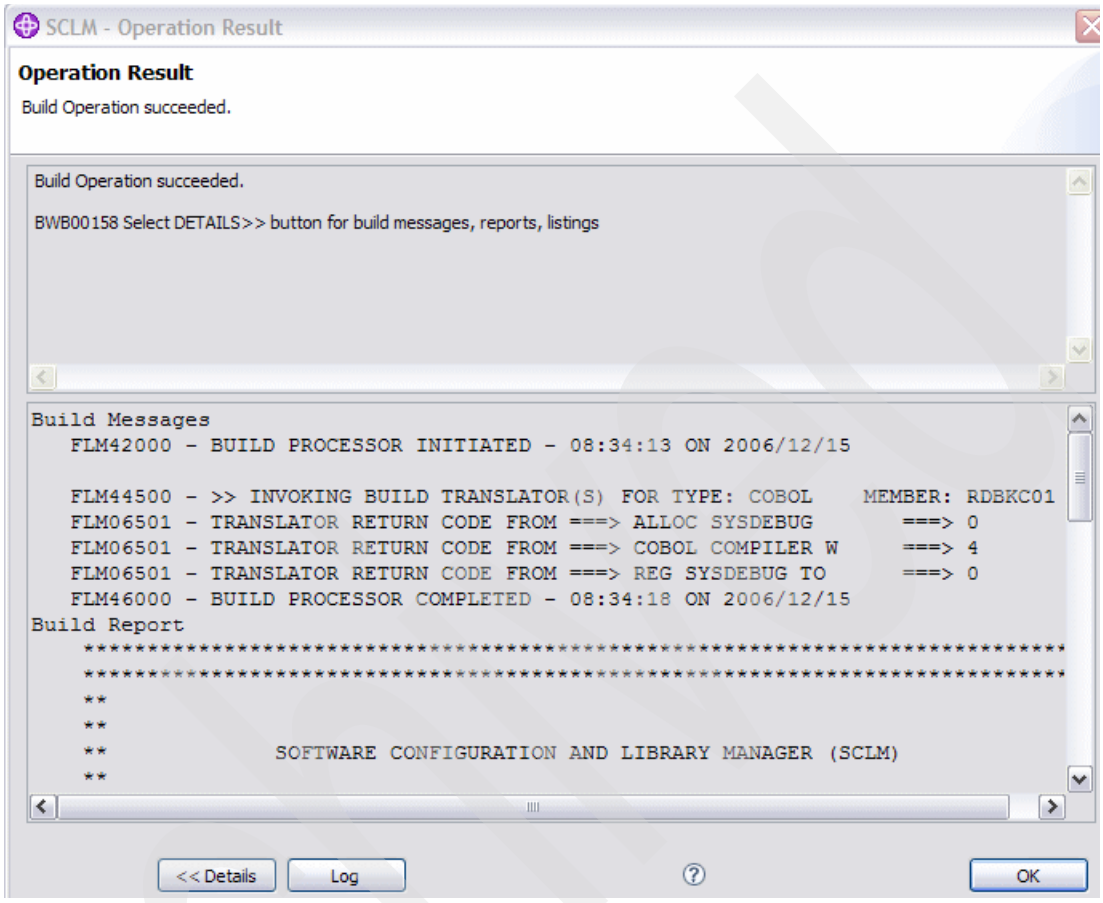


Figure 20-24 Build output

Invoking builds in batch

You can choose to invoke the build in batch mode by checking the Batch Build check box on the build panel. At this point you can enter your own jobcard. Alternatively, if your SCLM project administrator has enabled a build jobcard in the site or project configuration files, this jobcard will be used. To enter your own jobcard, click the **Jobcard JCL ...** button and you will be presented with the panel in Figure 20-25.

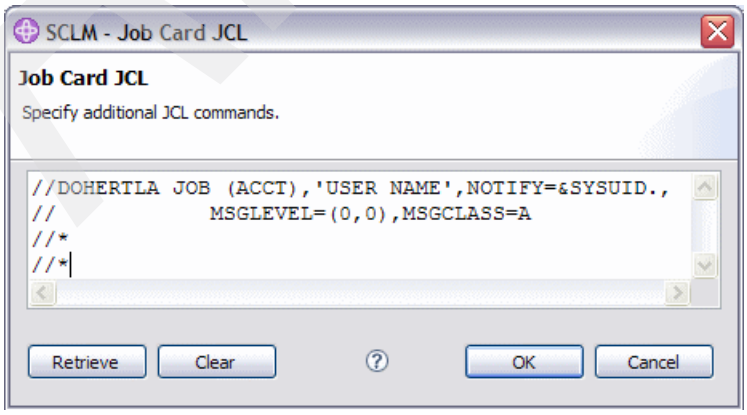


Figure 20-25 Entering your own jobcard parameters

Click **OK** to store the jobcard and then click **OK** on the build panel to invoke the build. You will be informed of the job number with which the job as been submitted, as shown in Figure 20-26.

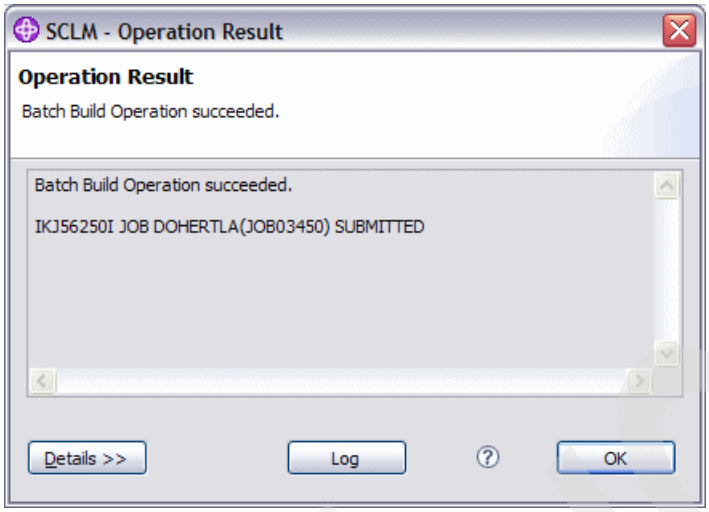


Figure 20-26 Submitting a batch job

If the batch build monitor is not active, then Developer Toolkit will ask you if you want to turn the monitor on, and you can optionally clear any outstanding build records. The batch monitor keeps a track of the job name and job ID in a table and periodically checks the jobs in the list to see if they have completed. It is possible that a record on this table can get orphaned, so clearing the list occasionally is a good idea. This can be done in **Window** → **Preferences** → **Team** → **SCLM Preferences**, then click the **View batch jobs** button. You can then click the **Remove** buttons to remove any old batch jobs.

Once the job finishes, a pop-up panel will appear informing you of this, as shown in Figure 20-27.

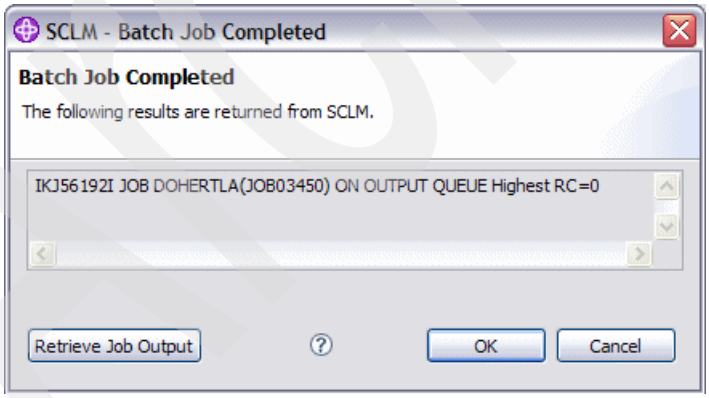


Figure 20-27 Job completion popup panel

By clicking the **Retrieve Job Output**, the job output is returned in a dialog box.

20.1.6 Promoting members

Just like build, the promote dialog offers the same options that the base SCLM option in ISPF offers. Additionally, the batch promote works in the same was as build. The promote panel can be seen in Figure 20-28.

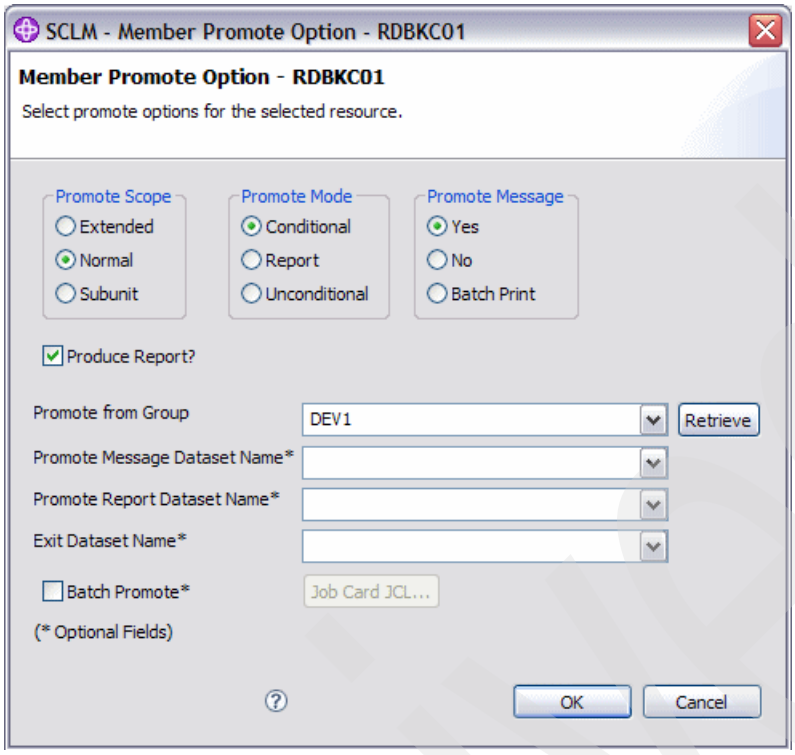


Figure 20-28 Promote panel

20.1.7 Configuring SCLM Developer Toolkit for Breeze use

If your site uses IBM Breeze for SCLM for package approval then your SCLM project administrator needs to have set the PROMOTEAPPROVER option to BREEZE in the site or project configuration files as discussed in 19.2.3, “Site and project configuration files” on page 455. This will activate Breeze specific code in the IDE during promote processing. If Breeze is active, then an additional promote panel will be displayed for you to enter the Breeze required parameters, as shown in Figure 20-29.

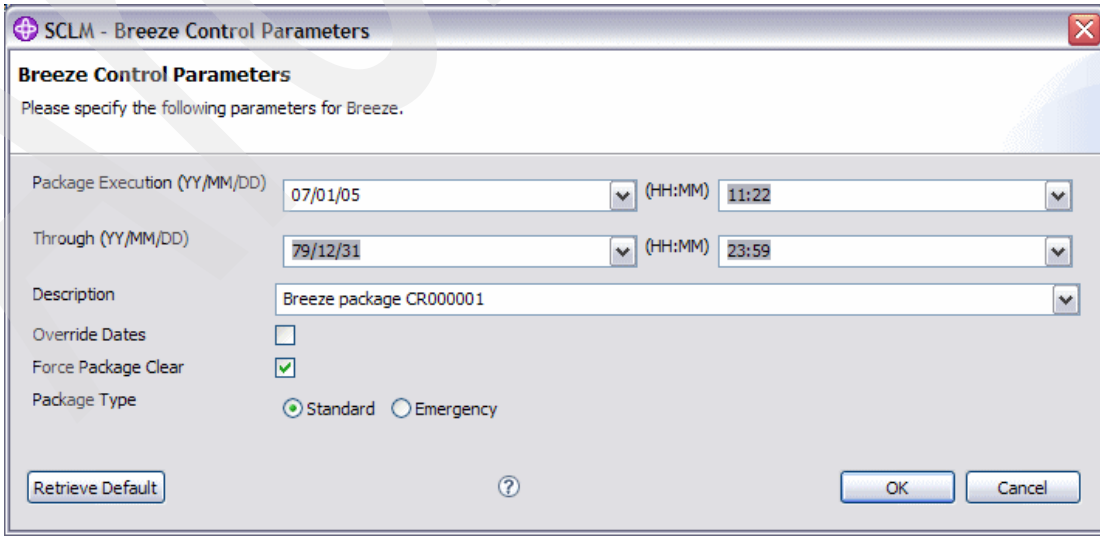


Figure 20-29 Breeze parameter panel

Additionally, any output that is written to SYSTSPRT, such as output from user exits, is also available in the log that is returned by SCLM Developer Toolkit. This is the case not just for promote, but for any SCLM operation that may have some SYSTSPRT output, such as an edit verification user exit. For example Figure 20-30 shows the messages written by the Breeze user exit as output in the log.

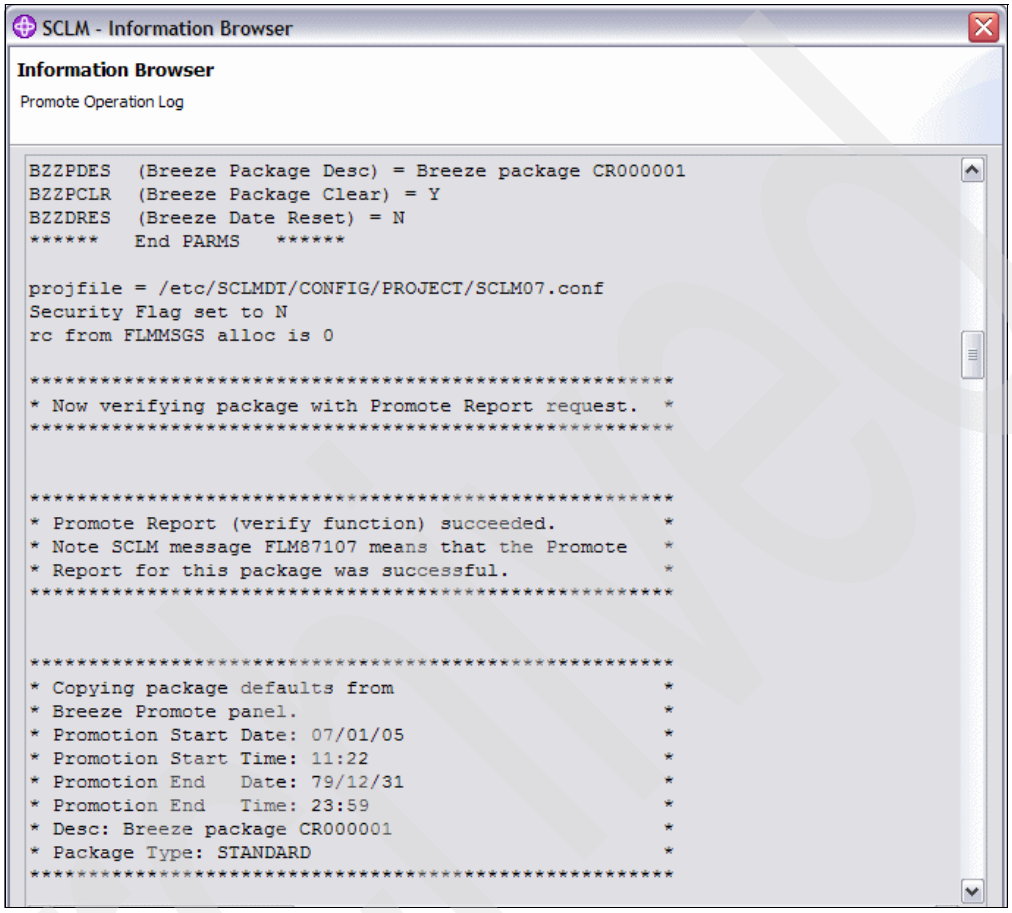


Figure 20-30 SCLM Developer Toolkit log showing SYSTSPRT output from Breeze user exit

20.1.8 Running database contents reports

The database contents report is the SCLM Developer Toolkit version of the Database Contents option in SCLM (Option 3.4). As the Developer Toolkit implementation of this option runs from the workstation, it is possible to utilize some of the features of the workstation and store the resultant report as a tab or comma delimited file with a suitable extension so that your preferred spreadsheet is opened to view the file.

The SCLM database contents parameters are the same as the base SCLM version, however there are some additional options in the IDE that make this utility more usable:

- The capability is provided to store the reports in the Eclipse workspace or in a directory on your workstation.
- The capability is provided to store the report generation options that were use to create a particular report. This enables you to re-run reports without having to enter all the parameters again.
- You can specify tab or comma delimiting of the information in the file.
- You can save the file with an extension that allows spreadsheets to be opened.

- Unlike the base SCLM version, which is limited to an FB 80 data set that wraps, the output file generated by Developer Toolkit is as wide as the data returned.
- Meaningful header names are used instead of the mainframe default of the @@FLM* variable names.

Prior to starting the Database contents report option, if you are going to store the reports in a project in the Eclipse IDE, you should create a suitable project. To do this, go to the navigator view. This is the pane that is normally in the top left of the panel. To create a new project, right-click anywhere in the pane to bring up the context menu and select **New** → **Project**. The new project wizard starts as shown in Figure 20-31.

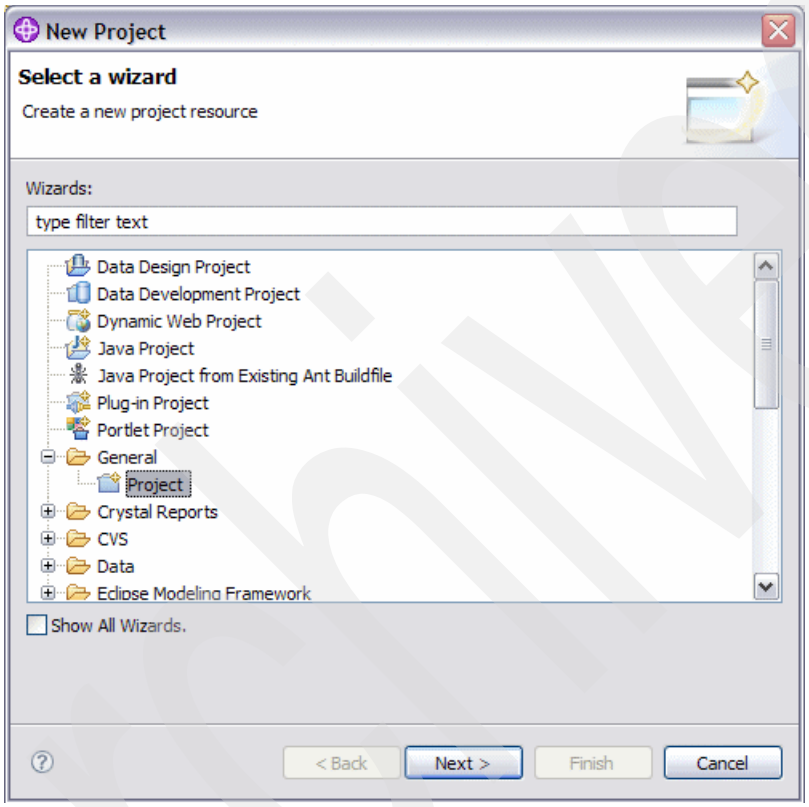


Figure 20-31 Create a new project to hold Database contents reports

Select **General** → **Project,** then click **Next**. On the next panel in the project name, enter a meaningful name, such as `Database Content Reports`. Then click **Finish**.

If you are going to be storing your reports as spreadsheet files, then you need to make sure that Eclipse knows to associate the file extension, .xls for example, with the Excel® program. Go to **Window** → **Preferences...** → **General** → **Editors** → **File Associations** and look in the list of file types for .xls (or whatever your spreadsheet extension is). If the one you want is not displayed, you can add it by clicking the **Add** box next to the File Types window. Highlight the language you have just added and click the **Add** button next to the Associated Editors.

The completed dialog for the *.xls extension is shown in Figure 20-32.

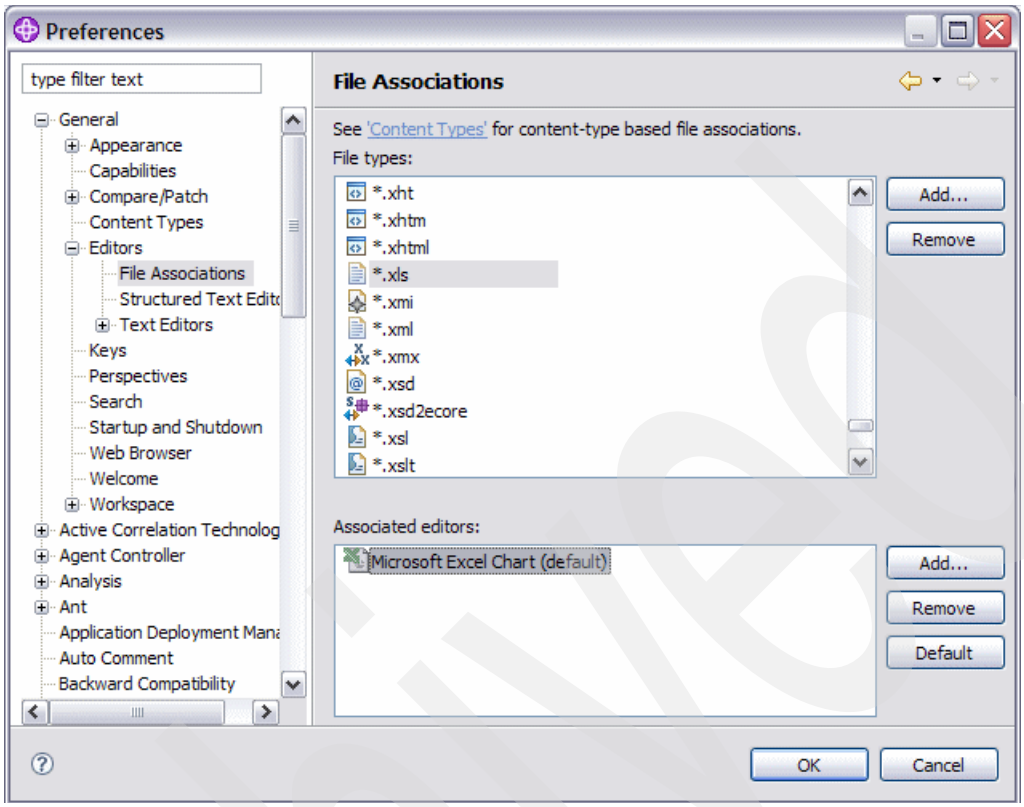


*Figure 20-32 Adding a file association for *.xls*

To start the database contents report, right-click any node underneath the connection node in the SCLM view, then select the **Database Contents Report** option. The database contents report wizard then starts.

Enter the name of the report, then select the radio button for an Eclipse Project, if that is where you are going to store it. Click the **Browse** button and select the Eclipse project where you want to store the report. If you are just creating the report on z/OS and not on your workstation, you can enter the data set name where the report will be stored. You can ignore this if you are storing the report in Eclipse. Finally, select the **Save report options** check box and browse to a location where you want to store the *.rpt files.

The first panel of the wizard is shown in Figure 20-33.

Click the **Next** button when you have reviewed what you have entered.

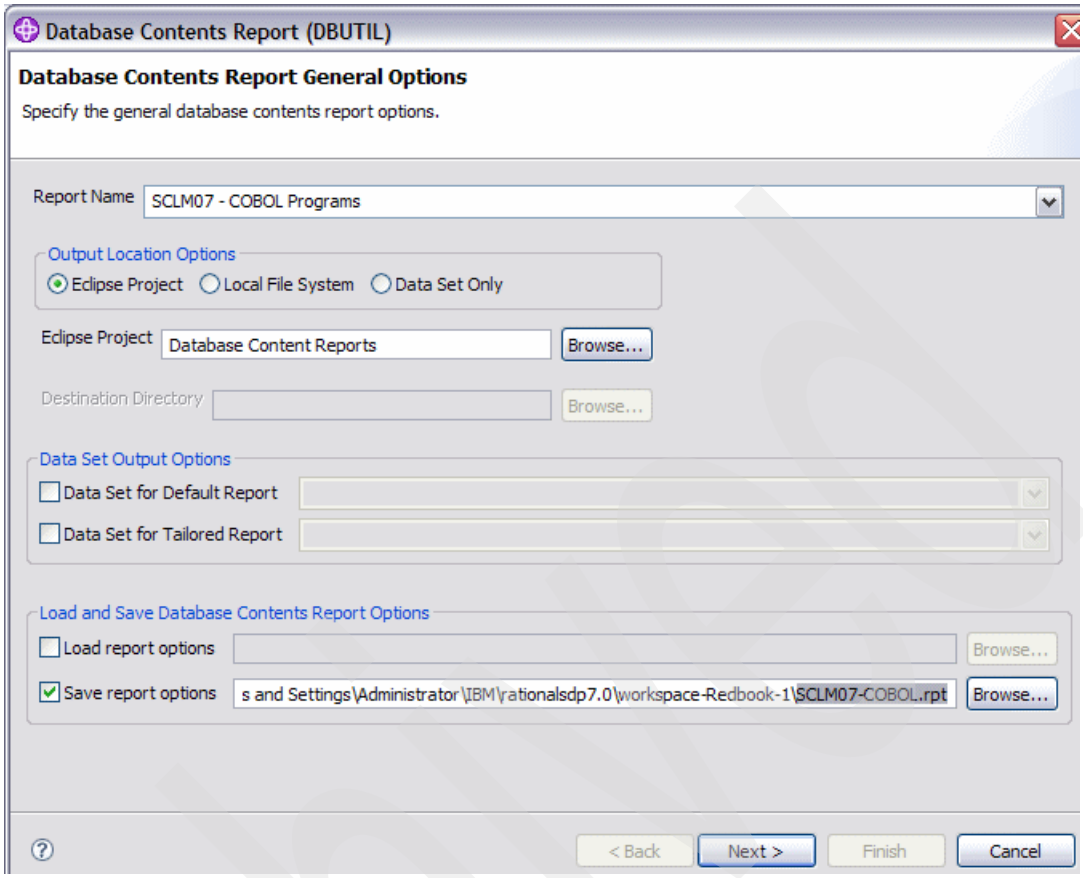


Figure 20-33 Database Contents Wizard - panel 1

The second panel contains parameters standard to the ISPF interface. Enter the groups you want to report on either individually in the boxes or enter an * in the Group 1 box for all groups. Enter the type you want to report on or * for all types, in our example we enter COBOL for the COBOL type. Finally enter an individual member, a member pattern or * for all members. Click **Next** to proceed to the third panel of the wizard.

On the third panel, enter any additional criteria you may want to limit the report by. For an explanation of these parameters, see the database contents section in the *Software Configuration and Library Manager (SCLM) Guide and Reference,* SC34-4817. However, the most common usage of these parameters is set as the default. The only parameter on the third panel that is Developer Toolkit specific is the **Use Long Header Name** parameter. This means that instead of the SCLM metavariable name that you select being used as the header name, as is the case on the ISPF interface, the more meaningful long header name will be used.

Click **Next** to proceed to the fourth panel of the wizard as shown in Figure 20-34.

On the fourth panel, select the column delimiting you want to use, blank, tab (/t) or comma, or enter your own delimiting character. Select also the file extension, such as .xls or .csv. There are two reports produced, the default report and the tailored report. The tailored report is the one that will be tab delimited and will contain a spreadsheet style report. You can choose not to produce the normal report if you do not require it. You can also include the headers on the tailored report and sum up any numeric fields, such as lines of code counted.

The main part of the tailored report is the variable that you select. These can be set up in two ways. One is to just select them from the list of variables. If you click the **Select...** button, a list of the most common variables is displayed as shown in Figure 20-34.

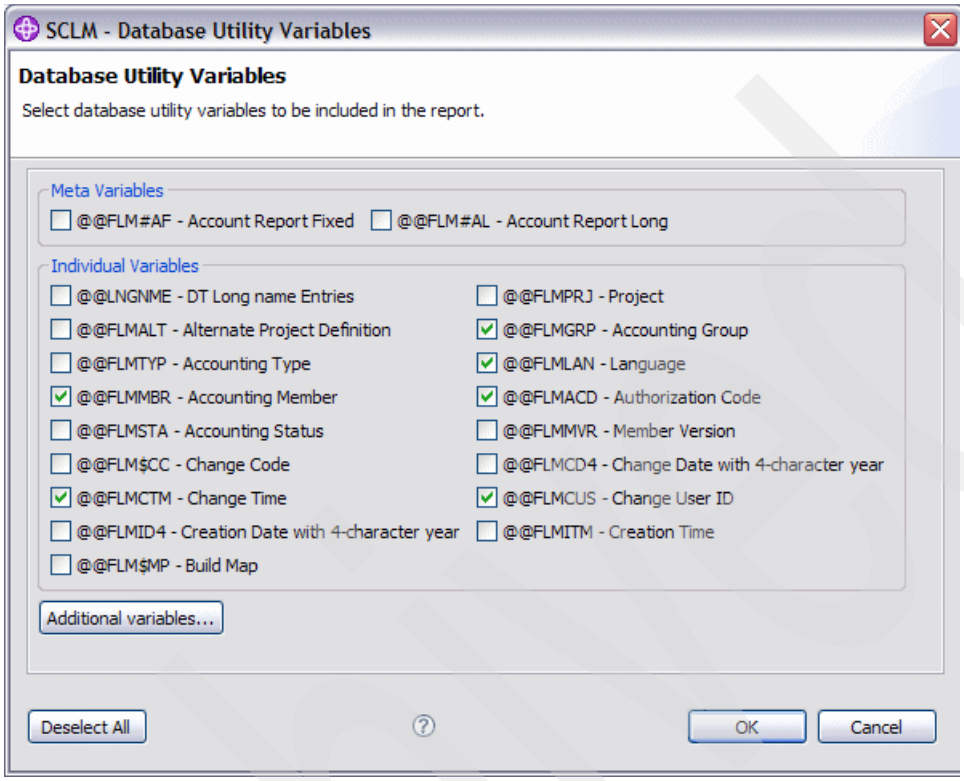


Figure 20-34 Database Contents Report - SCLM Metavariables

You can click the **Additional variables** button to get a full list of all the available variables.

By clicking **OK**, the variables selected will be added back to the tailored report contents area. You can just run the report like this, or if you prefer, you can change the order of these variables using cut and paste. To do this, click the **Free Format Text Entry** radio button, which will make this area editable. You can now cut and paste variables to move them around. You can also add your own text strings in between the variables. However, if you use the free format, the tab delimiting option will be overridden. So if you want to utilize the tab delimiting, then always select the variables.

The completed fourth panel of the wizard is shown in Figure 20-35.

Click **Finish** to run the report.

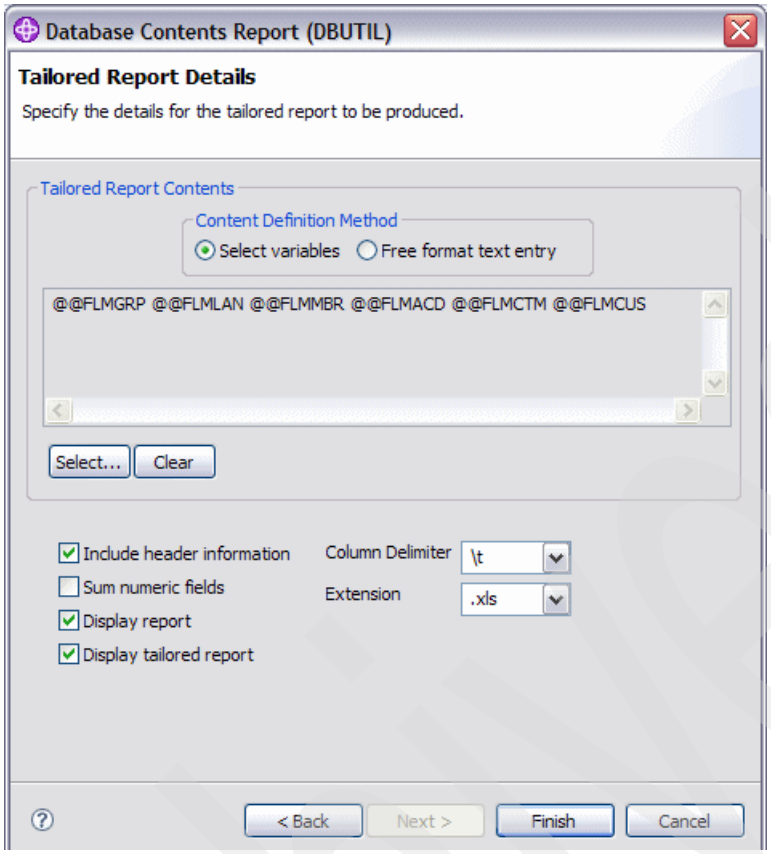


Figure 20-35 Database Contents Wizard - panel 4

When the report finishes running the spreadsheet application, in our example, Excel will start up automatically with the defaults it uses on startup. This means it starts with a proportional font. As this report came from z/OS, a non-proportional font is required to make it more readable. The following options should be performed on the spreadsheet. These instructions relate to Microsoft Excel:

1. Select all the data in the spreadsheet using Ctrl-A.
2. Change the font to Courier New or any other non-proportional font.
3. Select all the rows from the column headings to the last row.
4. From the menu bar, select **Format** → **Column** → **AutoFit Selection**. This makes the data in the columns more readable.
5. From the menu bar, select **File** → **Save As ...** And change **Save as type** to Microsoft Excel Workbook. Say **yes** to overwriting the file.

The spreadsheet after modification looks like the example shown in Figure 20-36.

36	SCLM07 - COBOL PROGRAMS					
37						
38	Accounting Group	Language	Accounting Member	Authorization Code	Change Time	Change
39	PROD	COBEDTC	BILLINGP	P	11:51:07	DOHERTI
40	DEV2	COBEDTC	BILLINGP	P	8:32:16	DOHERTI
41	DEV1	COBEDTC	BILLINGP	D	8:39:00	DOHERTI
42	PROD	COBE	PETER1	P	3:42:06	PCECK
43	PROD	COBDT	PRINTAPP	P	4:55:41	DOHERTI
44	DEV1	COBEDT1L	PRINTAPP	P	16:05:50	DOHERTI
45	TEST	CBCID2DT	RDBKC01	P	5:18:35	DOHERTI
46	PROD	CBCID2DT	RDBKC01	P	9:09:07	DOHERTI
47	DEV1	COBCICD2	RDBKC02	P	1:38:53	DOHERTI

Figure 20-36 Resultant tailored report

Once finished, you can close the spreadsheet. The workspace has also been updated. If you chose to produce the default report also, then that has been opened as a file in the editor pane. Also, the Eclipse project that you created now contains the tailored spreadsheet that was just created. This can be seen in Figure 20-37.

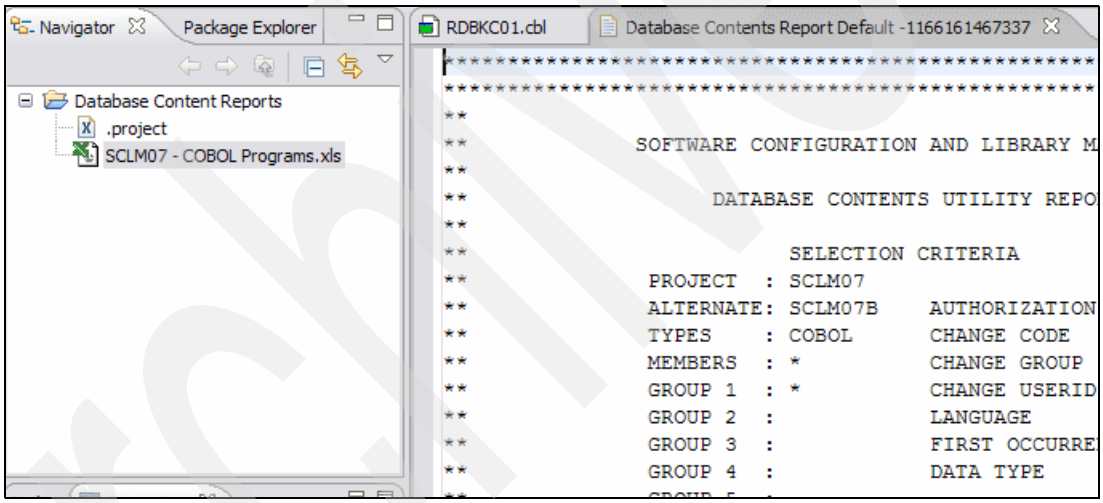


Figure 20-37 Workspace containing .xls file in the selected Eclipse project

Running the same report again

This is easy because you have saved the options that were used to generate the report in the first place. Right-click any node underneath the connection node in the SCLM view, then select the **Database Contents Report** option. The database contents report wizard then starts. On the first panel just check the Load report options check box and then browse for the .rpt file that you saved previously. The dialog should locate you on the directory automatically. This is shown in Figure 20-38.

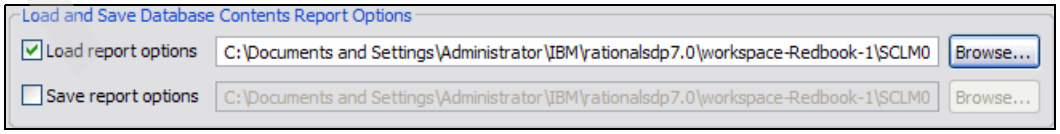


Figure 20-38 Using the Load report options to return a report

Click **Finish** to run the report straight away.

Archive

20.1.9 Audit and versioning options

SCLM Developer Toolkit provides all of the standard SCLM Audit and Versioning options for members that are in groups and types that have had versioning and auditing activated. The audit and versioning information for a want to recover versions for.

Listing versions

There are two options available:

- Version information only:

  From the member context menu, select **Retrieve Versioning Information**.

- Audit and version information:

  From the member context menu, select **Retrieve Audit and Versioning Information**.

The Audit information and Version information are differentiated by a different node icon as can be seen in the example shown in Figure 20-39. In this instance audit and versioning has been retrieved for member STARTAPP in the TEST group. The audit information is represented by the , icon and the versioning by the , icon.

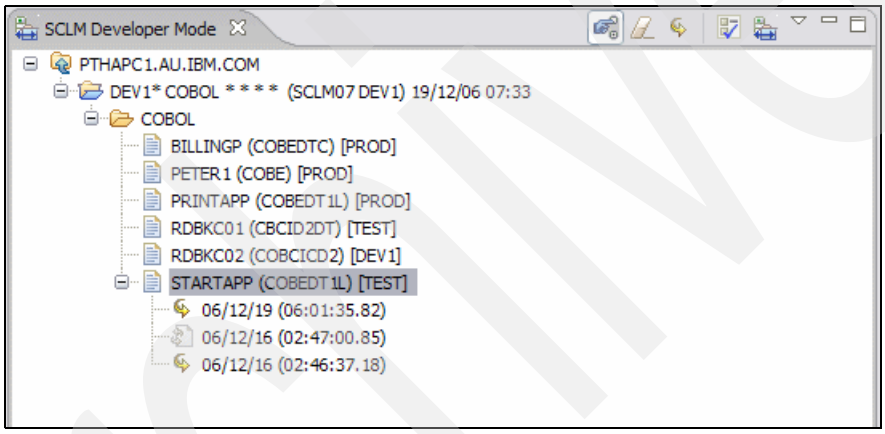


Figure 20-39 Audit and versioning retrieval for member STARTAPP

In the above example, we see that STARTAPP has two versions, with the version at the top of the list, the 06/12/19, being the current version. There is also an additional audit record.

Listing audit and version information

Now that we have a list of audit and version records, two new context menus will be available with the allowable actions for these records. The two context menus are shown in Figure 20-40.

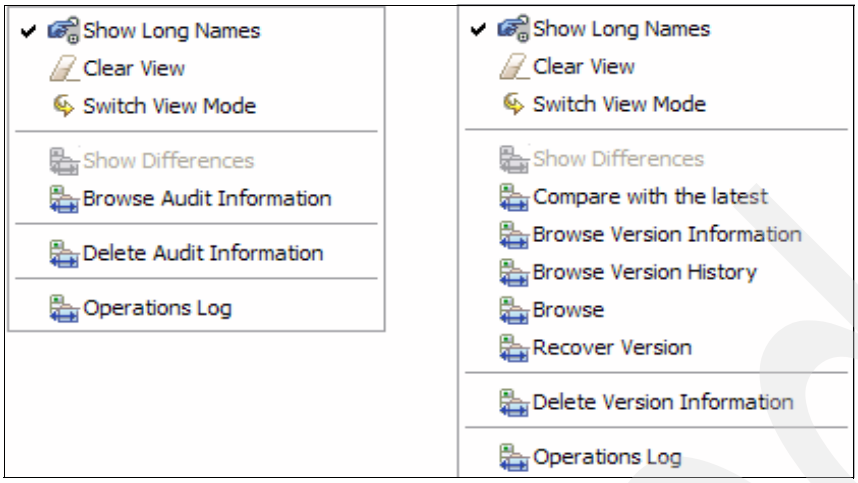


Figure 20-40 Audit context menu (left) and version context menu (right)

For Audit records, select the **Browse Audit Information** menu item from the Audit context menu and the audit information for the record will be retrieved from cache, as it was already stored when the audit and versions were listed, and displayed as shown in Figure 20-41.

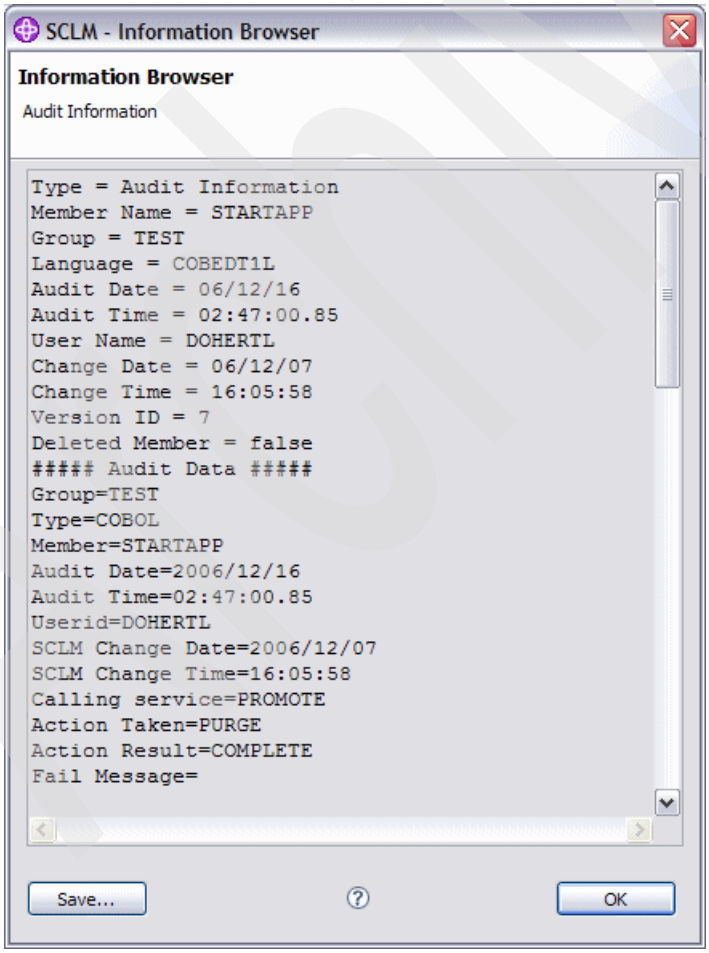


Figure 20-41 Audit information panel

Version records implicitly have audit information stored with the record, so selecting the version record by selecting **Display Version Information** from the version context menu will actually show the audit and version details for the record. The additional information shown for a version record is shown in Figure 20-42.

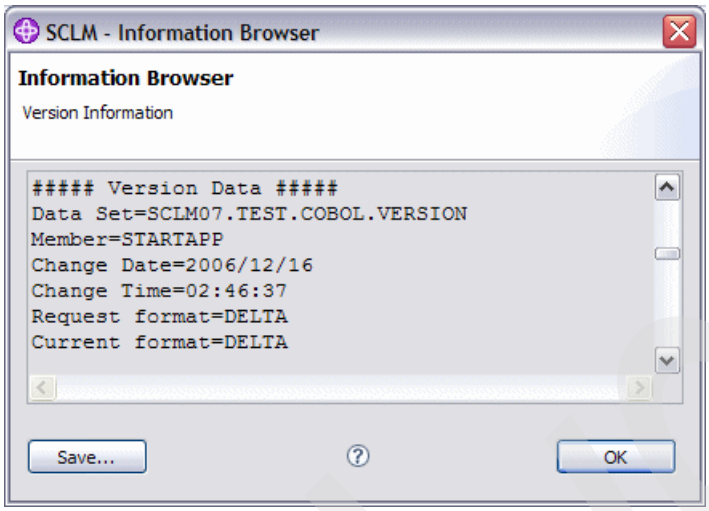


Figure 20-42 Additional version information shown

Comparing versions

There are two ways to invoke the comparison utility. The first is to select a single version and then select the **Compare with the Latest** context menu item in the versioning context menu. The other is to select two versions from the list of versions and select the **Show Differences** context menu option.

We can see that in both context menus shown in Figure 20-43, the **Show Differences** option is greyed out. This is because we need to select the two versions we want to compare from the list by using the Ctrl key, at which point the **Show Differences** option will become active. Once the versions required are selected, right-click to bring up the version context menu and select the **Show Differences** option. Or we can just select a single version and then select the **Compare with the Latest** option.

At this point SCLM Developer Toolkit retrieves both versions from SCLM and then uses the Eclipse compare facility to show the differences, as shown in Figure 20-43.

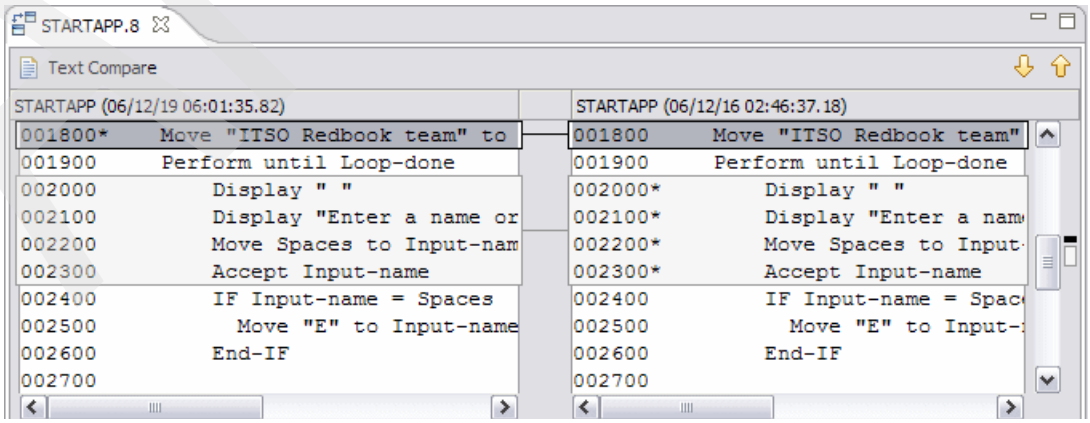


Figure 20-43 Comparing differences in versions using Eclipse compare

Browsing versions

You can browse any version you want by selecting the **Browse** context menu option from the version context menu. SCLM Developer Toolkit will retrieve the selected version and open an editor window for the version, shown in Figure 20-44.

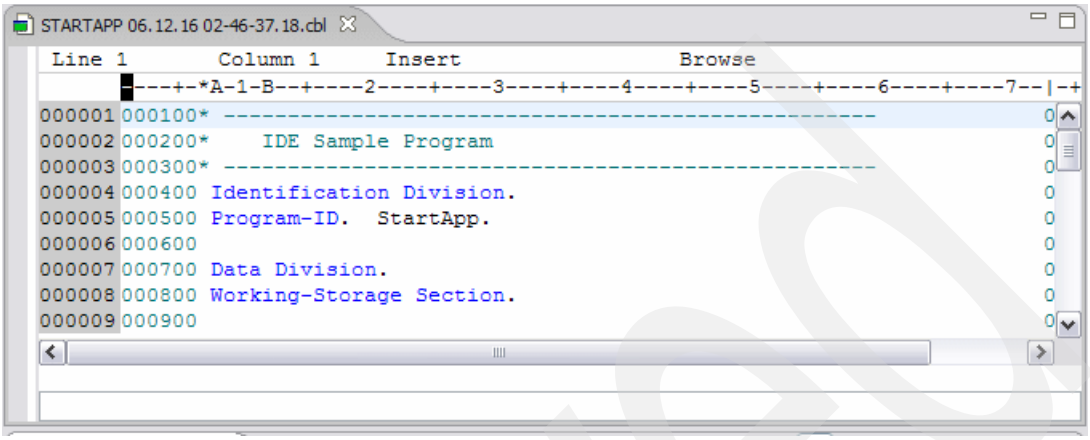


Figure 20-44 Browsing a different version

Browse version history

It is also possible to see the version history in the same format that is available in the base SCLM option on z/OS. To do this, select a particular version and right-click to bring up the version context menu, select the **Browse Version History** option. Once complete, the following file is displayed in the editor pane as shown in Figure 20-45.

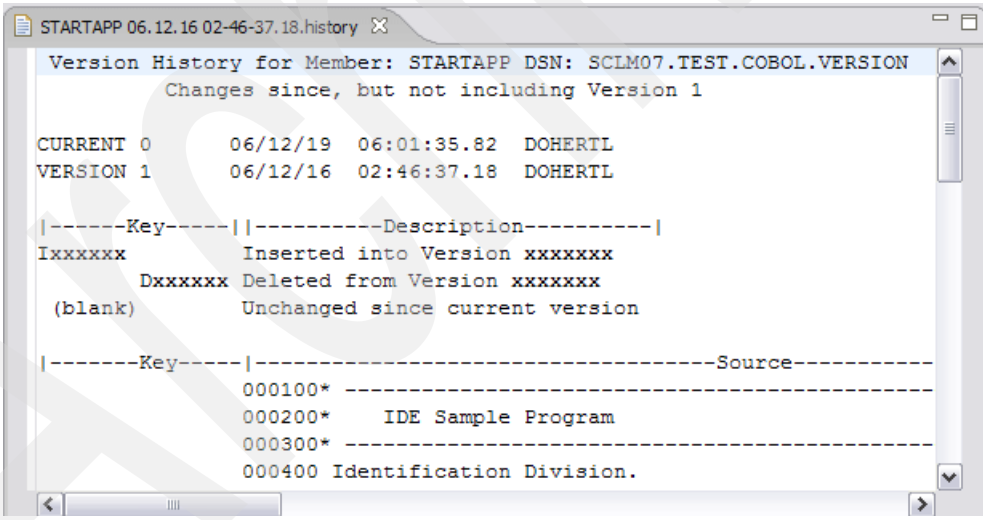


Figure 20-45 Browse version history

Listing deleted versions

There may be an occasion where a version of a module has been deleted from SCLM, but there are still audit and version records available for the deleted version. It is possible to list these deleted versions so that you can browse them and if necessary recover them.

We need to switch to the Explorer mode in order to recover deleted versions. So we populate the view as we would normally with our required filters. In the example shown in Figure 20-46, we have a program called STARTAPP that is at PROD, but there was a version at TEST that was deleted. We need to list and browse the version that was at TEST.

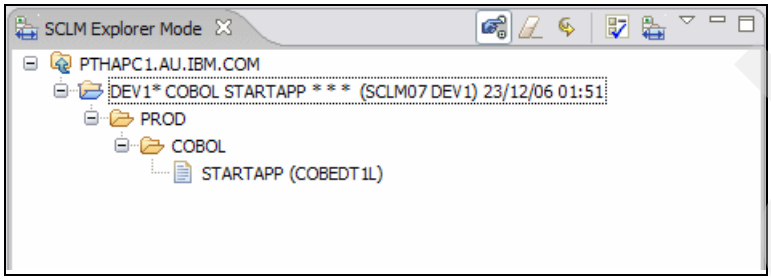


Figure 20-46 Populate View prior to retrieving deleted version

We only have versioning turned on at TEST in this project. However, there are no members that match out filters at TEST, so we need to add the TEST group to the view to enable us to invoke the version list. Of course if we had widened the scope of our filters and the TEST group was in the view, we would not need to do this.

To add the TEST group to the view, on the project node, right-click and select **Add Group to View**. The panel shown in Figure 20-47 is displayed, where you can select the group you want to add from the drop-down list. In our case we select the TEST group.

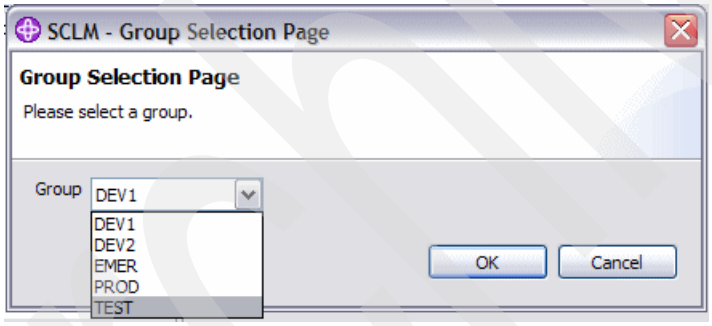


Figure 20-47 Selecting the group to add to the view

The group node is added to the list with no type nodes under it. So we now need to perform the same action for the type that we are interested in. So we right-click the TEST group we have just added and select **Add Type to View**. A similar panel as before is displayed where we select the type we are interested in, as shown in Figure 20-48.

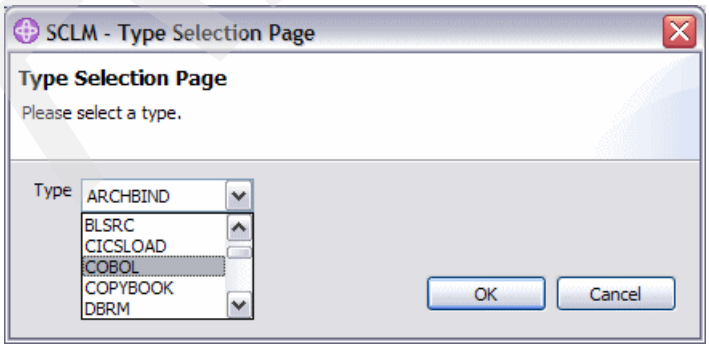


Figure 20-48 Selecting SCLM Type from a drop-down list

We now have the TEST group and COBOL type in the view, so now we can list the deleted versions. Right-click the COBOL type node under the TEST group node and select **Retrieve Deleted Member Information** as shown in Figure 20-49.

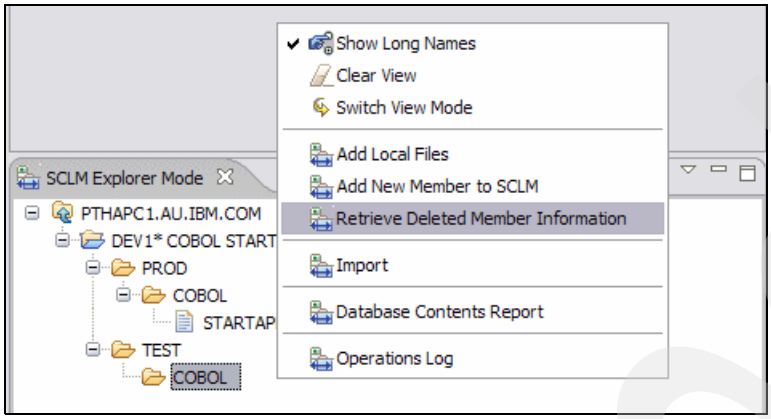


Figure 20-49 Retrieving deleted member information

SCLM Developer Toolkit will now retrieve both normal version records as well as deleted version records, but will add a member node for the deleted members. All members in the specified type that have a deleted version record will be listed under the type node. In our example there is only one member, STARTAPP shown in Figure 20-50. From here we can browse the version or recover the version using the available context menu options.

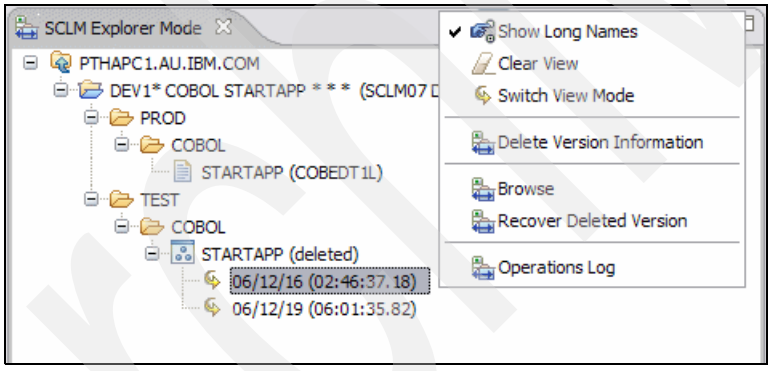


Figure 20-50 Listing deleted version information

Recovering versions

Once you have a list of versions, you can then recover any of those versions to your development group. If the version you want to recover is a deleted version, then select the **Recover Deleted Version** from the deleted version context menu, or if this is a non-deleted version, select the **Recover Version** option from the version context menu shown in Figure 20-51. Both options perform the same function. Developer Toolkit will restore the selected version into your development group and then refresh your view to show the development group, if it was not in the view, plus the recovered member.

Using the example above, we recover the 06/12/16 version of STARTAPP into our development group. The View after the action is complete is shown in Figure 20-51. If the member already exists in the development group, you will be asked whether you want to overwrite it, at which point you can cancel.

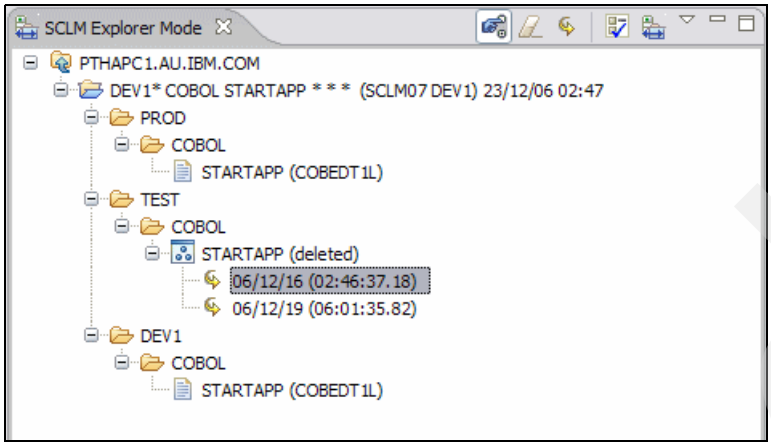


Figure 20-51 SCLM View after version has been recovered

20.1.10 Adding new members to SCLM

There are three ways to add members to SCLM from the SCLM view. These are:

- Adding a new member through the Eclipse editor
- Adding a local workstation file to SCLM
- Adding a non-SCLM PDS member to SCLM

Add new member by using the Eclipse editor

This option can be equated to using the editor in base SCLM on z/OS to type in a new member and then assign a language to it. The **Add New Member to SCLM** option is available from the Project node, Group node, and Type node. Depending on which context menu you select this option from, the amount of information that is filled in on the New member detail panel will vary. Let us assume that we have populated the view as shown previously in Figure 20-9 on page 469. If we want to create a new member called RDBKC03, then we can right-click the type node and select the **Add New Member to SCLM** option. Figure 20-52 shows the panel that is presented for us to enter member information.

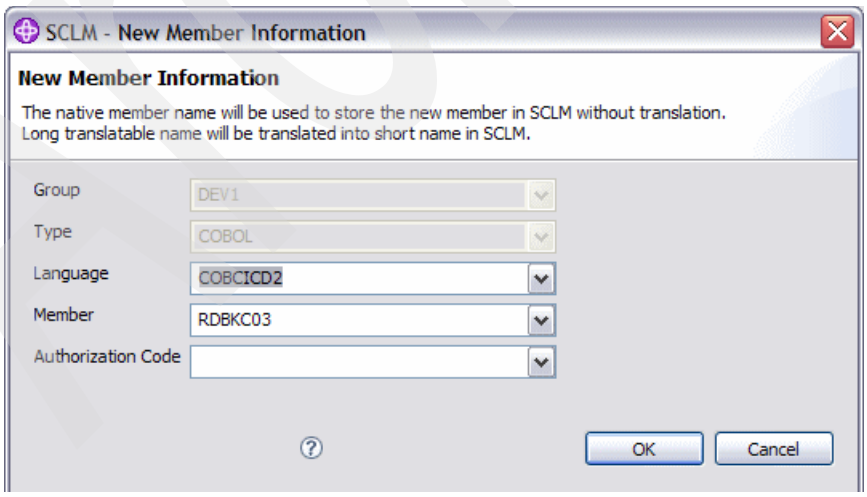


Figure 20-52 Member information panel

The group and type entry fields are greyed out as we invoked the add new member option from the type node. If we had invoked the add new member option from the project node, then the type entry fields would allow us to enter the required type for the member. The tree view would then be repopulated to add that type to the view. We also need to enter a language for the member and the member name. The language can be selected from the drop-down list. Optionally an authorization code can be entered, or left blank to take the default primary authcode. Once we click **OK** the member is locked in SCLM in an INITIAL state to stop another user from creating the same member, and the Editor pane is opened with the empty member. This can be seen in Figure 20-53.

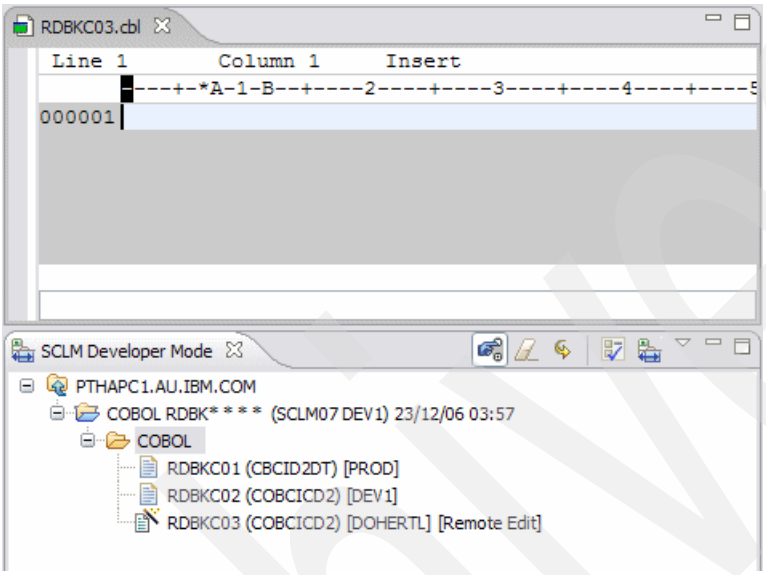


Figure 20-53 Adding a new member to SCLM

Once the code or text has been entered into the member through the editor pane, we can save the member by right-clicking on the new member in the SCLM view and selecting the **Check In** option. You can enter a change code if required, or change the language and select **OK**, at which point the member will be stored in SCLM.

Adding local workstation files to SCLM

It is also possible to add workstation files to SCLM. Later we will discuss in more depth working with long name files and SCLM. For this example, let us assume we have two COBOL programs in a directory that we want to add to our project in SCLM. The **Add Local Files** option is available from the project node, Group node and Type node. Right-click one of the nodes and select the **Add Local Files** option and you presented with an Explorer window where you can browse for the files you want to add, shown in Figure 20-54.

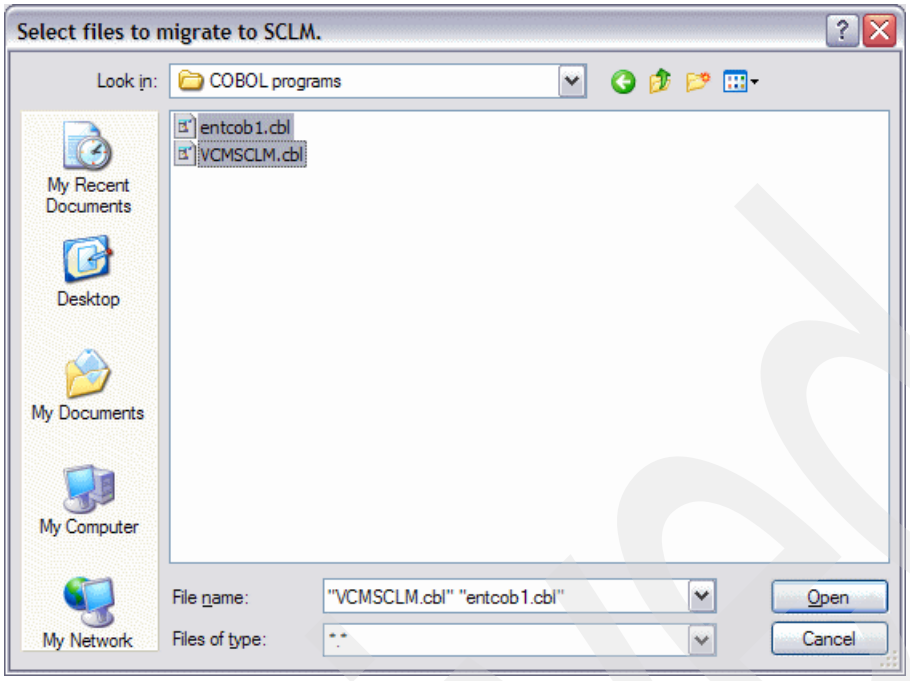


Figure 20-54 Selecting workstation files to add to SCLM

Select all the files you want to migrate to SCLM and click **Open**. You are then presented with the panel shown in Figure 20-55 where you assign a language to all the members that will be added to SCLM.

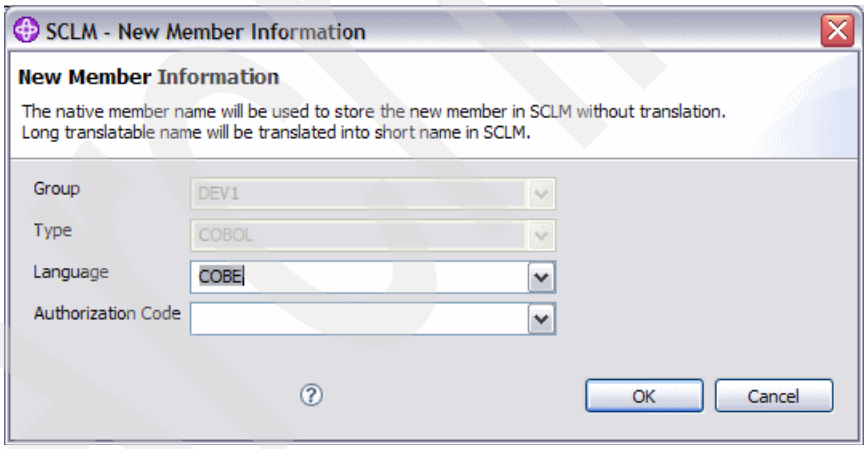


Figure 20-55 Assigning a language to the members

Developer Toolkit will zip all the selected files up and send them up to z/OS where they are unzipped and migrated into SCLM with the selected language. If you want to store members with different languages, then these must be grouped together and a number of separate migration actions will have to be performed.

Adding non-SCLM PDS members to SCLM

One of the tasks in migrating to SCLM is to copy members, or complete libraries of members, into SCLM controlled data sets. Or you may just have one or two members in a non-SCLM controlled PDS that you want to migrate into SCLM. This option is available from the project node in the SCLM view. Right-click the project node and select the **Add non-SCLM members to SCLM** option, and the Add non-SCLM members to SCLM wizard is displayed. The first thing you must do is specify the data set or data sets that contain the members that you are going to migrate into SCLM. The data set pattern can be specified in different ways:

XXX.* Retrieves all data sets starting with XXX (only data sets having 1 additional qualifier)

XXX.** Retrieves all data sets starting with XXX.

XXX.*.YYY Retrieves all data sets with XXX, one qualifier and ends with YYY.

XXX..YYY** Retrieves all data sets with XXX, multiple qualifiers and ends with YYY.

XXX.YYY(*) Retrieves all members from the data set XXX.YYY. A member pattern can also be specified.

The member specification (*) or (member pattern), if used, will cause the members that match the pattern to also be returned. The panel is populated with the returned list of data sets and members as shown in Figure 20-56.

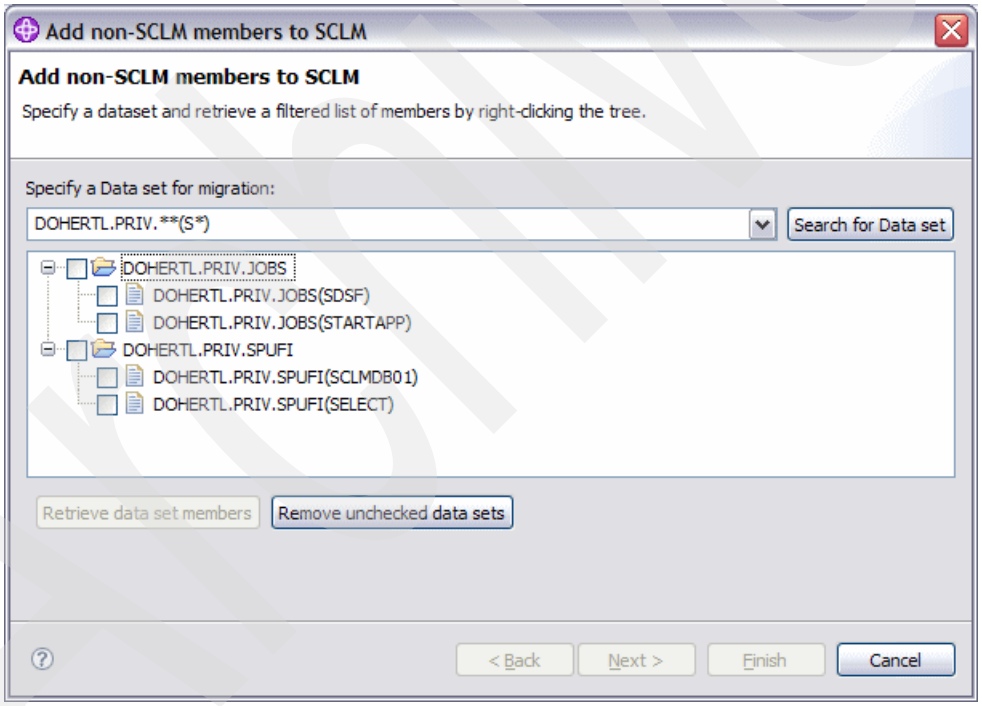


Figure 20-56 Specifying non-SCLM libraries to migrate members from

If a complete list of members is required, then the library in the list can be selected by clicking the data set name. This in turn activates the **Retrieve data set members** button, which can be clicked to retrieve members matching a specified pattern or all members. This is shown in Figure 20-57.

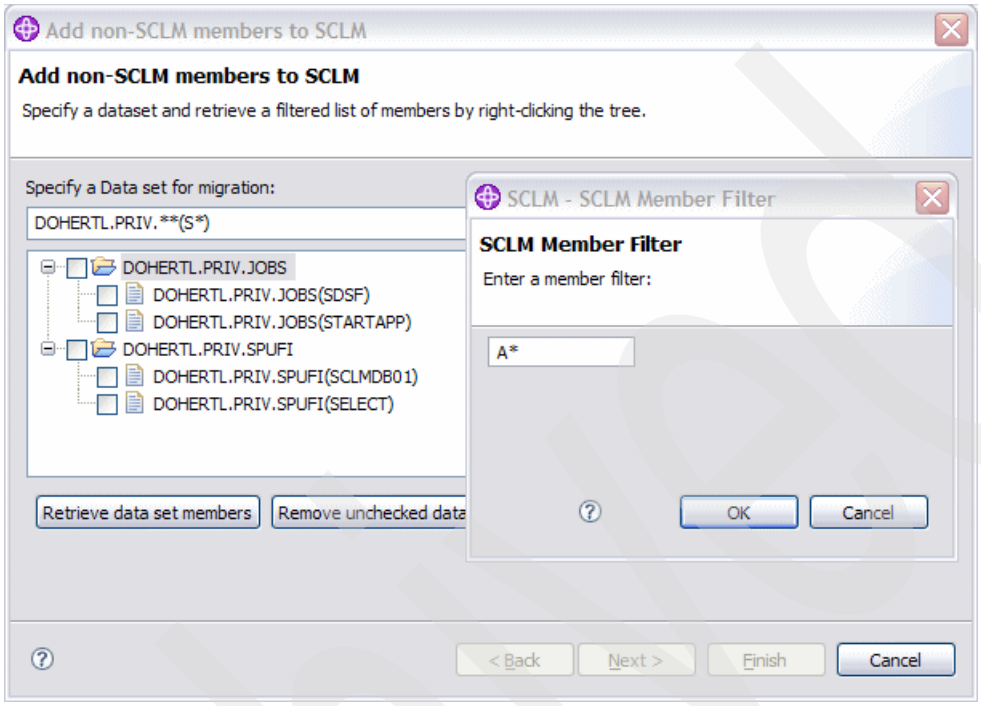


Figure 20-57 Retrieving members that match a certain pattern

When the **OK** button is selected, the list of members previously retrieved will be replaced with the newly retrieved member list.

It is also possible to build up the list of data sets that will contain members to migrate. For example, if there were members from a number of external COBOL libraries that were all going to be migrated into the same SCLM type, this could be specified by specifying the different data sets for migration until they were all listed. then member patterns for each data set could be used to build a complete list.

Once you have a complete list of data sets and members, you select the members to migrate by checking the box next to the member name. Alternatively, check the box next to the data set name to migrate all of the members in the list. If you only have data set names in your list, so you have not specified a member pattern, then all members in the checked data set will be migrated into SCLM.

Clicking the **Next** button will bring up the next panel of the wizard where you can specify the required language an SCLM type where the members will be migrated into. If all members will have the same language and type, use the **Select All** button to select all the members in the list, or use standard Windows techniques of using the Shift and Ctrl keys to select the members you want. Then click the **Update Selected** button to show the panel where you can specify the type and language, as shown in Figure 20-58.

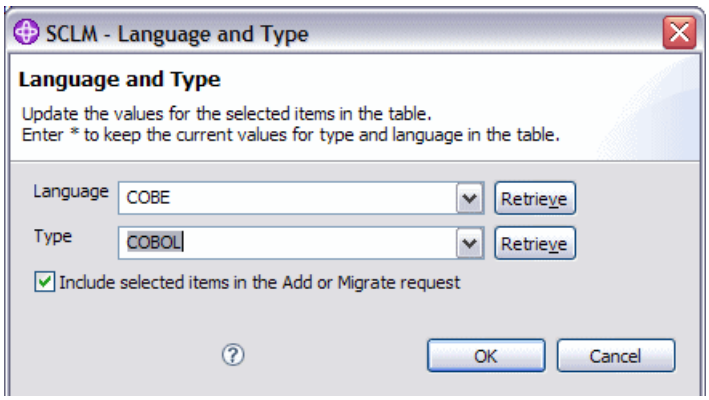


Figure 20-58 Specifying language and Type for migrated members

Once you click **OK**, the Add member property page, as shown in Figure 20-59, is updated with the required type and language information. You can now specify an authcode and/or change code if required that will be applied to all members in the list. Clicking **Finish** will add all the members to the specified types in SCLM.

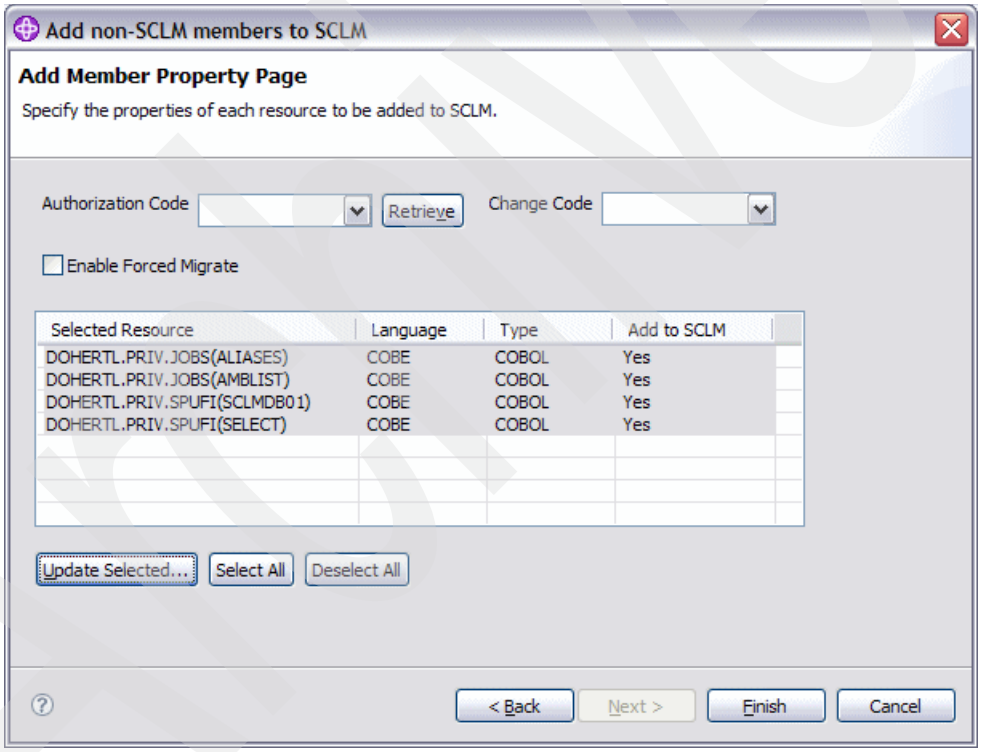
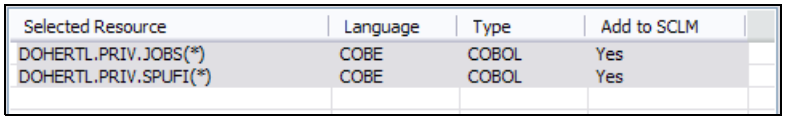


Figure 20-59 Add non-SCLM members to SCLM

If some of the members already exist in SCLM, you will need to select the **Enable Forced Migrate** check box for the migrate to work.

If you were migrating a whole data set, such that you did not specify a member pattern on the first panel of the wizard, then the language and type will be applied to all members in the selected data set as shown in Figure 20-60.



Selected Resource	Language	Type	Add to SCLM
DOHERTL.PRIV.JOBS(*)	COBE	COBOL	Yes
DOHERTL.PRIV.SPUFI(*)	COBE	COBOL	Yes

Figure 20-60 Migrating complete data sets into SCLM

20.2 Working with workstation files in a mainframe environment

SCLM Developer Toolkit provides the facility to store normal workstation files, such as Word documents, in SCLM. This useful feature allows you to store documentation related to traditional mainframe code in SCLM along with the programs that it relates to. Using ARCHDEFs, the documentation can then be promoted and kept with the code as it moves through the normal development life cycle.

This is achieved by using a long to short name mapping file. When a long named member is added to SCLM, the language that is specified for the member is checked against the TRANSLATE.conf file, which is defined by your SCLM administrator. If the language is defined as a long name language (LONGLANG keyword), then a generated 8-character member name is assigned to the file when SCLM adds the member. Any subsequent actions on that file in SCLM will result in a lookup on the long name to find the actual short name, so the member can be retrieved from SCLM.

Even though these long name files are stored as members in SCLM, they can still be worked on with the correct editors for their file extension. For example, if you stored a Word document in SCLM, then when the SCLM member is edited, Microsoft Word will be opened for the file. This process is defined by associating extensions to editors in the Eclipse preferences.

Before beginning to work on long name files with SCLM, there are three things that have to be configured:

- A language definition must be created for long name types. This will be done by your SCLM administrator. It is good practice to create translators for each of the types, such as LANG=DOC for Word Documents, LANG=XLS for Excel spreadsheets, and LANG=PDF for PDF files. The language definitions will be the same, with the only difference being the language. But by having separate language definitions in SCLM, it is easier to run reports such as getting a list of all Word documents. An example of a translator can be seen in 19.2.2, “TRANSLATE.conf” on page 453. However, if you want, you could have a single translator for all of these types files, for example, with a LANG=BINARY.
- Long name languages have to be specified in the TRANSLATE.conf, or in the site/project configuration files. This is done by the SCLM administrator and was described in 19.2.2, “TRANSLATE.conf” on page 453.
- You should ensure that you have configured your workspace so that any long named files you want to work with are associated with the correct editor or program. This is described in the following sections.

Adding a file extension to editor associations

Select **Windows** → **Preferences** → **General** → **Editors** → **File Associations** to see a listing of file extensions. This is shown in Figure 20-61.

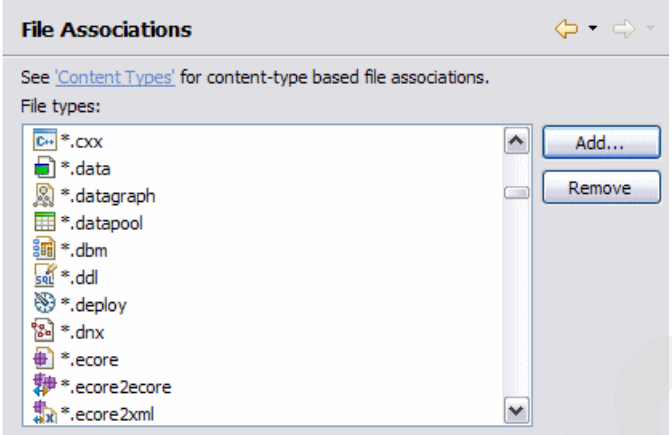


Figure 20-61 Associated file extensions

If the one you want is not displayed, such as *.doc, you can add it by clicking the **Add** box next to the File Types window. By default, extensions such as *.doc and *.xls will not be set up in Eclipse, so you will have to do this. A panel will be displayed as shown in Figure 20-62, where you can enter the extension you require, such as *.doc or *.xls.

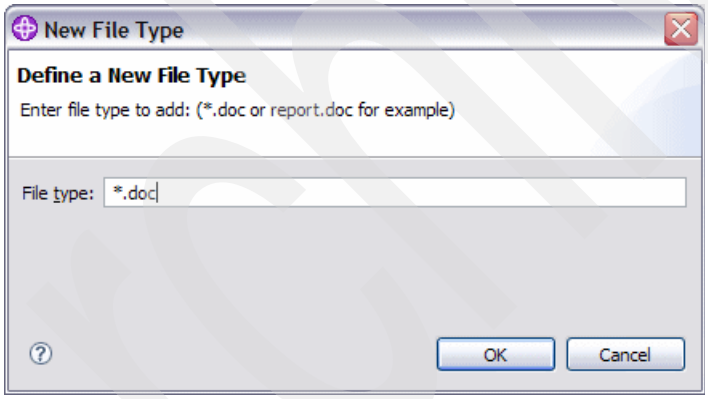


Figure 20-62 Adding a new extension to the editor preferences

Once you have added the extension, you can now associate it with an editor. Highlight the language you have just added and click the **Add** button next to the Associated Editors. From here you can select an Internal Editor or an External Program. For Word, you will select the **External Programs** radio button and then select the Microsoft Word program from the list as shown in Figure 20-63.

Add extensions for all of the long name file types that you are working with, such as *.doc, *.xls and *.pdf.

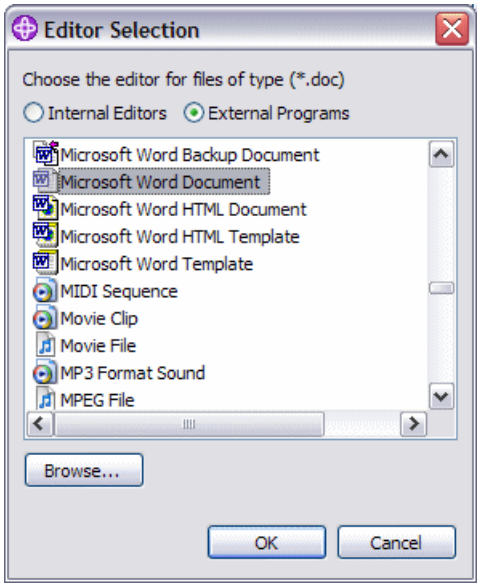


Figure 20-63 Editor Selection panel

Adding long name files to SCLM

You can add a new long name file to SCLM in two ways:

- Migrate an existing long name file into SCLM from your local workstation.
- Add a new long name file and save it in SCLM.

Migrate an existing long name file into SCLM

Normally the file, such as a Word document, already exists on your workstation and you just have to add the file to SCLM. The process for doing this is the same as was described in "Adding local workstation files to SCLM" on page 497. Using the **Add Local Files** context menu item, find the files on your workstation and select all the files of a single type/language combination that you are going to migrate. Associate a language and type with the files as shown in Figure 20-64. In this example we have selected a Word document called DBUTIL Support in Developer Toolkit.doc and we are going to store that into a data set with an SCLM type of DOC using a language of DOC.

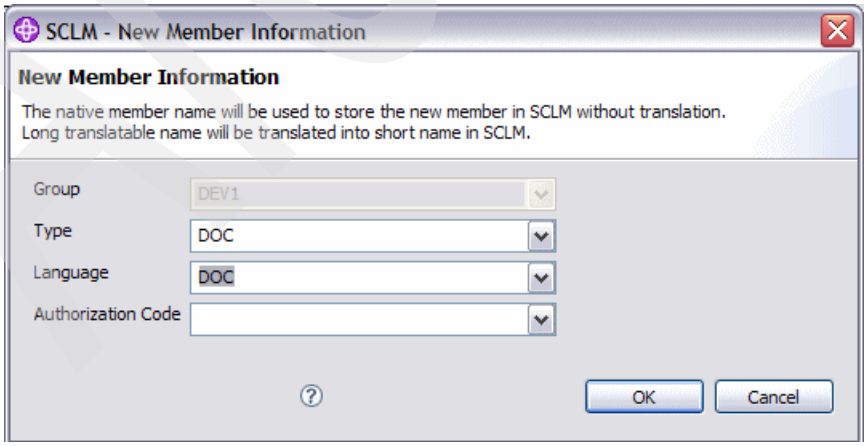


Figure 20-64 Adding a Word document to SCLM

Once you click **OK**, the files will be added to SCLM, and the SCLM view will be updated to reflect the new members that have been added. The short name that SCLM has generated will be visible, as will the long name that is associated with the member, as in Figure 20-65.

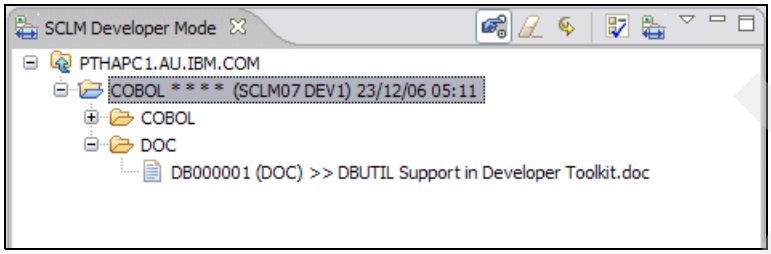


Figure 20-65 Viewing long name members in SCLM

In the above example we see that when the member was added to SCLM, it was assigned a short name of DB000001. You can toggle the long name on and off by using the , icon.

Add a new long name file and save it in SCLM

You can also add a new long name member to SCLM in a similar way to what was described in “Add new member by using the Eclipse editor” on page 496. On the project, group, or type node, select the **Add New Member to SCLM** context menu option. Instead of entering an 8-character member name in the Member field on the displayed panel, enter a normal workstation name, as shown in Figure 20-66.

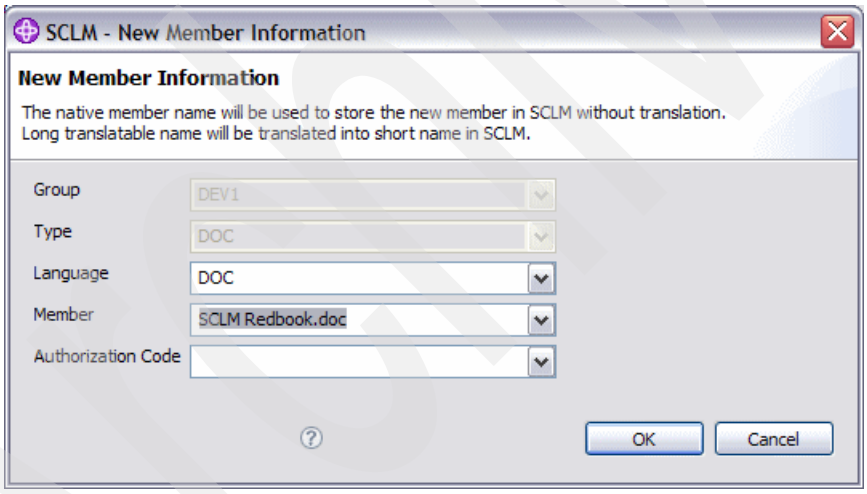


Figure 20-66 Adding a new long name file to SCLM

Clicking **OK** will cause SCLM to generate a short name for the new file name and then open the associated editor for the specified extension, in the example above, .doc. Figure 20-67 shows the message SCLM sends back informing you of the short name generated.

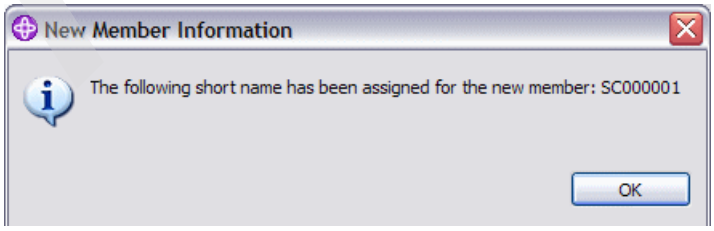


Figure 20-67 Generated short name for the new long name file being added

Microsoft Word starts automatically with an empty Word document where you can start working on the new document. Additionally, the SCLM view is updated with the new member name showing that the member is locked to your userid, as shown in Figure 20-68.

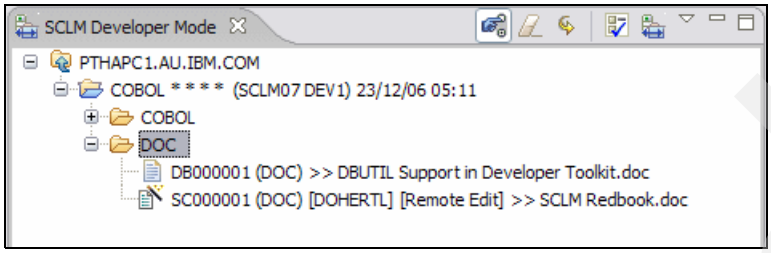


Figure 20-68 SCLM View showing new file being edited

Once you have finished editing the new word document, you can just save it. However, you may have to ensure that the type is saved as a "Word Document" in the Save as panel. Once saved, you can now check the file back into SCLM by using the **Check In** option on the context menu for the member as shown in Figure 20-69.

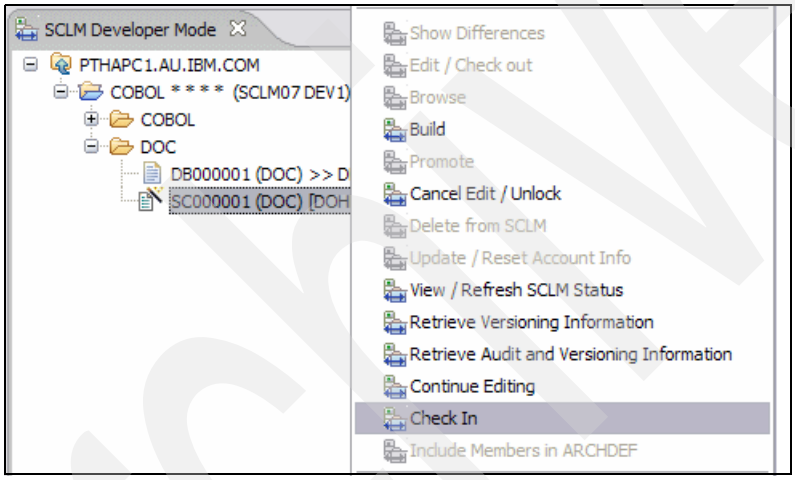


Figure 20-69 Checking the new Word document back in

Editing long name files with SCLM

Editing long name files works in the same way as when adding them. Use the filter list to list the members in the library that you store your long name files in, such as DOC. When you right-click the member and select the **Edit / Check Out** option SCLM Developer Toolkit will transfer the file down to the workspace and as it knows the long name, it can open the associated editor for that file extension.

20.3 Using the architecture definition wizard

Using the ARCHDEF wizard, it is then simple to add the document we created in the previous section, or any member, to an existing or new architecture definition member. The first step is to check out the required ARCHDEF from SCLM and open it for edit. Do this by making sure that the architecture definition type you use, in our examples ARCHDEF, is listed in the SCLM View. Right-click the member and select **Edit / Check Out** from the context menu. The required ARCHDEF will be open in the editor pane. If you then right-click the locked member in the SCLM View to bring up the context menu, you see a new option, **Include Members**, in ARCHDEF, which is now available, because we are editing a member with an ARCHDEF language. This can be seen in Figure 20-70.

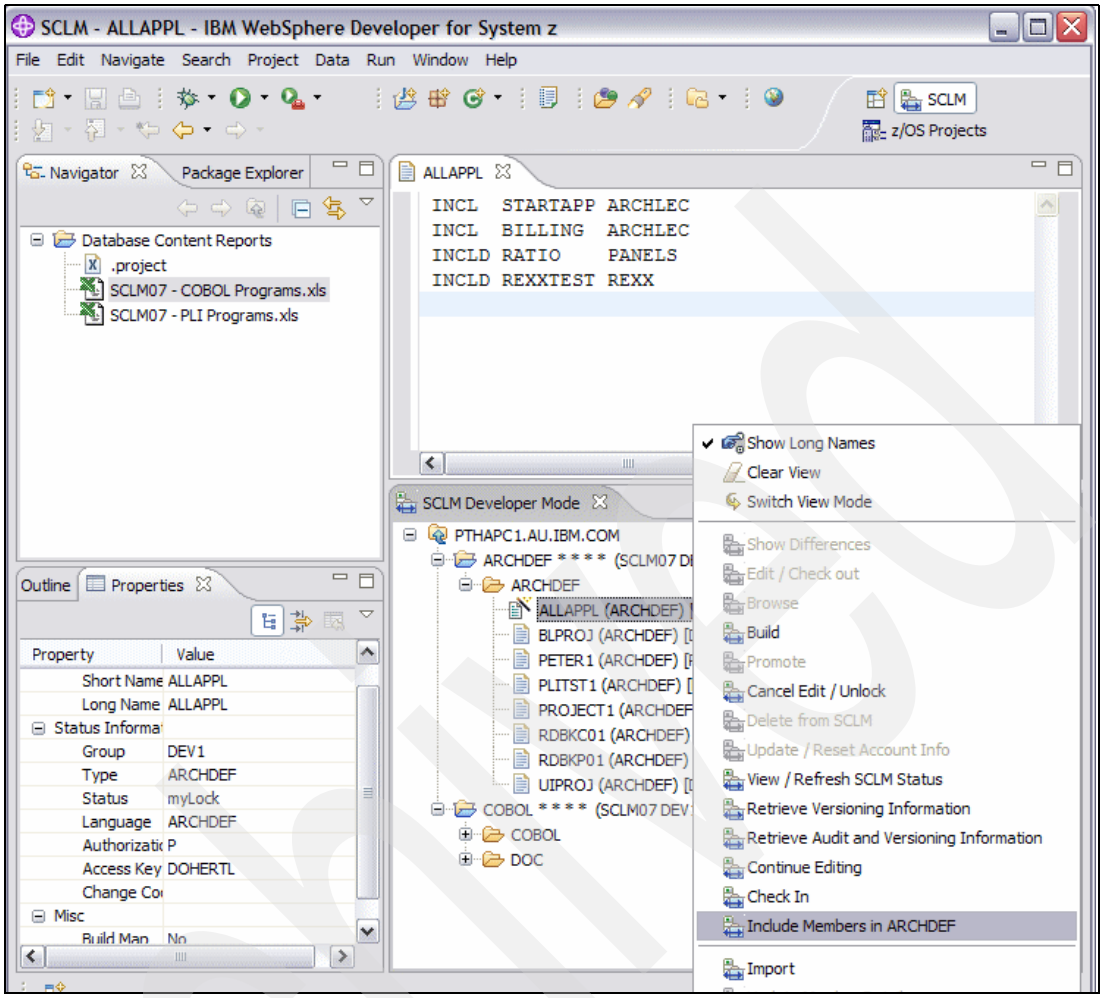


Figure 20-70 Including members in an architecture definition member

Selecting this option brings up the panel in Figure 20-71 to allow you to start adding members to the ARCHDEF.

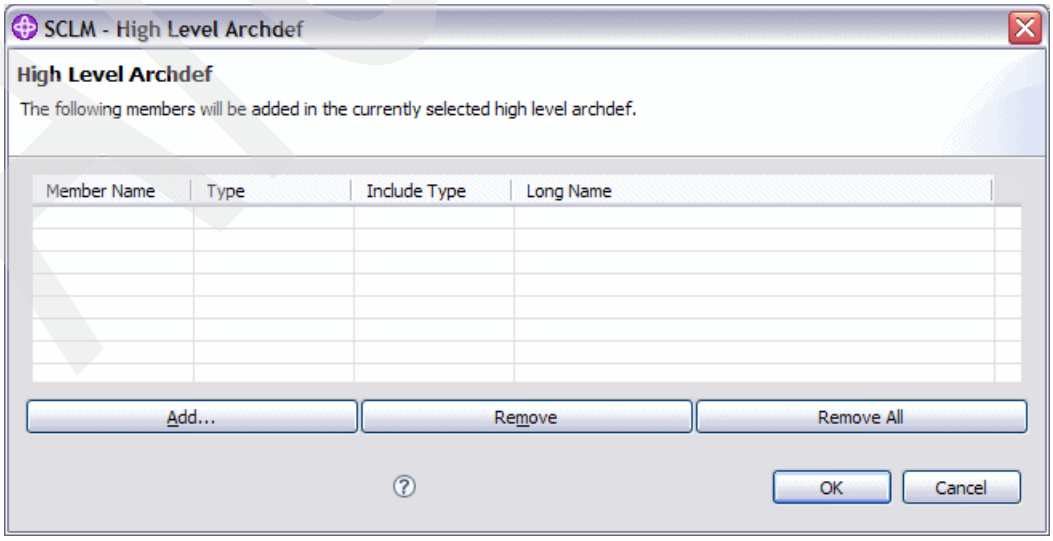


Figure 20-71 Adding members to an architecture definition member

Click the **Add** button and standard SCLM Developer Toolkit search panel is displayed where you can list the members you want to include in the ARCHDEF. Enter a type of DOC and click **OK** and the documents that exist in this type in SCLM will be listed in a panel, shown in Figure 20-72.

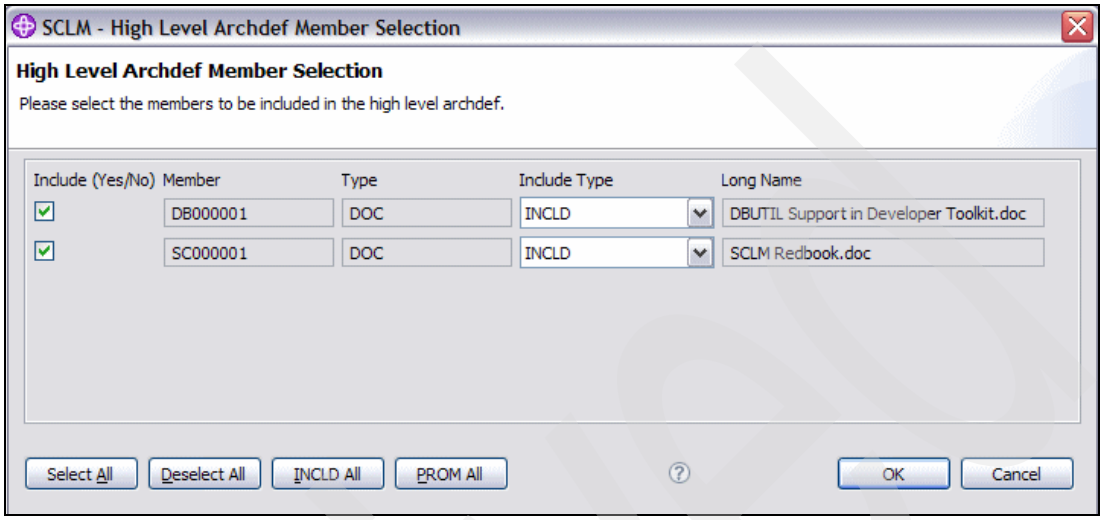


Figure 20-72 Returned DOC members from filter search

From here we can select the members we want, by selecting all or deselecting all and selecting individual members. We can also change the Include Type to **PROM** instead of INCLD.

Additionally, if you were to use the filter page to list ARCHDEF members, then the Include Type would be set to **INCL** as per normal SCLM rules.

Once you have selected the members you are interested in, click the **OK** button and the selected members are added to the selection list as shown in Figure 20-73.

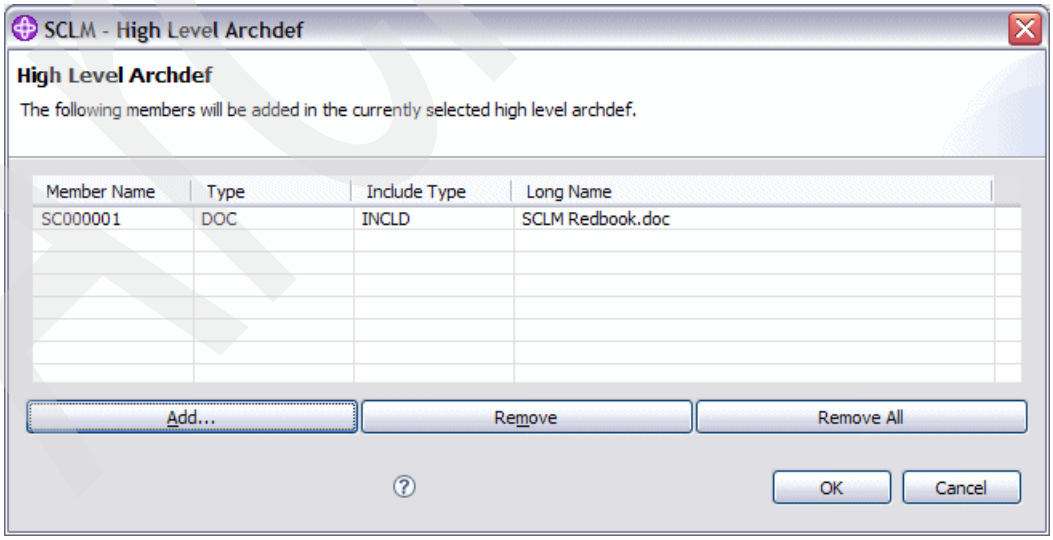


Figure 20-73 List of selected members to add to the architecture definition member

At this point you can click the **Add** button again to go and search for additional members to include in this ARCHDEF. Once finished, clicking the **OK** button will bring you back to the Editor pane on the ARCHDEF with the selected members added to the ARCHDEF, as shown in Figure 20-74.

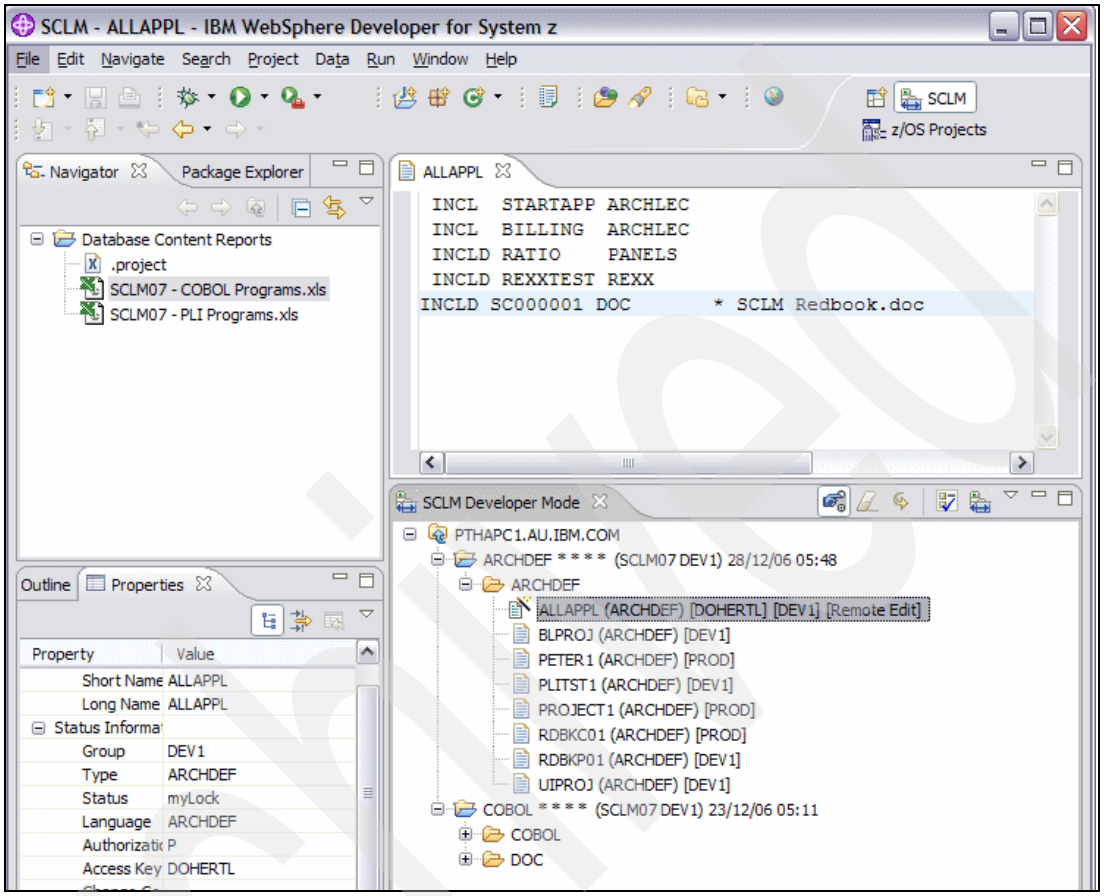


Figure 20-74 Selected members now included in the ARCHDEF

Once again, at this point you can right-click and go through the architecture definition wizard to add additional members.

20.4 Working with the SCLM IDE View

Most of the time when working with traditional z/OS members you will use the SCLM View as described throughout this chapter. However there are occasions when you can use the SCLM IDE View. The most common instance of this is when you have created a COBOL or PL/I application in the workspace and have tested and debugged it solely on your workstation, but are now ready to add the modules to an SCLM project to be integrated into a larger application.

How to code, test, and debug traditional COBOL and PL/I modules through WebSphere Developer for System z (WD/z) is out of the scope of this book. So let us assume that we have already performed unit test on a COBOL module on out workstation using the tools provided for us by WD/z as shown in Figure 20-75. What we now want to do is add that source program to SCLM.

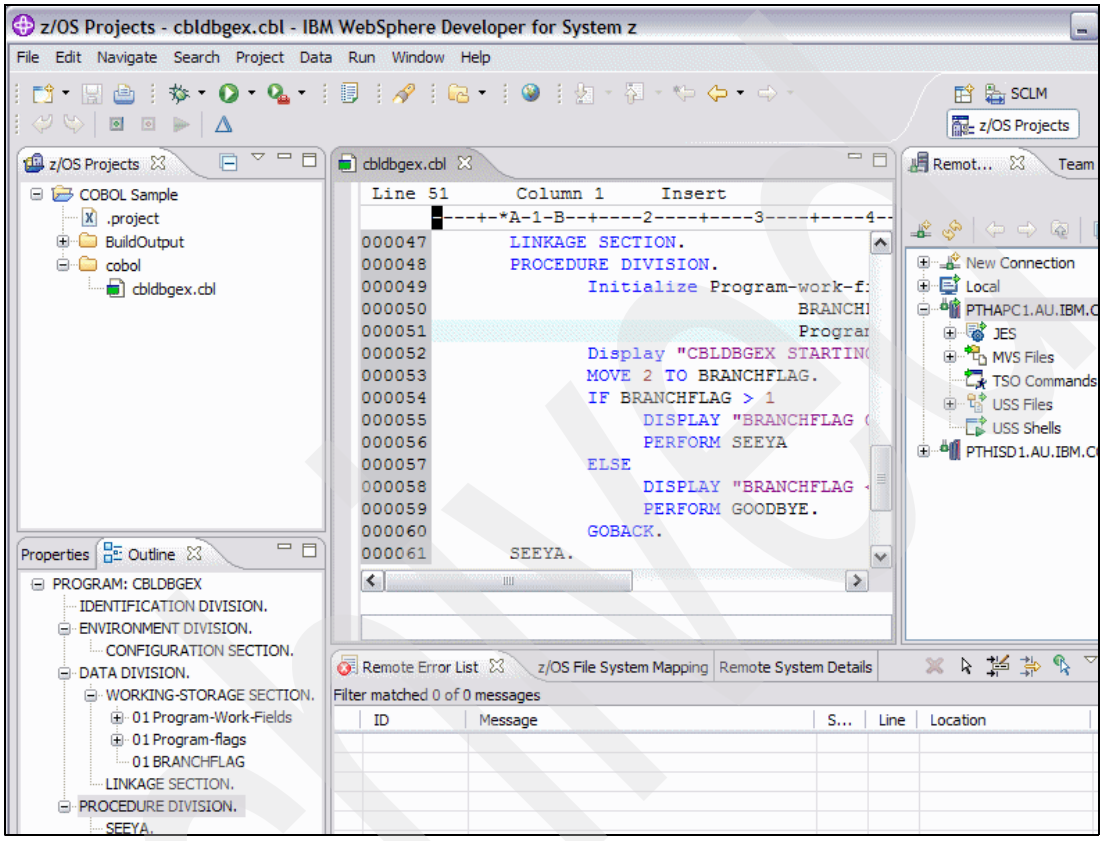


Figure 20-75 Creating a COBOL application through WD/z

Once we have finished developing and testing our COBOL program in WD/z, we can add the module or modules to SCLM. The difference between a COBOL application developed in WD/z and a Java Application developed in WD/z is that many of the additional files the IDE uses, such as the .project file, may not be required. If the ultimate destination for the COBOL program is z/OS, then all that is required is to migrate the COBOL programs into SCLM. Using the SCLM IDE view utilizes the Eclipse “Team” options. This is a standard interface that various Software Configuration Managers (SCMs) can hook into and many functions are provided via the standard context menus.

Many of the Eclipse perspectives offer a view that SCLM Developer toolkit calls the IDE View. For example:

- Working with the z/OS projects perspective, you see the z/OS projects view as shown in Figure 20-75 on page 510.
- In the Resource perspective or the SCLM perspective, you see the Navigator view as show here in Figure 20-76.

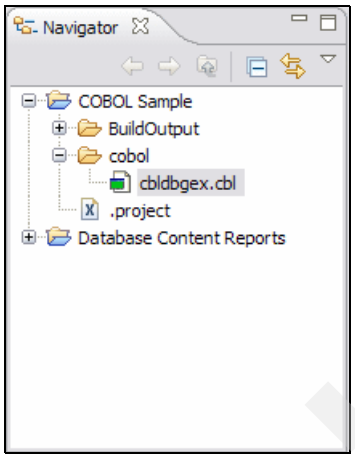


Figure 20-76 Navigator View

- In the Java perspective you see the package Explorer view as shown in Figure 20-77.

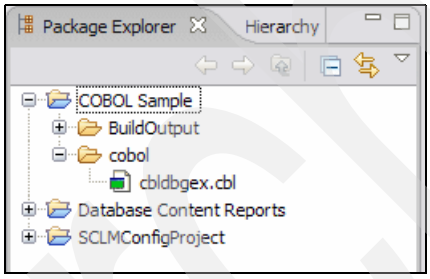


Figure 20-77 Package Explorer View

Each of these views can offer different options relevant to the types of projects displayed within them, but all of the views offer the team context menu item. So regardless of how you work in the IDE, you will have access to the team options and subsequently the SCLM Developer Toolkit options.

Archived

Associate the project in the IDE view with SCLM as the SCM provider

The first task is to associate the project and the parts contained in the project with SCLM as the SCM provider. This is the action of sharing the project. Taking the COBOL Sample project shown in the previous figures, we right-click the Project name, then expand the Team menu item to select the **Share Project ...** menu item as shown in Figure 20-78.

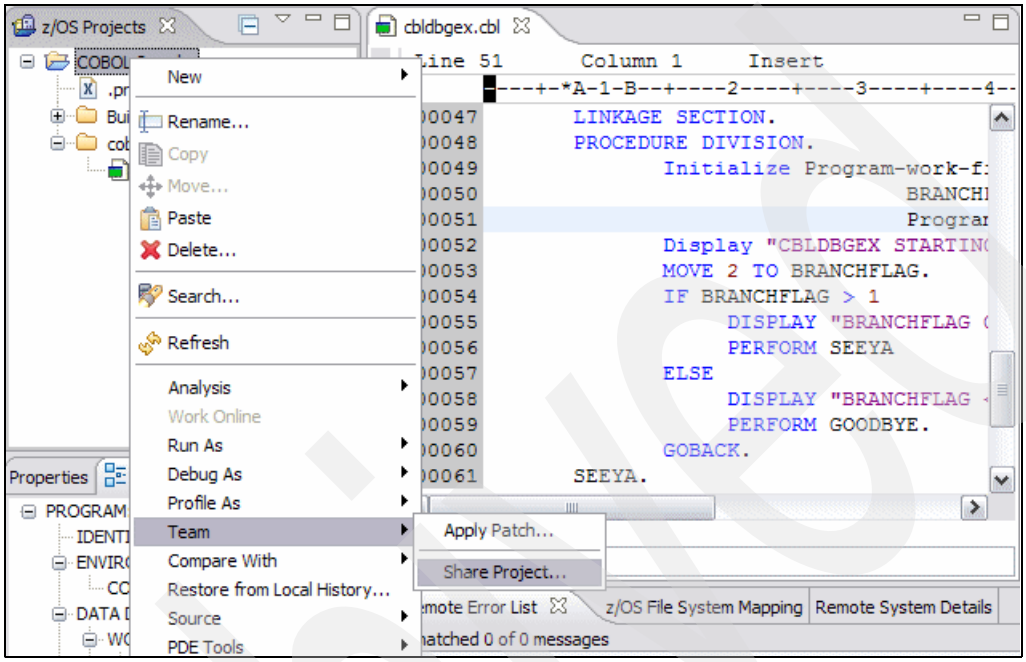


Figure 20-78 Selecting the Share Project team item

This then gives us a list of SCM providers to choose from. SCLM will be listed there as shown in Figure 20-79 if the SCLM Developer Toolkit plug-in has been installed.

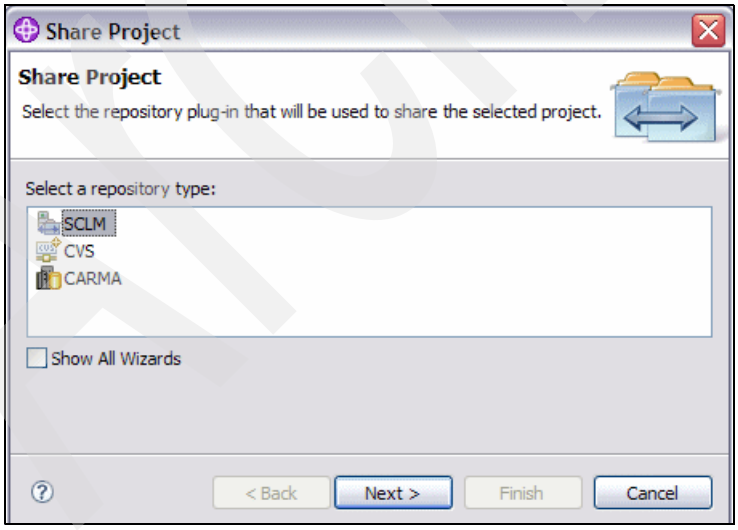


Figure 20-79 Selecting the SCM provider

You now need to associate the Eclipse project with and SCLM project as this is where the selected code will be stored. Click the **Next** button and you can then select the machine ID where the SCLM project exists from the pull down. If you are not already connected to that system through your RSE or HTTP connection, you will be prompted for a userid and password. You then select an SCLM project from the pull-down list of previously entered projects, or enter a new SCLM project name. Alternatively you can enter a wildcard for the project filter. This panel is shown in Figure 20-80.

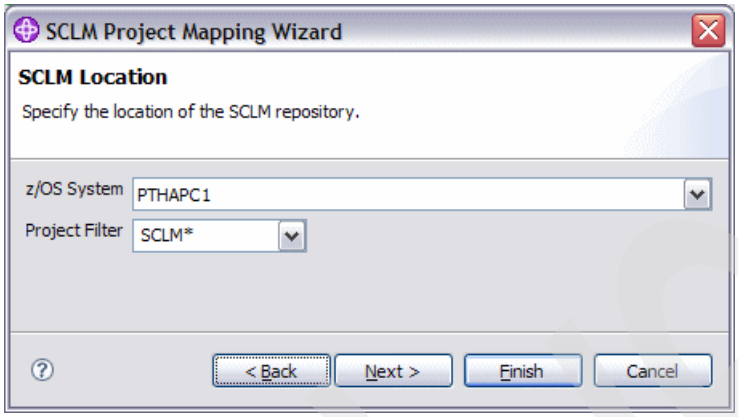


Figure 20-80 Selecting the z/OS system and SCLM Project. Filter

Clicking **Next** with either prompts you for a userid and password if you are not already logged on, or if you are already logged on, you will then be prompted to enter a development group, which can be selected from the pull down list, plus if you are going to use an alternate project definition for this Eclipse project you can enter that, as shown in Figure 20-81.

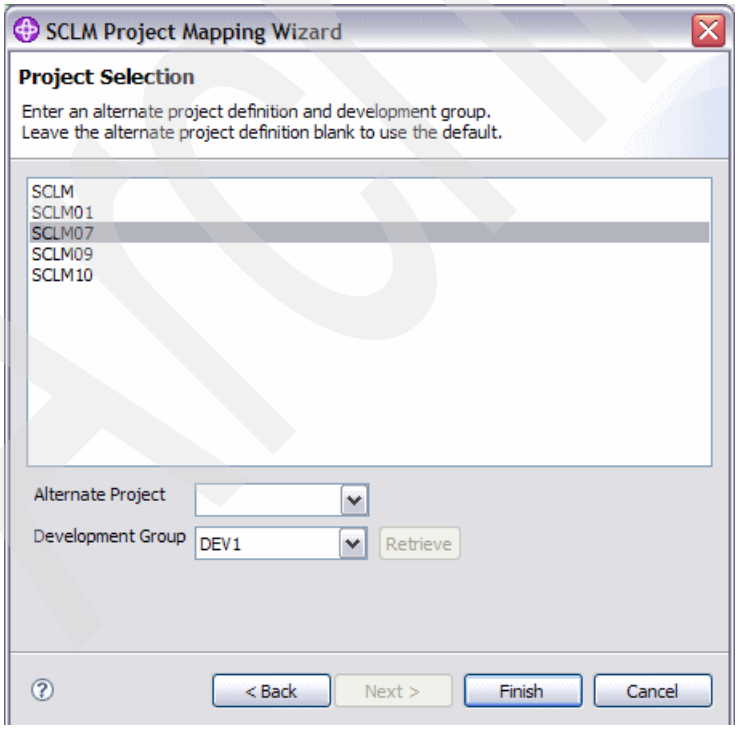


Figure 20-81 Selecting the SCLM project, Alternate and development group

Clicking **Finish** will associate the Eclipse project with this specific SCLM project. The decorators in the IDE view are updated to reflect this association as shown in Figure 20-82. Any files that are in the Eclipse project will also have a decorator added, **[Not in SCLM],** to indicate that no files have yet been migrated to the associated SCLM project.

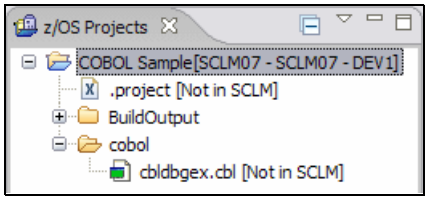


Figure 20-82 Eclipse project associated with an SCLM project

Add the files to SCLM

The next step is to migrate any files that are required to the SCLM project. You can select individual files by selecting them and then right-clicking one of them to bring up the context menu and selecting **Team** → **Add to SCLM** or you can select all files in the project by right-clicking the project to bring up the context menu and then selecting **Team** → **Add to SCLM**. Figure 20-83 shows the Resource Selection panel where you can select files and directories that will be imported to SCLM. As this COBOL program is a stand-alone module that will be maintained on z/OS once we have migrated it there, the only directory that we select is the **cobol** directory. We deselect the **BuildOutput** and the **.project** file, as these are not required.

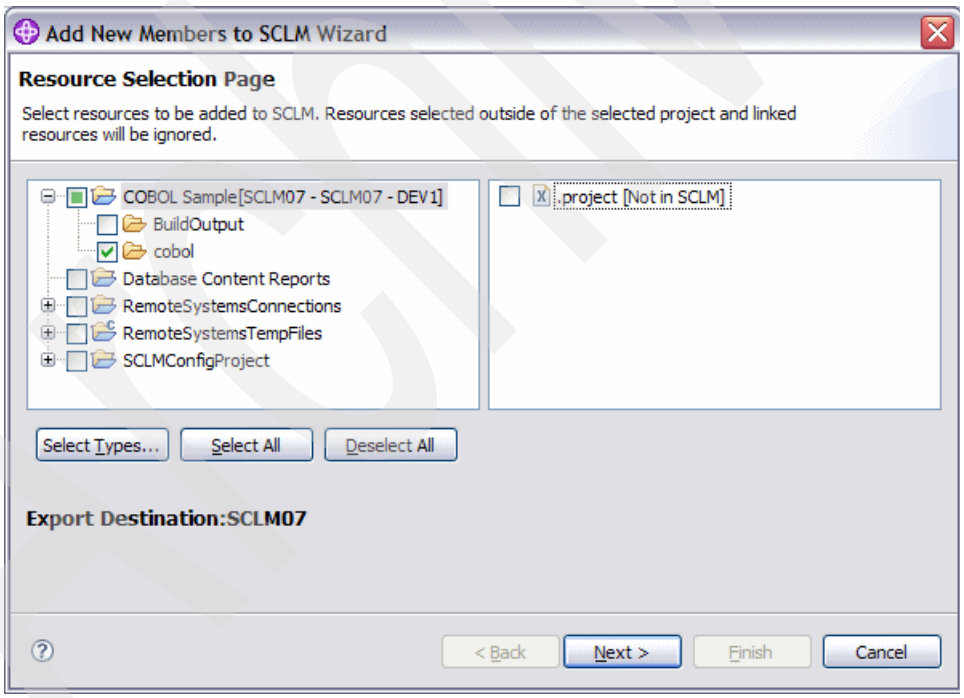


Figure 20-83 Resource Selection panel

Clicking **Next** then brings up the Add member property page shown in Figure 20-84. This is the panel where we associate SCLM language and type to all the files selected. We can also associate a non-default authorization code and a change code if required. If the member already exists in SCLM, then there is a forced migrate option to migrate over the existing member. Additionally, the migrate can be run in batch mode, for example, if there were a large number of files to add to SCLM.

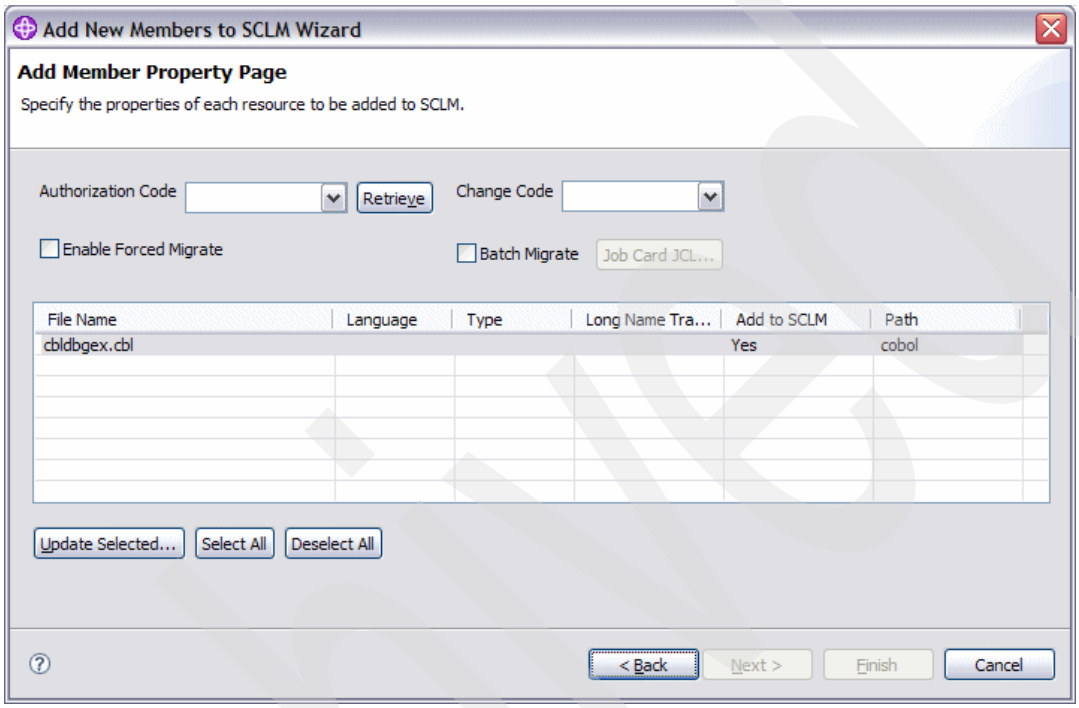


Figure 20-84 Add Member Property panel

Select the member(s) that you want to add to SCLM and click the **Update Selected ...** button, The Language and Type panel is displayed as shown in Figure 20-85. You can select the language and type from pull-down lists and you may need to click the **Retrieve** button to first populate these lists.

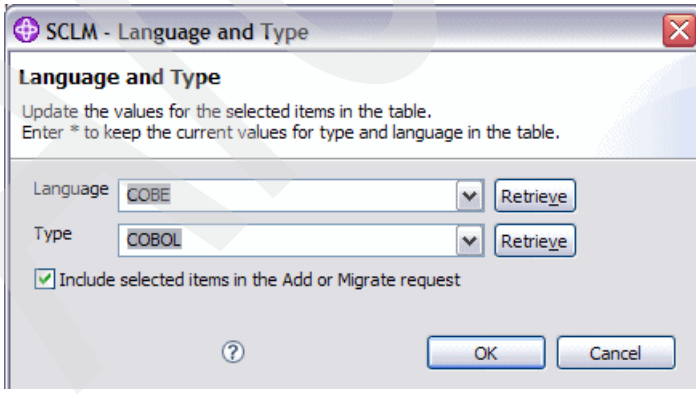


Figure 20-85 Adding SCLM Language and Type

Perform this action for all members in the list. Also, if there are members that you do not want to include in the migration, you can uncheck the **Include selected items ...** check box shown in the previous figure.

Once all the files you want to migrate have had languages and types associated with them as shown in Figure 20-86, you can click the **Next** button to proceed to the next panel in the migration wizard.

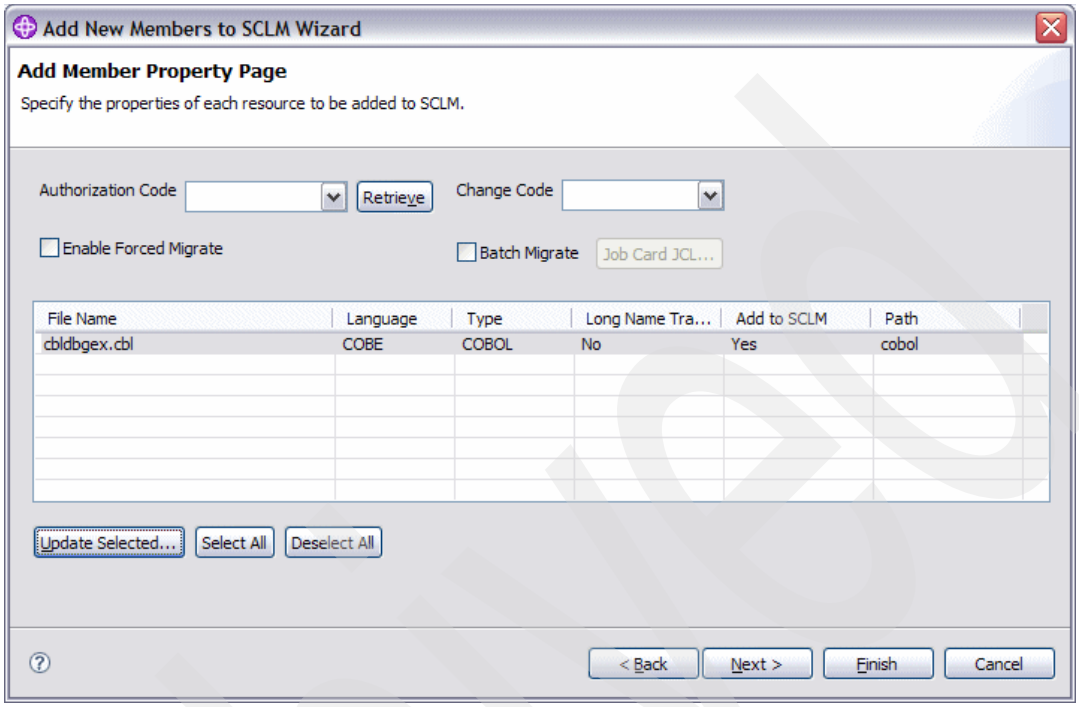


Figure 20-86 List of files to migrate to SCLM

The Project Architectural details panel optionally allows you to create an ARCHDEF that will contain SCLM INCLD statements for all of the files you are migrating into SCLM. At this point you can add any additional INCL or INCLD statements that may be required for members that already exist in SCLM. Figure 20-87 shows the Project Architectural Details panel. Enter the ARCHDEF name you want to create or update and the ARCHDEF type, as the type may actually be called something else such as PACKAGE. If you are updating an existing ARCHDEF you can search for the ARCHDEF name by clicking the **Browse** button.

By clicking the **Add...** button the standard SCLM Developer Toolkit filter panel is displayed so that you can search for members that match the filter criteria and then add them to the ARCHDEF if required.

There are also some Java/J2EE specific fields on this panel that are not required to be filled in for migrating traditional applications. Leave the **Include archdef statements for J2EE ...** check box unchecked so these Java/J2EE fields are left deactivated.

Once you have added any additional SCLM controlled members to the ARCHDEF, if required, you click the **Finish** button and Developer Toolkit will transfer the selected files over to SCLM and store them in the SCLM controlled library in the development group that you specified when you mapped the project to SCLM. Additionally the ARCHDEF that you specified, if you specified one, will be created if it does not exist or updated if it does exist.

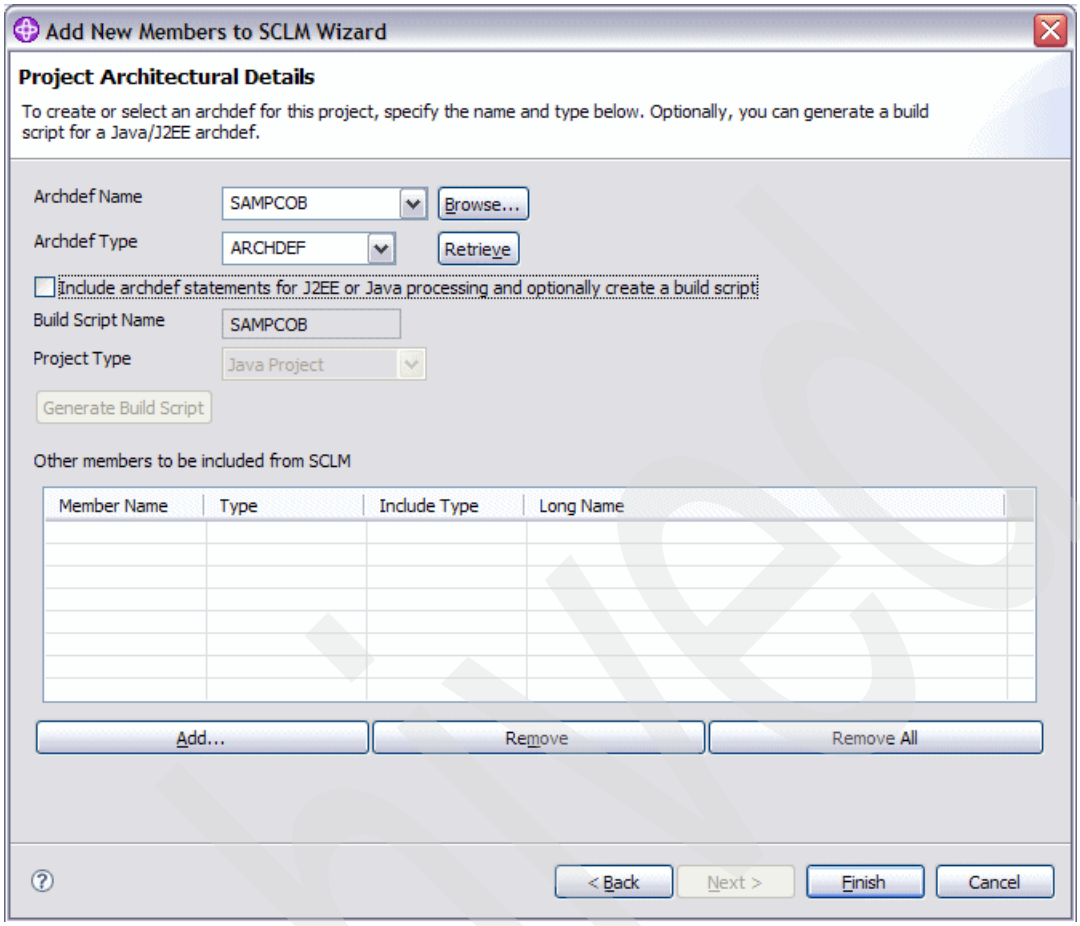


Figure 20-87 Project Architecture Details panel

The IDE view will be updated to reflect the fact that the member is now in SCLM as shown in Figure 20-88.

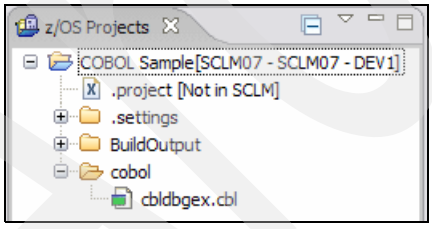


Figure 20-88 Eclipse project associated with an SCLM project post migration

Editing the member through the IDE view

You can now check the member out using the IDE view context menu by right-clicking the member then selecting **Team** → **Edit / Check Out**. The member is locked in SCLM and opened in the Editor pane in the IDE. Additionally, in the IDE view, the decorators are updated to show that the member is now locked, as shown in Figure 20-89.

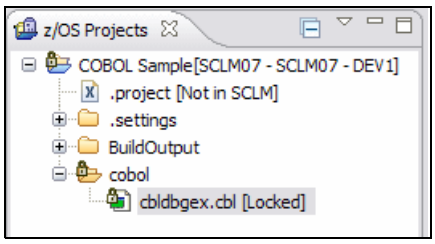


Figure 20-89 Editing the migrated member

Once edited, the member can be saved back in SCLM by right-clicking the member and then selecting **Team** → **Check In**.

Building the application through the IDE view

When you added the application to SCLM you will optionally have created an ARCHDEF that, in the case of a COBOL application, will be used to build the application into a load module. However by default the ARCHDEF that is created will not have all the correct SCLM keywords required to create a linkedit control ARCHDEF to build a load module. So before you can build the ARCHDEF for the first time you need to edit it to add the required keywords.

The easiest way to do this is to use the SCLM View as described previously in 20.1, “Working with the SCLM explorer view” on page 464. If you prefer to use the IDE View, however, it is possible to add the ARCHDEF to your Eclipse project as a file and then check it out and in as you would with any file in the project.

To import the ARCHDEF from SCLM into the Eclipse project, right-click the project to bring up the context menu then select **Team** → **Import SCLM Project**. You are presented with the standard SCLM Developer Toolkit preference page, where you enter the filter parameters to select the ARCHDEF you want as shown in Figure 20-90 on page 519.

In this example we know that the ARCHDEF we created was called SAMPCOB.

Archived

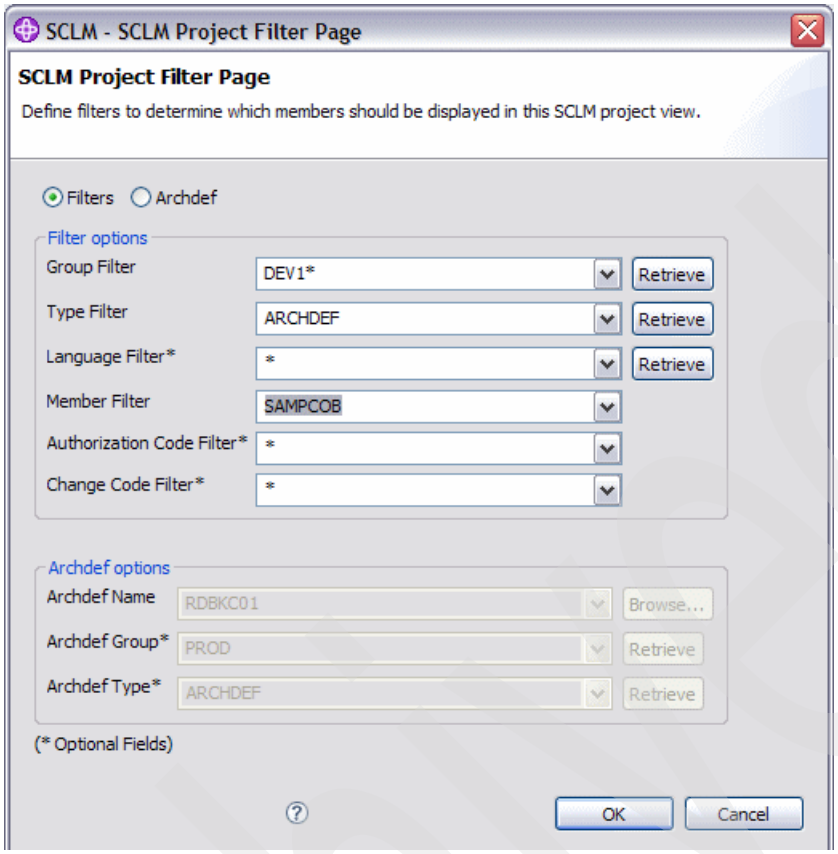


Figure 20-90 Selecting files to import into the IDE view

When you click **OK** the import process is run and the selected file(s) is then added into the Eclipse project as shown in Figure 20-91.

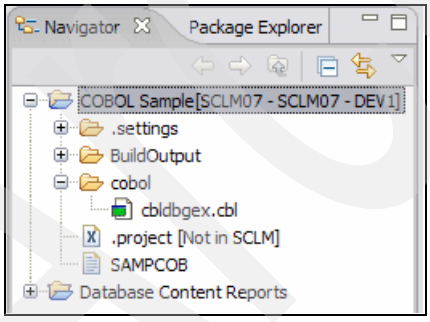


Figure 20-91 Project with ARCHDEF added

You are then able to edit the file in the same way as any other file in the IDE view by right-clicking the SAMPCOB file to bring up the context menu and then selecting **Team** → **Edit / Check Out.** Add the required SCLM keywords to allow this application to be built into a load module. You may need some assistance here from your SCLM Administrator, but by default, the default parameters to do this are the LOAD and LMAP keywords as shown in Figure 20-92.

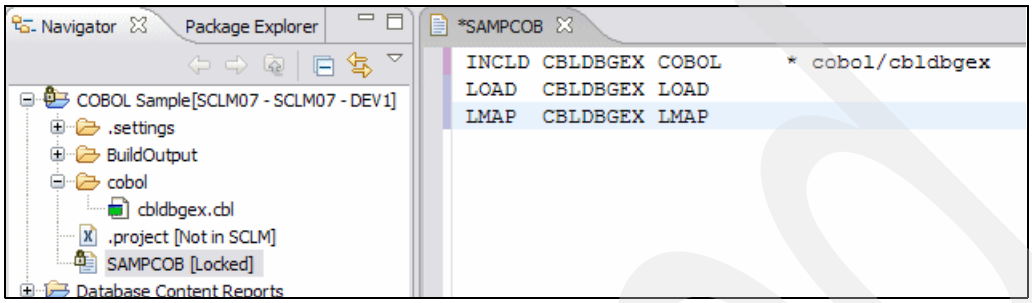


Figure 20-92 Adding SCLM keywords to the ARCHDEF member

You then check the file back in to SCLM by right-clicking the member and selecting the **Team** → **Check In** context menu option.

To build the member now, right-click the Eclipse project and from the displayed context menu, select **Team** → **Archdef Build**, which displays the Build panel shown in Figure 20-93. Enter the ARCHDEF name and type and click **OK** to invoke the build. For more information on Build, see 20.1.5, “Building members” on page 478.

Archive

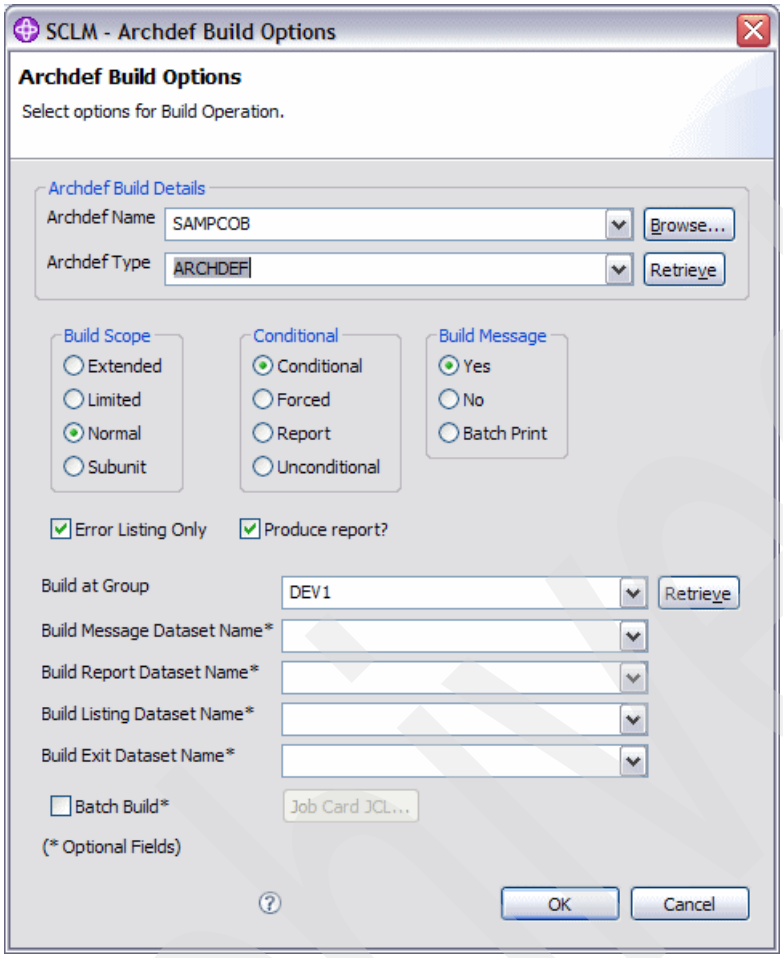


Figure 20-93 Building the ARCHDEF

Once the build completes you will be informed. Subsequently you can now make additional changes to the source members and then build the ARCHDEF that generates the load module by using the **Team** → **Archdef Build** context menu option on your Eclipse project.

Promoting the application through the IDE view

Once you have finished changing and testing the application on z/OS you can then promote the application ARCHDEF through the IDE view. Right-click the Eclipse project and select Team Archdef Promote from the context menu. This will display the ARCHDEF promote panel where you enter the name of your application ARCHDEF as shown in Figure 20-94.

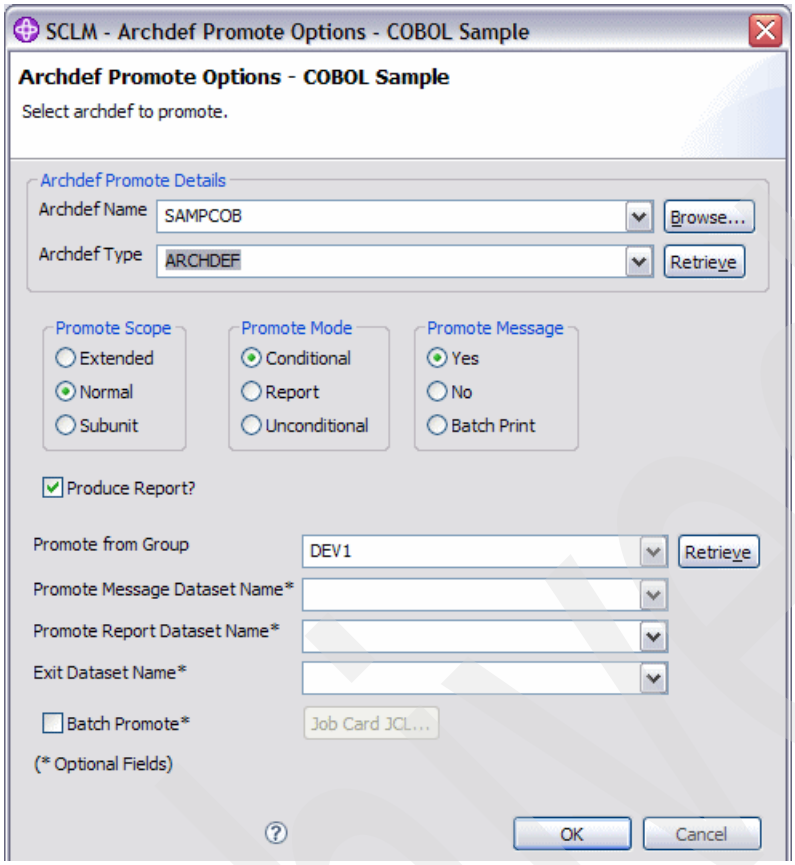


Figure 20-94 Promoting an ARCHDEF via the IDE view

Promoting the ARCHDEF will move the code from one group to the next in the SCLM hierarchy. There will be no visible change to the files shown in the IDE view. next time you need to check a module out through the IDE view, SCLM Developer Toolkit will lock it and copy it from wherever in the SCLM hierarchy it finds the code.

21

SCLM Developer Toolkit for Java Applications

In this chapter we provide an overview of how to use the features provided by SCLM Developer Toolkit to manage Java/J2EE projects, including basic configuration requirements and a usage scenario of how features should be utilized by Java/J2EE developers in their day-to-day operations. We also provides a foundation for more advanced topics such as deployment of Java/J2EE projects.

Archived

© Copyright IBM Corp. 2007. All rights reserved.

21.1 Overview

IBM SCLM Developer Toolkit is a Source Code Management (SCM) system to manage resources of projects in the Eclipse environment. The product is developed using the Eclipse's team plug-in, which defines additional APIs that allow plug-ins to integrate the function of a versioning and configuration management repository. Because of this, SCLM Developer Toolkit provides a consistent user experience to other SCMs available in the Eclipse environment such as CVS, as well as providing additional SCLM specific functionality.

Most of the functionality provided by SCLM Developer Toolkit contributes to the standard Eclipse's team extension points, hence people who are familiar with an SCM in the Eclipse environment would find it easy to navigate the features of SCLM Developer Toolkit.

- **SCM mapping configuration via Share Project wizard:** When registering a project to be managed by SCLM, it is mapped via the standard wizard. In the first page of the wizard, SCLM DT is listed along with other SCM to choose from, as shown in Figure 21-1.

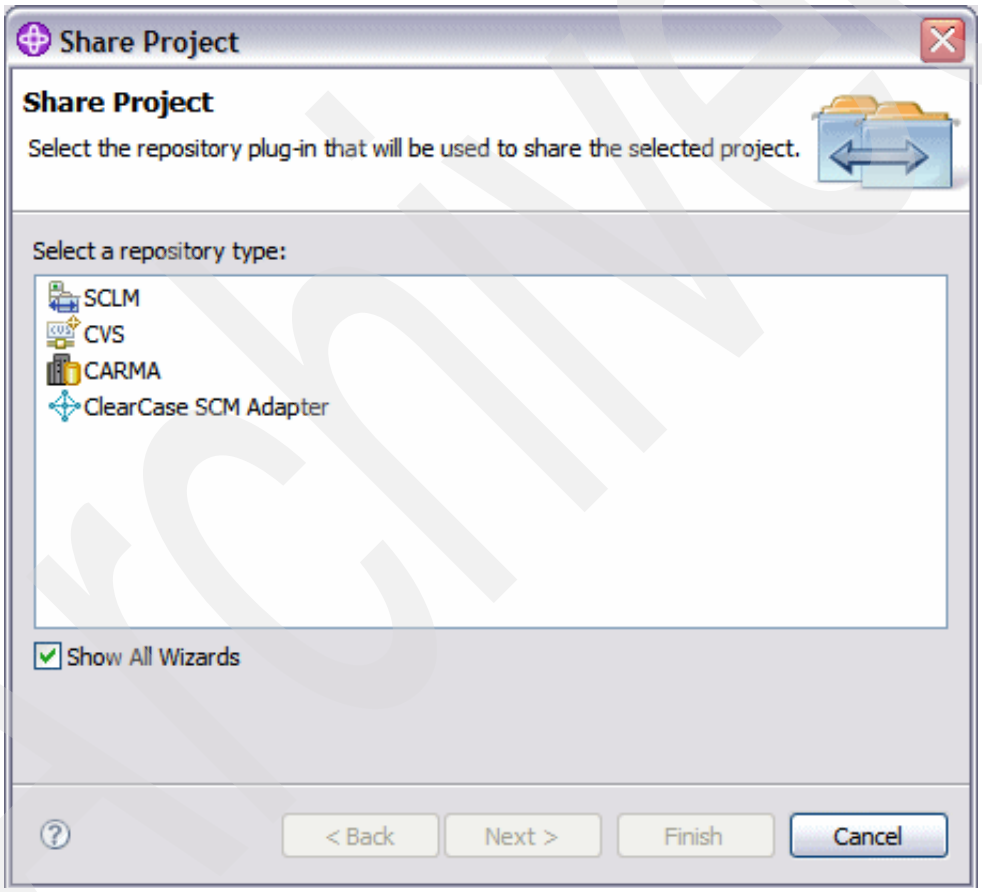


Figure 21-1 SCM Share Project configuration wizard

- **SCM actions accessible via context menu under Team:** Most of the DT actions applicable to the resources of a project are accessible via the *Team* menu item in the pop-up/context menu.
- **Preference page under Team:** Configurable parameters can be specified in the SCLM DT preference pages, which are available under the *Team* preference tab.

Archive

21.2 Setting up SCLM Developer Toolkit and your SCLM projects

Additional language types and types must be added to store Java/J2EE projects in SCLM using IBM SCLM Developer Toolkit. Such additional requirements are described in this section.

21.2.1 Java/J2EE language definitions

A language type associated with a member stored in the SCLM repository defines the way the member is built in SCLM. The following language types are required to store Java/J2EE project in SCLM.

JAVA — JAVA language translator (EBCDIC source)

(SCLM translator for individual Java programs)

This language definition should be assigned to Java programs if you want to store the Java source in EBCDIC on the host (that is, the source may be viewed or edited directly on the host through ISPF).

The advantage to defining programs with this language definition lies in being able to edit and view the source directly on the z/OS host. The disadvantage is that codepage conversions must take place when migrating or importing projects from the Client to Host. Fortunately, SCLM Developer Toolkit handles these conversions automatically in its import operations.

When building the source of the language definition, JAVA the translator behaves differently depending on how the source is being built (directly or via an ARCHDEF). If it is an individual Java program selected, then this source is copied into the z/OS UNIX file system workarea and JAVAC compiled, the subsequent classes are short name translated and stored back into SCLM under SCLM type JAVACLAS. If this is part of an ARCHDEF build, then compilation has already taken place when the J2EE build script translator was invoked (multiple Java source was compiled together). The JAVA translator then just copies and stores the resultant classes back into SCLM.

JAVABIN — JAVA language translator (ASCII source)

(SCLM translator for individual Java programs)

This language definition should be assigned to Java programs if you want to store the Java source in ASCII on the host (that is, the source is binary transferred between Client and host). The build process provided by SCLM DT transparently handles members stored as JAVA and JAVABIN at build time.

J2EEPART — J2EE text language part

This language definition should be assigned to Java/J2EE components where no particular parsing occurs upon building the components. This language definition would be associated with text based J2EE parts that require ASCII/EBCDIC translation when moving between the Client and z/OS host (that is, the source is stored in EBCDIC on the host and directly editable). Non-Java source may be generically slotted under this language definition if no particular build parsing is required, such as .classpath, .project, html, xml, or property tables.

J2EEBIN — J2EE binary language part

This language definition could also be assigned to Java/J2EE components where no particular parsing occurs upon building the components.

This language definition would be associated with binary based Java/J2EE parts that require no translation when moving between the Client and z/OS host. Binary objects may be generically slotted under this language definition if no particular build parsing is required (such as gif, jar, class, war, or ear). Additionally other J2EE files, as mentioned above (.classpath, and so on) may also be stored under this language definition if source is desired to be stored on the host as ASCII.

J2EEANT — J2EE ANT Verify/Build Translator

(SCLM language translator for Java/J2EE build script)

This is the main language definition for Java/J2EE builds and is invoked when a J2EE ARCHDEF is built as a verify translator. The J2EEBLD build script is stored with this language definition and is referenced in the ARCHDEF by the SINC keyword.

This verify translator determines what parts are required to be built (including nested ARCHDEFs) depending on the build modes, and copies these parts into the z/OS UNIX Systems services workarea directory. A skeleton ANT xml is dynamically customized according to the build script and the parts built in the workarea using ANT. The class files are passed to the JAVA/JAVABIN language translators to store the class files back into SCLM. J2EE objects generated such as a JAR, WAR, or EAR are passed to the ARCHDEF language translator (J2EEOBJ) to be stored back into SCLM.

J2EEOBJ — J2EE ARCHDEF Translator

(SCLM language translator for Java/J2EE generated outputs)

This is the final build translator invoked as part of the ARCHDEF build process. This translator determines what J2EE objects (JAR, WAR, EAR) were previously generated during the build of the ARCHDEF in translator J2EEANT and copies these objects into SCLM with the generated short name provided.

21.2.2 Java/J2EE type requirements

The following types, as shown in Table 21-1, have to be defined and their corresponding data sets must be allocated to store Java/J2EE projects in SCLM.

Table 21-1 SCLM type requirements to store Java/J2EE projects

Type	Description
J2EEBLD	Build scripts are stored in this type.
ARCHDEF	ARCHDEF files which define Java/J2EE projects are stored in this type.
JAVALIST	Listing outputs from Java class compilations are stored in this type.
J2EELIST	Listing output from J2EE ANT builds are stored in this type.
JAVACLAS	**.class* files generated from the Java build process are stored in this type.
J2EEJAR	JAR files generated from the build process are stored in this type.
J2EEWAR	WAR files generated from the build process are stored in this type.
J2EEEAR	EAR files generated from the build process are stored in this type.
Source types	Various source types need to be defined. Project per type is the recommendation to avoid the name conflicts.

21.3 Setting up your IDE environment

There is no special setup required to use SCLM Developer Toolkit to manage Eclipse's projects. Any project in the Eclipse's workspace can be configured/mapped and use SCLM as SCM provider (see 21.5, "Usage scenarios: End-to-end usage scenario of a typical Java application development lifecycle" on page 531 for a description of how to register SCLM as an SCM provider).

Special considerations

Special considerations about the implementation details of SCLM Developer Toolkit are discussed in this section.

Long name, short name translation

Mainframe data set member name convention is much more restrictive than the file name convention on a distributed platform such as Windows environment. For example, the maximum length of a data set member is 8 characters with many more restrictions on permitted characters. For this reason, SCLM Developer Toolkit provides a long name, short name translation framework.

A long name of a file typically represent the entire path relative to the project root. For example, the *image1.gif* file in *My non-Java project* has a long name of *images/image.gif* in Figure 21-2. This is true for most of the cases except for Java/J2EE projects with source folders being defined elsewhere from the project root. In such cases, SCLM Developer Toolkit puts tags around the folder names that are not part of a Java package so that they can be ignored during the build process as shown in Figure 21-2.

This annotation of source folders is important for building Java/J2EE projects correctly because the folder structure corresponds to the Java/J2EE package structure of their source members.

Archived

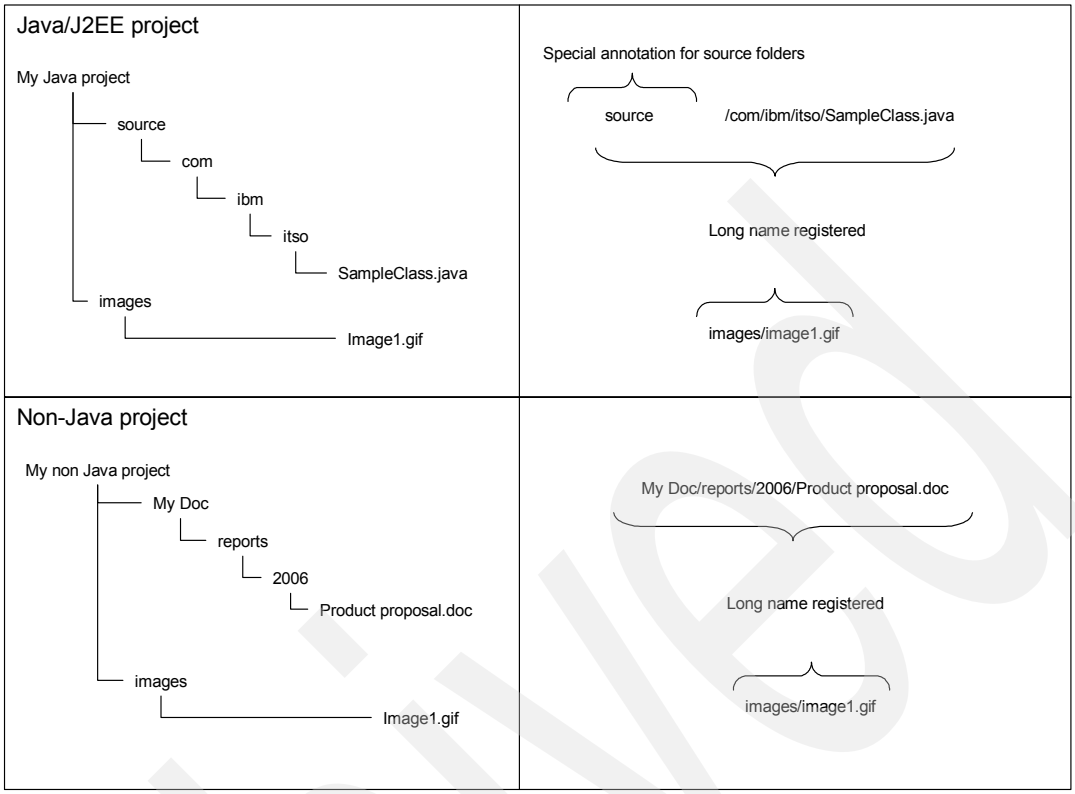


Figure 21-2 Long name, short name translation treatments

Non-ASCII UNICODE file name support

When a file in an SCLM managed project contains non-ASCII characters, its name is translated into a special format using Base64 encoding before being applied the usual long name, short name translation[1]. This is because there is no direct support for UNICODE characters on the mainframe system, hence no support is provided to store UNICODE file names.

This requires special consideration when such a file is copied to a directory under UNIX System Service (USS). A typical scenario might be when you are considering writing your own build script to perform some actions against the file stored in an SCLM repository. When the file is copied to a directory in USS, its name is still in the special format and it is not the original UNICODE name. When you are writing a script to manipulate such files, special care must be taken not to use their original UNICODE names.

21.4 Working with the IDE view (Eclipse projects)

In this section, we describe a set of SCLM DT actions that are available when you are working with a project in an Eclipse workspace. A detailed description of each action can be found in the online help (accessible via **Help** → **Help Content** → **IBM SCLM Developer Toolkit Online Help**). Furthermore, there is a usage scenario that describes which actions to use during a Java/J2EE development lifecycle in 21.5, “Usage scenarios: End-to-end usage scenario of a typical Java application development lifecycle”.

[1] The same treatment is applied with a file name containing special characters under USS; for example, brackets and dollar signs.

The actions provided by SCLM DT are available via pop-up menu (right-click against the resources managed by SCLM DT) as shown in Figure 21-3.

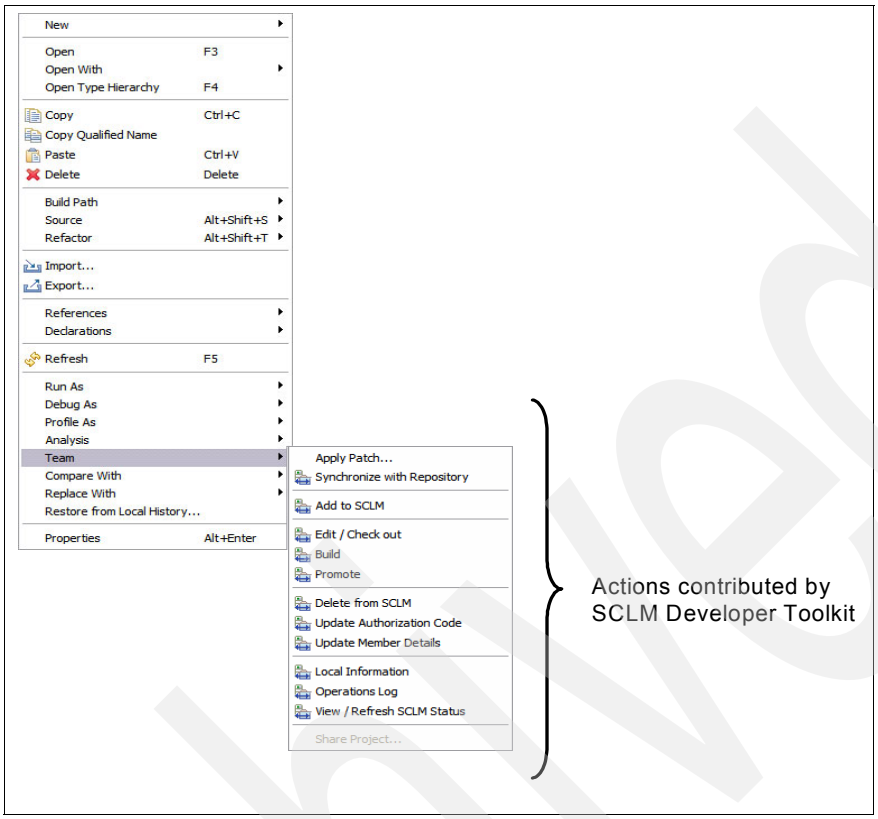


Figure 21-3 SCLM Developer Toolkit actions made available via context menu

Table 21-2 shows the summary of actions available.

Table 21-2 Summary of actions available against SCLM managed project

Action	Description
Logon	Authenticates with the server. Presents a dialog to specify the login information as well as provides opportunity to change password if required.
Register project to SCLM	Registers and selects SCLM to manage resources in a project in Eclipse environment. Once the project is registered, all the action described here become available against resources in the project.
Un-map project from SCLM	When you finished working with a project being managed by SCLM, you can un-map the project (or unregister SCLM as repository provider) so that all SCLM specific actions will not be available against the resource of the project.
SCLM Import	Imports members in SCLM to the selected project. You can specify the members to import from SCLM using various filters or ARCHDEFs. By importing members from SCLM, you are getting local copies of the members in the selected project. However, members are not locked on the host.
Add to SCLM	Adds additional resources to SCLM.

Archived

Action	Description
Check out Member	Places an exclusive lock against the selected member and makes it available for editing in the local project. The latest copy of the member is copied from the repository by default.
Check in Member	Once finished making all the changes required, the contents of the member should be saved back in the repository. This action saves the contents of local resource in the repository and removes the exclusive lock placed on the member via check out action.
Cancel Edit / Un-lock	This action allows you to remove the exclusive lock being placed for the selected member.
Delete Member from SCLM	This action deletes the member from the repository.
Get SCLM Status	Retrieves the latest information from the repository for the selected member.
Synchronize Project with SCLM	Compares the status of all members in the selected project and the members in an SCLM repository. If status differences exist, then they are reported in the synchronize view.
Version Information	Provides version information.
Operations Log	Allows you to browse and save operational log of all actions being performed.
Local Info	Displays the cached SCLM information about the selected resources.
Update Authorization Code	Allows you to update the authorization code of the selected member.
Update Member Details	Allows you to change the language type and change code for the selected member.
Upload Jar Files	Allows you to upload local files to a directory under USS. This feature can be useful when moving all dependent JAR files from PC to the host.
View Cached Project Information	DT caches the project information. This action allows you to view the cached project information
Update Project Information	Retrieves the latest project information from the host and refresh the cached project information.
Generate Database Content Reports	This function allows you to select various criteria in order to extract information from SCLMs meta data tables.
Compare with latest version	Allows you to compare the contents of the selected local file and the latest version in the repository.
Compare with different version	Allows you to compare the contents of the selected local file and different versions of the member in the repository (versioning feature must be turned on for this to work correctly).
Replace with latest from SCLM	Replaces the contents of selected resource with the contents of corresponding resource in the repository.

Archived

21.5 Usage scenarios: End-to-end usage scenario of a typical Java application development lifecycle

In this section, we describe how to manage a typical Java project using SCLM Developer Toolkit.

21.5.1 Sample Java project

A typical Java project consists of several components, each providing a distinct logical grouping of functionality, and they are often developed by different teams of developers in parallel or different stages of the project as shown in Figure 21-4. The latest software engineering literature promotes methodologies that allow for the iterative development of products to meet the need of ever increasing complex software solutions.

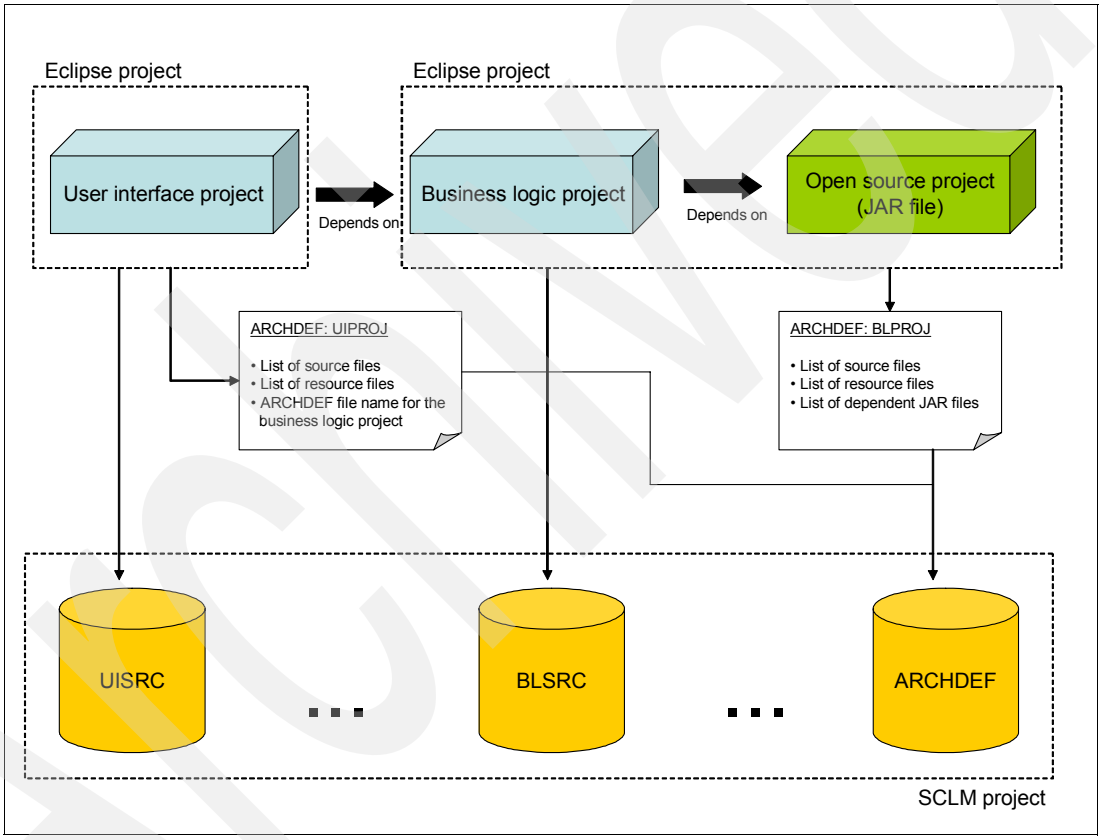


Figure 21-4 A typical Java project configuration

From Eclipse's development perspective, the result translates into a number of inter-related projects, and each project represents a logical unit of functionality. Figure 21-4 illustrates a typical configuration of a Java project which consists of two Eclipse projects.

One project is concerned with the implementation of user interface components, and another project is concerned with the implementation of business logic. To implement the business logic, it depends on an open source library, which was downloaded from the Web. The UI project depends on the functionality provided by the business logic project.

Figure 21-5 shows Eclipse's package explorer view, which lists the resources of the two projects described before. *BusinessLogicsProject* provides three classes, *Employee*, *EmployeesDatabase* and *SalaryReview*, which implement the logic to manage the employees information in a database. *utility.jar* is an open source library used to implement part of the functionality of the project. UI project contributes one class, *EmployeeManagement*, which implements a simple user interface to interact with the employee information using the functionality provided by the business logic project. Another file, *employee.info*, is used to store all employee information.

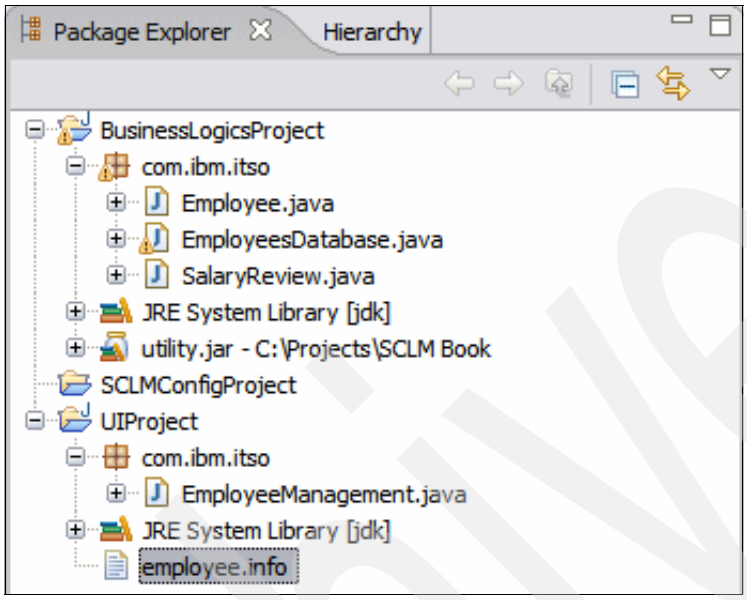


Figure 21-5 Eclipse projects overview (package explorer view)

The dependency between two projects is described in the properties associated with the UI project. That is, the business logic project is included is included in the Java build path of the UI project (right-click on the UI project, select **Properties**, select **Java Build Path**, and select **Projects**. The reference to the business logic project is shown).

21.5.2 Migrating projects to an SCLM repository

In this section, we describe the steps required to migrate Java projects into an SCLM repository.

When migrating a set of interrelated Java projects into an SCLM repository, IBM recommends that you store all the resources in a single SCLM project, and group the resources of a Java project (that is, Eclipse project) using SCLM types. For example, we store all the resources of two projects described in the previous section in an SCLM project, and group the resources using two types (BLSRC and UISRC). Table 21-3 summarizes the type and language of all resources in the project.

Archived

Table 21-3 Summary of resources and corresponding SCLM type and language

Project name: business logic project	**ARCHDEF name: BLPROJ**	
File name	**Type**	**Language**
Employee.java	BLSRC	JAVA
EmployeeDatabase.java	BLSRC	JAVA
SalaryReview.java	BLSRC	JAVA
utility.jar	BLSRC	J2EEBIN
.project	BLSRC	J2EEPART
.classpath	BLSRC	J2EEPART
Project name: UI project	**ARCHDEF name: UIPROJ**	
File name	**Type**	**Language**
EmployeeManagement.java	UISRC	JAVA
employee.info	UISRC	J2EEBIN
.project	UISRC	J2EEPART
.classpath	UISRC	J2EEPART

It is important to understand the usage of an ARCHDEF file in the context of Java/J2EE projects. An ARCHDEF file is used to describe the resources of a Java project, dependencies to other SCLM managed Java projects, and how to build the project using SCLM build framework with DT extensions to support Java/J2EE projects.

Tip: SCLM Developer Toolkit provides an option to store Java/J2EE source members in either EBCDIC or ASCII. To store in EBCDIC, select the language type of **JAVA**. Otherwise select **JAVABIN** to store in ASCII format. The build process is able to handle source members stored in either format and able to perform translation appropriately.

Describing the resources of a Java project

Because an SCLM project and Java projects are organized as one-to-many relationship (that is, one SCLM project may have multiple Java projects), an ARCHDEF file is used to describe the resources of a Java project. An ARCHDEF file is a logical grouping of SCLM resources for a particular Java project, and the build process is driven by the specification. The output from the build process against such an ARCHDEF file is a JAR file containing all class files generated and any other resources specified in the ARCHDEF file. It is possible that one member belongs to multiple ARCHDEF files (that is, multiple Java projects).

Describing the dependencies

Dependencies between Java projects can be specified using the *INCL* keyword in the ARCHDEF file. When the build process is invoked against an ARCHDEF file that has dependencies for other ARCHDEF files (Java projects), SCLM checks to see if the dependent projects are also up to date (that is, built successfully). If a dependent project is out-dated, it invokes the build process against it before building the specified ARCHDEF members. For example, when you invoke the build process against the UI project (UIPROJ ARCHDEF file), SCLM recognizes that it depends on the business logic project (BLPROJ ARCHDEF file). If any resources of the business logic project are outdated, it invokes the build process against the project first.

Building the project in SCLM

SCLM requires a number of parameters to build a Java project. One such parameter is the name of anSCLM member which contains the name of a base ANT build script in SCLM and a set of parameters to be used by the ANT build script. We describe the contents of a build script later in the chapter.

In the rest of this section, we describe the steps required to migrate projects to SCLM.

1. Choosing SCLM DT as the repository provider:

   Eclipse workbench provides a generic framework (extension point) for SCM vendors to work with Eclipse based projects, and DT toolkit uses the extension point to provide SCLM specific functionality. From the context menu, select the **Share Project** wizard, which appears under the **Team** menu.

   Figure 21-6 and Figure 21-7 shows pages from the SCLM Share Project wizard, where the location an SCLM repository, name of the SCLM project, alternate project, and the user's development group are specified. This wizard assumes that the SCLM project is set up for your project and ready for migration.

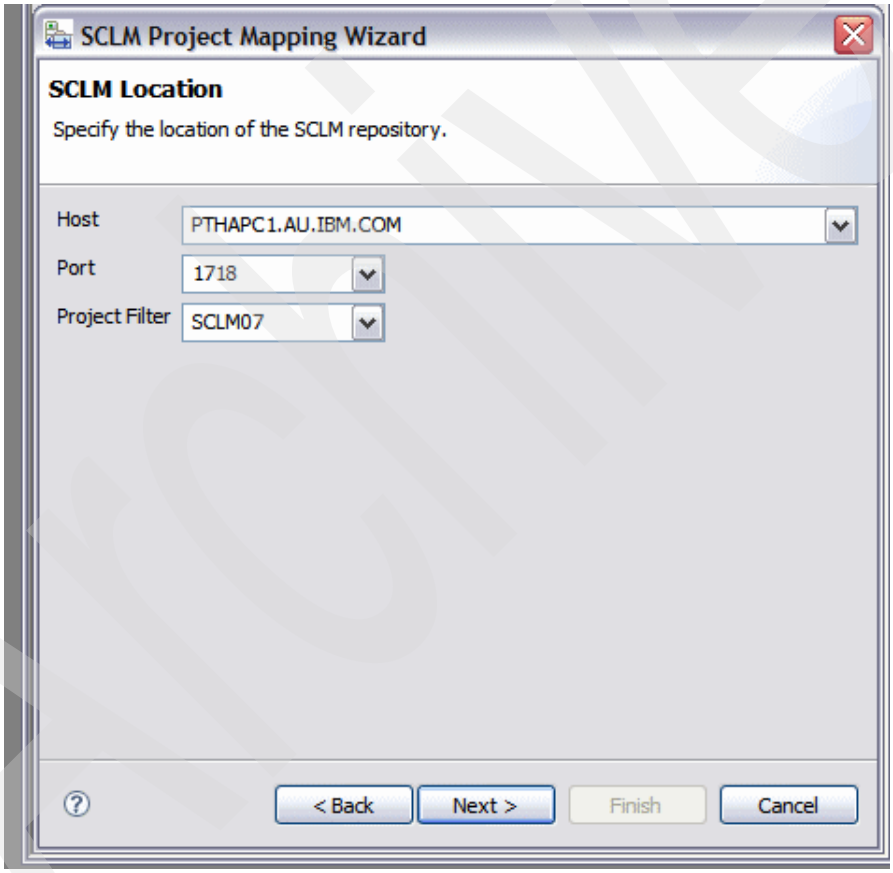


Figure 21-6 Sharing Eclipse project using SCLM DT (1 of 2)

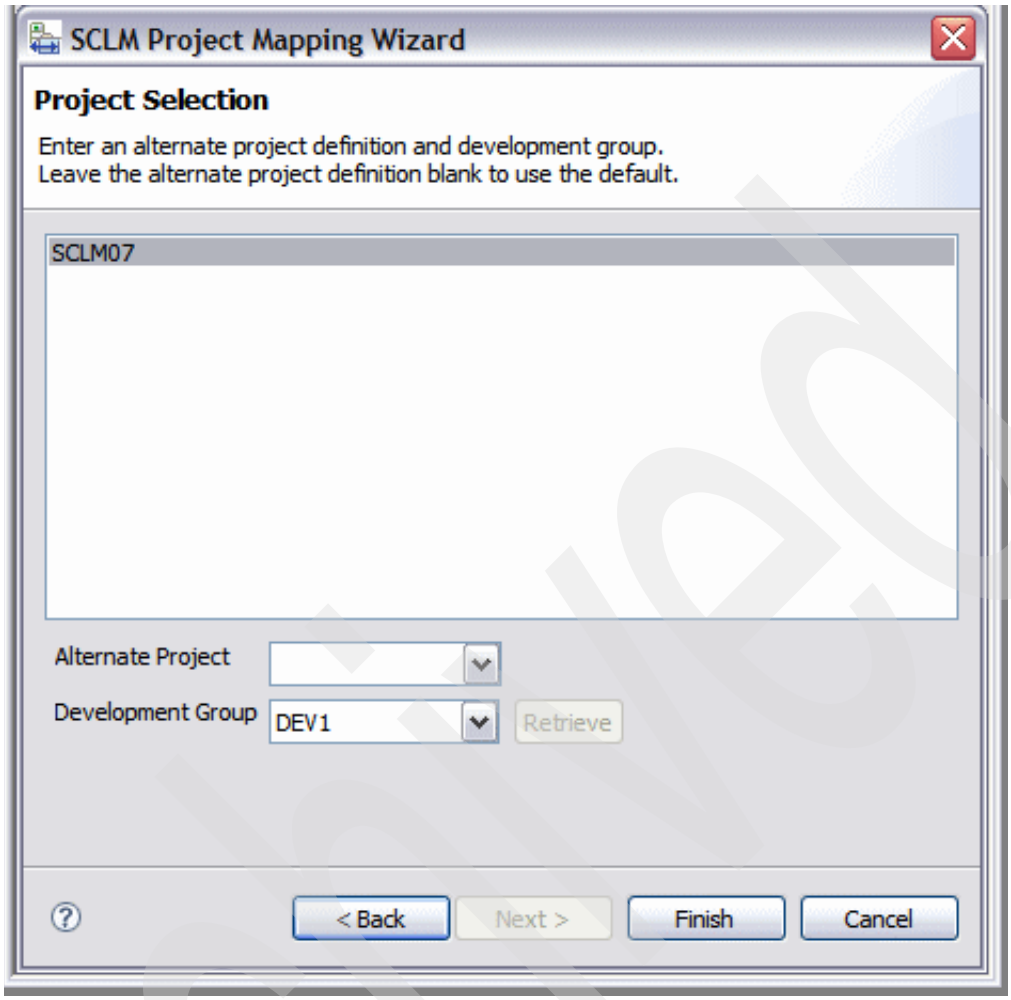


Figure 21-7 Sharing Eclipse project using SCLM DT (2 of 2)

Archived

2. Migrating members to SCLM:

   Members in the shared Eclipse project can be migrated to an SCLM repository via a **Migrate to SCLM** action, which is accessible via the context menu under **Team** menu. In the first page of the wizard, members to be migrated to SCLM are selected. In the second page of the wizard, as shown in Figure 21-8, SCLM language and type are specified for the members to be migrated to SCLM. In addition to specifying the language and type, *authorization code* and *change code* can also be specified in this page.

   Two additional options are available from this page. Firstly, the Enable Forced Migrate option allows you to overwrite existing members in SCLM. If you do not check this option, and you attempt to migrate a file that already exists in SCLM, the migration will be unsuccessful, and SCLM will retain the member exist in the repository already. Since this option allows you to overwrite any members in SCLM, IBM discourages to use this option, especially migrating a large number of members. Secondly, you can specify the **Batch Migrate** option which migrates the selected member in batch mode.

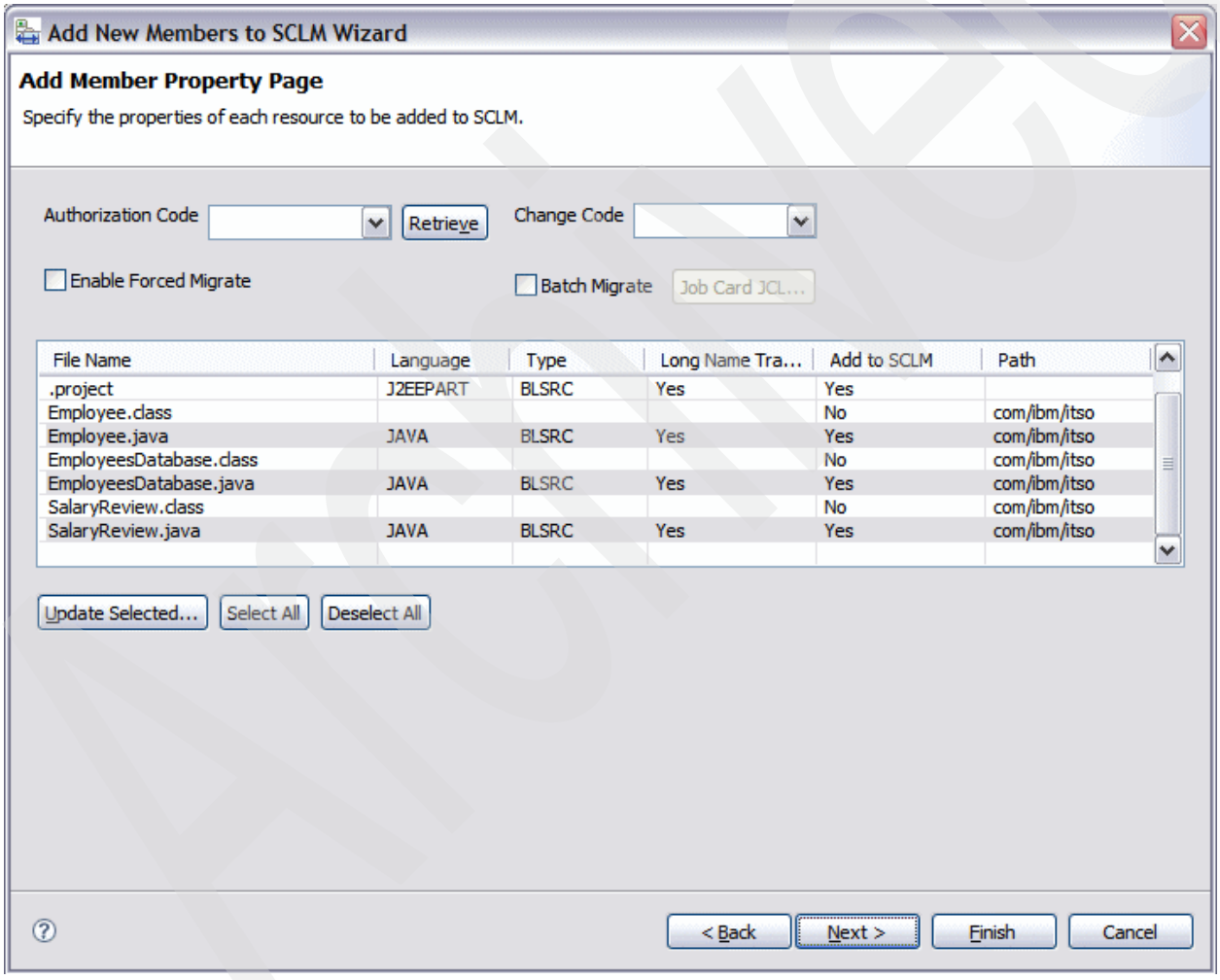


Figure 21-8 Specifying language and type for members to be migrated to SCLM

In the third page of the wizard, as shown in Figure 21-9, details of ARCHDEF for the project are specified.

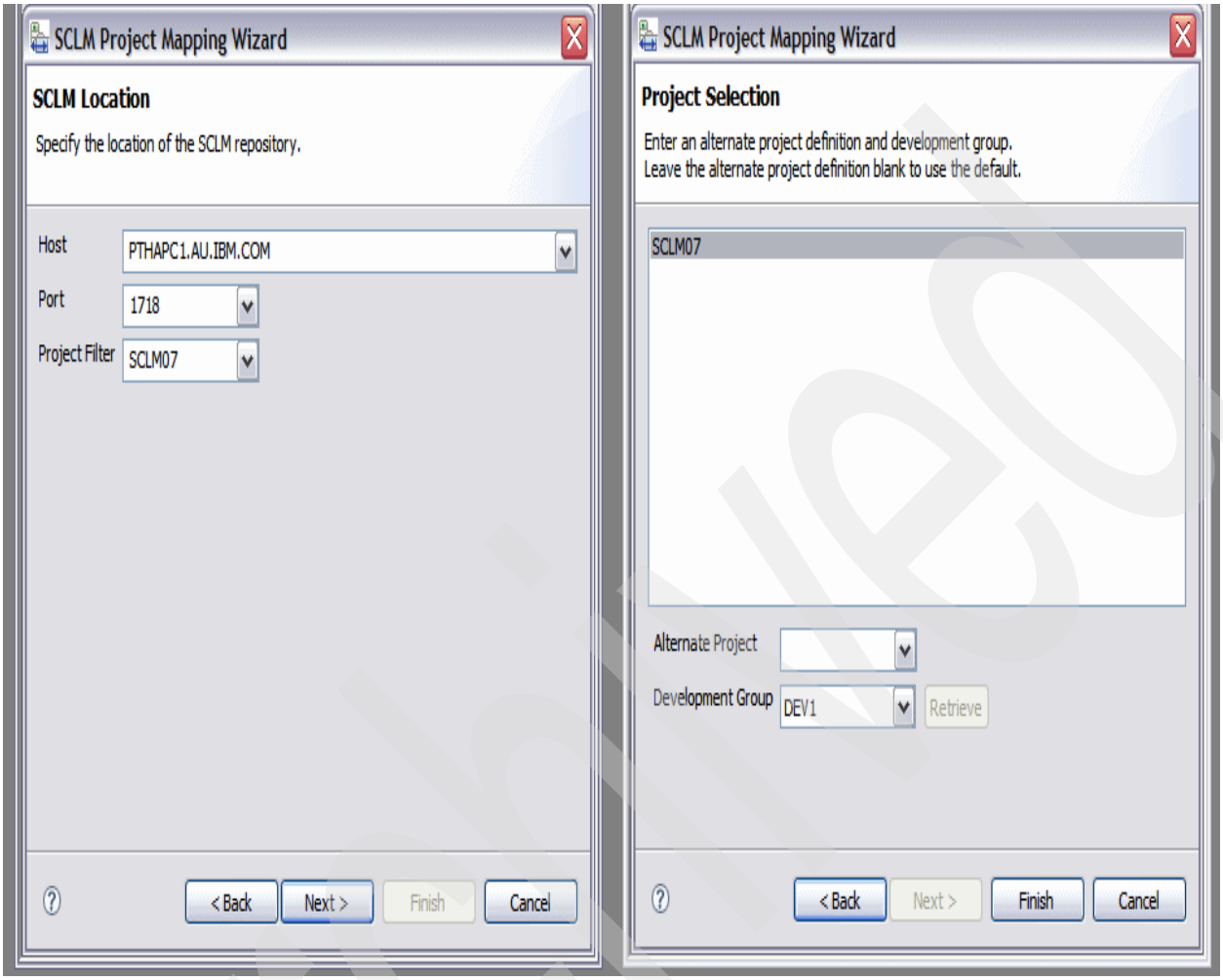


Figure 21-9 ARCHDEF file information for the project

As noted earlier, ARCHDEF file captures the essential information needed to manage your Java project. When ARCHDEF name and ARCHDEF type are specified, all members included in the migration are included in the specified ARCHDEF file. If the ARCHDEF does not exist, a new member is created in the specified type. If the member exists already, then new members are added without duplication.

Optionally, you can also choose to generate a new build script for the project. By selecting the check box for the option, you are presented with the dialog to enter the information required to generate the build script for your project, as shown in Figure 21-10. It is possible to generate build scripts for Java projects, Web projects, EJB projects, and EAR projects. For our example, we choose the Java Project option. The generated build script shares the member name with the ARCHDEF member which represents your project and stored in type J2EEBLD.

In addition, you are able to select an option to include the build map information in the resultant archive file (JAR, WAR, and EAR files). This is useful as the map contains an inventory of the components included in the JAR/WAR/EAR file during assembly. It is also possible to include the source file in the resultant archive file.

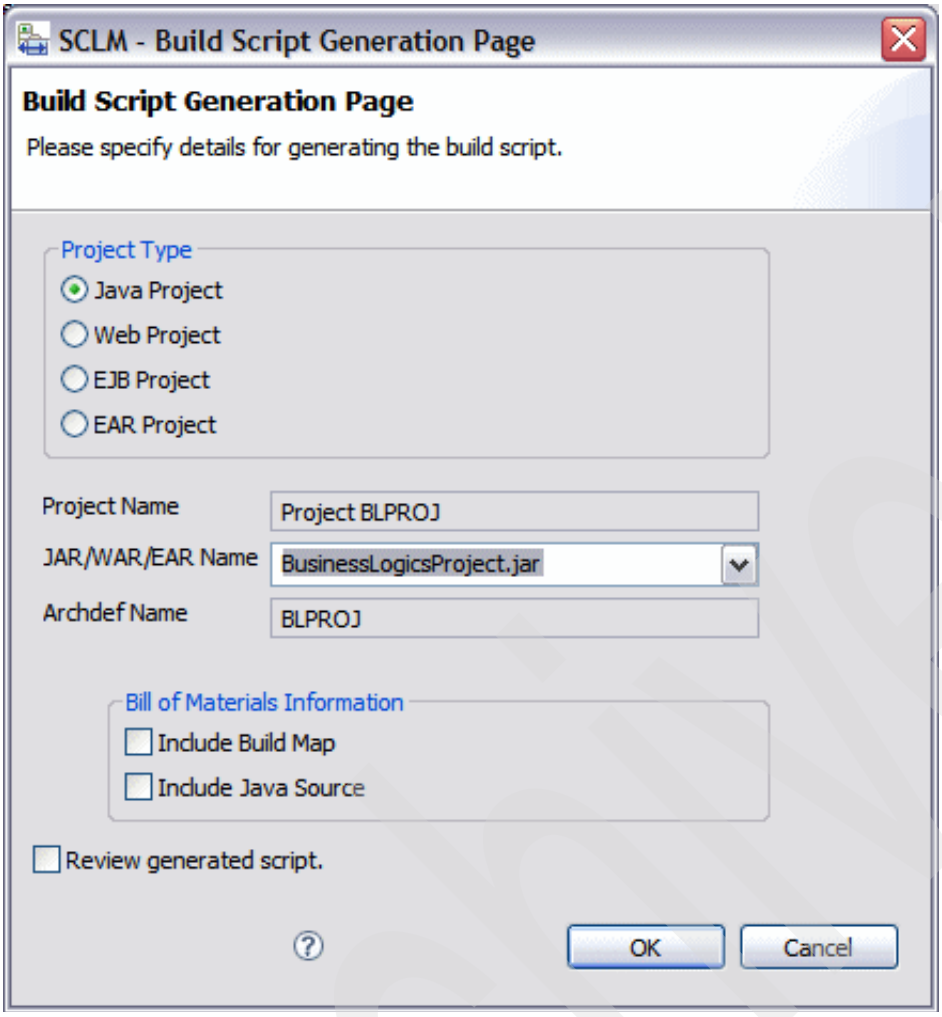


Figure 21-10 Build script generator for the selected ARCHDEF file

Tip: Some parameters that are used by the build script generator can be customized, since these parameters can vary between installations. See *Build Script Parameter Options* under **Team → SCLM Preferences → Build Script Options**.

Once a project is shared using DT, status information of members are displayed using decorators. Figure 21-11 shows projects being decorated by DT. The decorator against *BusinessLogicProject* indicates that it is shared with an SCLM project called *SCLM07*, with project alternate of *SCLM07* and the user development group of *DEV1*. All members of the project indicate that they are managed by SCLM (no decorator when a member is managed by SLCM and no further status information to display), except that *Employee.java* is being marked as checked out.

In comparison, *UIProject*'s members indicate that they are not managed by SCLM despite the fact the project itself is shared with an SCLM repository. All members of the project need to be migrated to SCLM following the steps described above.

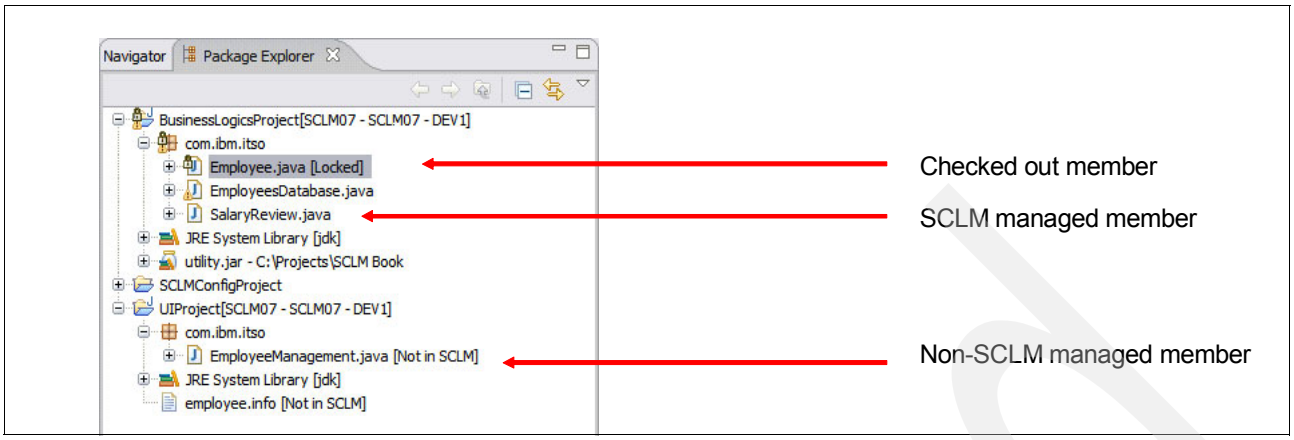


Figure 21-11 DT project member decorations

A summary of status of members of an SCLM managed project can be viewed using *Local Information* action which is accessible via the context menu. It displays a list of status information of the selected members in the project. Figure 21-12 shows an example of status summary after migrating the members to SCLM. Columns in the table can be sorted by clicking on the table headings. A previous version of a member can be viewed by right-clicking on a member in the table.

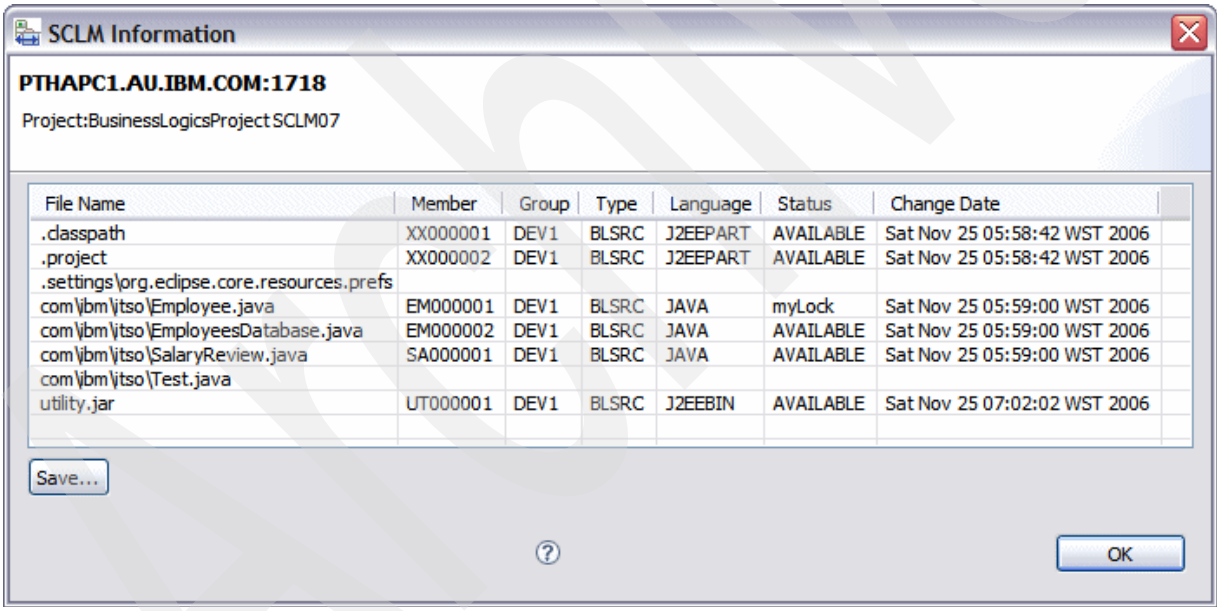


Figure 21-12 Summary of SCLM status of members in the selected project

21.5.3 Day-to-day activities by a single user

This section covers the day-to-day activities performed by a single user.

Locking members (check-in and check-out)

SCLM implements the pessimistic approach to keeping the integrity of the members stored in the repository. That is, it requires an explicit lock on a member before it can be edited[2]. Follow these steps in order to avoid the possibility of integrity problems:

1. Check out the member before start editing the file (**Team** → **Edit/Check out**). DT decorates the checked out member visually to distinguish it from the rest of members in the project as shown in Figure 21-13.
2. Make the changes to the checked out member.

Figure 21-13 DT decorates the checked out member

3. Compare with the latest copy in the SCLM repository to make sure the changes made to the checked out member are correct (**Compare with** → **Latest from SCLM**). This invokes the Eclipse's compare editor which highlights the differences between the local file and the latest copy in the repository.
4. Save (check in) the changes back to the repository (**Team** → **Check-in**).

Deleting a member from SCLM

When the member is no longer required so that it should be deleted from the repository. The member can be deleted from the repository by invoking *Delete action* against the unwanted member in the project (**Team** → **Delete from SCLM**). DT provides a number of options with the delete action which are summarized in Table 21-4.

Table 21-4 Options for delete action

Item to delete	Explanation
Member	Deletes the data set member which represents the member being deleted. Unless accounting record is deleted, SCLM repository considers the member still exist in the repository although the contents of member is no longer exist when this option is selected.
Accounting record	Accounting information associated with the selected member. If the accounting record is deleted, the member does not exist as far as SCLM repository is concerned.
Build map	Build map associated with the selected member if the member was built previously.
Workspace file	Local copy of the member being imported from SCLM.

[2] SCMs typically implement the optimistic approach today in which multiple users are allowed to modify a file concurrently and the changes made by different users are merged when the file is checked back in to the repository. If a SCM is unable to merge the changes automatically, it asks the user to resolve the conflicts manually.

Archived

Delete action only deletes the selected member at the specified development group. For example, if a member exists in two development groups, they must be deleted at both levels explicitly. Furthermore, it does not delete the member information from an ARCHDEF file that contains the reference to the deleted member. Such ARCHDEF files must be updated manually to reflect the changes.

Tip: DT does not currently provide an action to change the name of a member being stored in the repository. This is true for both short names and associated long names. Consequently, when the name of a file has to be changed (for example, the Java class name or package name), then it must be deleted from the repository first and re-migrated.

Refreshing status information

It is possible that the status of an SCLM managed member has changed since the member was imported to an Eclipse project. For example, a team member might work on a member and promote it up to a different development group. Consequently, certain actions might fail due to the inconsistency between the status of SCLM member and its local copy in the Eclipse project. To recover from such failures or simply to view the latest status of a member in the repository, *View / Refresh SCLM Status* action is useful (**Team → View/Refresh SCLM Status**). It reports any inconsistency between the status and provides the option to update the status of local copy. If no inconsistency is found, the action displays a dialog which displays the detailed status information of the selected member in the repository.

Comparing with different versions

If the versioning feature is turned on, it is possibly to retrieve a list of versions for a member and compare the differences (**Compare with → Different Versions...**). The action presents a dialog to select groups where the versioning information should be retrieved and all the versioning information are presented in a table. The differences between the latest copy in the workspace and a selected version of the member in the repository are shown using the Eclipse's comparison editor[3].

Building members and Java projects

DT provides actions to build members individually or as a project using ARCHDEF file which describes the resources of the project. Build action against a member allows you to build a member individually. This is a simplistic approach where only the selected member is built independently without considering the dependencies to other resources. This action is useful to build members that do not depend on other resources. For example, members having the language types of *J2EEPART* and *J2EEBIN* are good candidates for the individual build.

Java source files in a Java project typically have dependencies to other Java source files in the project, JAR files, and other Java projects. For this reason, Java source members are typically built using an ARCHDEF member.

[3] Additional actions concerned with versioning are available in the SCLM view.

Some steps to consider are summarized next:

- **Making dependent JAR files available:** A Java project typically has dependencies to JAR files containing Java classes which are used by the application. For example, it is common to download an open source project library and used in a project. In the sample project described above, it uses such a library, which is called *utility.jar*. Such JAR files need to be included in the CLASSPATH environmental variable when building the project.

  JAR files used by the project can easily uploaded to a directory under USS using *Upload JAR files* action (**Team** → **Upload JAR files**). Figure 21-14 illustrates a dialog used to specify JAR files to be copied to a directory under USS. The specified directory must matched the directory where JAR files are included during the build process which is specified via *CLASSPATH_JARS* property in the build script. The default location is */var/SCLMDT/CLASSPATH* (which is specified in the preference page), however it should be customized depending on the build script used by the build process.

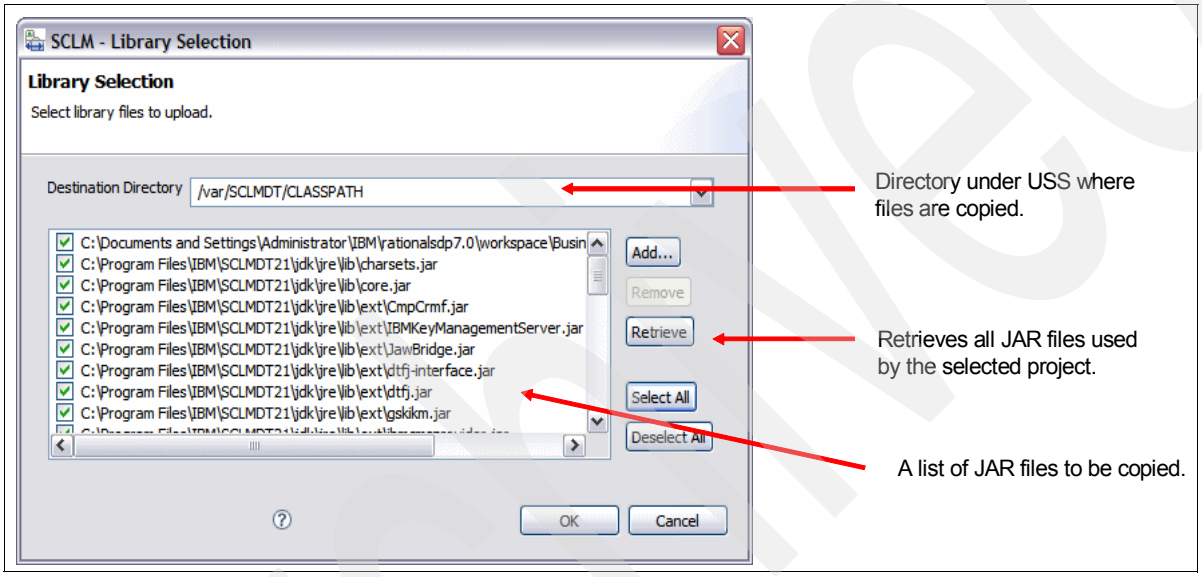


Figure 21-14 Dialog used to select JAR files to be copied to a directory under USS

- **Ensuring ARCHDEF contains the list of all members of the Java project and dependencies to other projects are specified:** All members of a Java project needs to be included in an ARCHDEF file in order to produce a JAR file for the project.

  The ARCHDEF file a Java project is typically configured during the migrate process and maintained as new members are added to the repository. However, it is possible that the file gets out of sync and does not represent the list of members. If the number of members of a project is small, then it can easily be updated manually to reflect the changes. However, it become a daunting task as the number of members of a project increases. DT provides a wizard to create a new ARCHDEF file for the project based on the members in the local project with additional references to the resources in the repository (**Team** → **Generate ARCHDEF**).

Figure 21-15 illustrates the steps required to generate a new ARCHDEF file for the selected project.

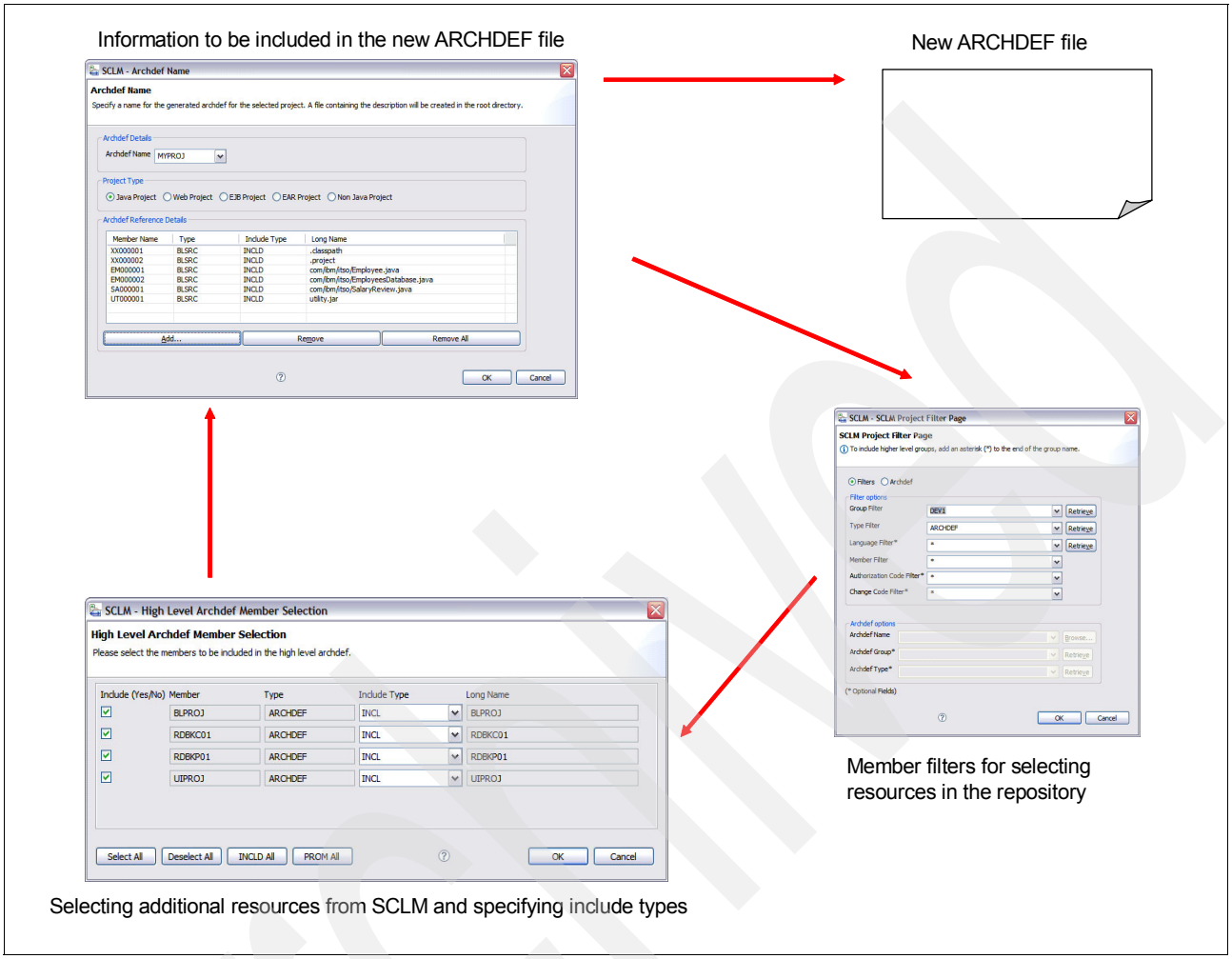


Figure 21-15 Steps for generating new ARCHDEF file for the project

The wizard pre-populate the table based on the SCLM managed resources available in the selected project and enables you to add additional resources required by the project in the repository. For example, as shown in Example 21-1, if you want to generate a new ARCHDEF file for the UI project from the example discussed above, you need to include the reference to the business logic project since UI project relies on the functionality provided by the business logic project. That is, you should include an ARCHDEF file representing the business logic project in the ARCHDEF file for the UI project using the *INCL* keyword. Example 21-1 shows a sample ARCHDEF file for the UI project.

When the dependencies to another project is specified using the INCL keyword, the build process ensures that the dependent project is also up-to-date when the project is built. That is, if any member of the dependent project is not built correctly or out-dated, then the build process builds the dependent project first before building the project.

The wizard creates a new ARCHDEF file in the root directory of the selected project with the specified name.

Example 21-1 Example of an ARCHDEF file which shows dependencies to another project in SCLM

```
*
* Initially generated on 25/11/2006 by SCLM DT V2 client
*
 LKED  J2EEOBJ           * J2EE Build translator
*
* Source to include in build
*
 INCLD XX000001 UISRC     * .classpath                                  *
 INCLD XX000002 UISRC     * .project                                    *
 INCLD EM000003 UISRC     * com/ibm/itso/EmployeeManagement.java        *
 INCLD EM000004 UISRC     * employee.info                               *
*
* References to other resources in SCLM
*
 INCL  BLPROJ   ARCHDEF   * BLPROJ                                      *
*
* Build script and generated outputs
*
 SINC  MYARCH   J2EEBLD
 OUT1  *        J2EEJAR
 LIST  *        J2EELIST
```

- **Ensuring that the build script is configured:** The ARCHDEF build framework provided by DT involves a number of different components that must be configured correctly in order to build a project successfully. Figure 21-16 illustrates an overview of the build process.

  a. The build process is invoked against an ARCHDEF file.

  b. All members in the selected ARCHDEF file are copied to a directory under USS for build.

  c. The selected ARCHDEF file includes a statement that specifies the build script parameter files to be used by the build process (*SINC* keyword).

  d. The build parameter file specifies the base build script. The contents of parameter file are overlayed at the beginning of the base build script to create a tailored build script file in a USS directory, which is used by the ANT build process.

  e. The project members are built using the build script and results are stored back in the repository.

Archived

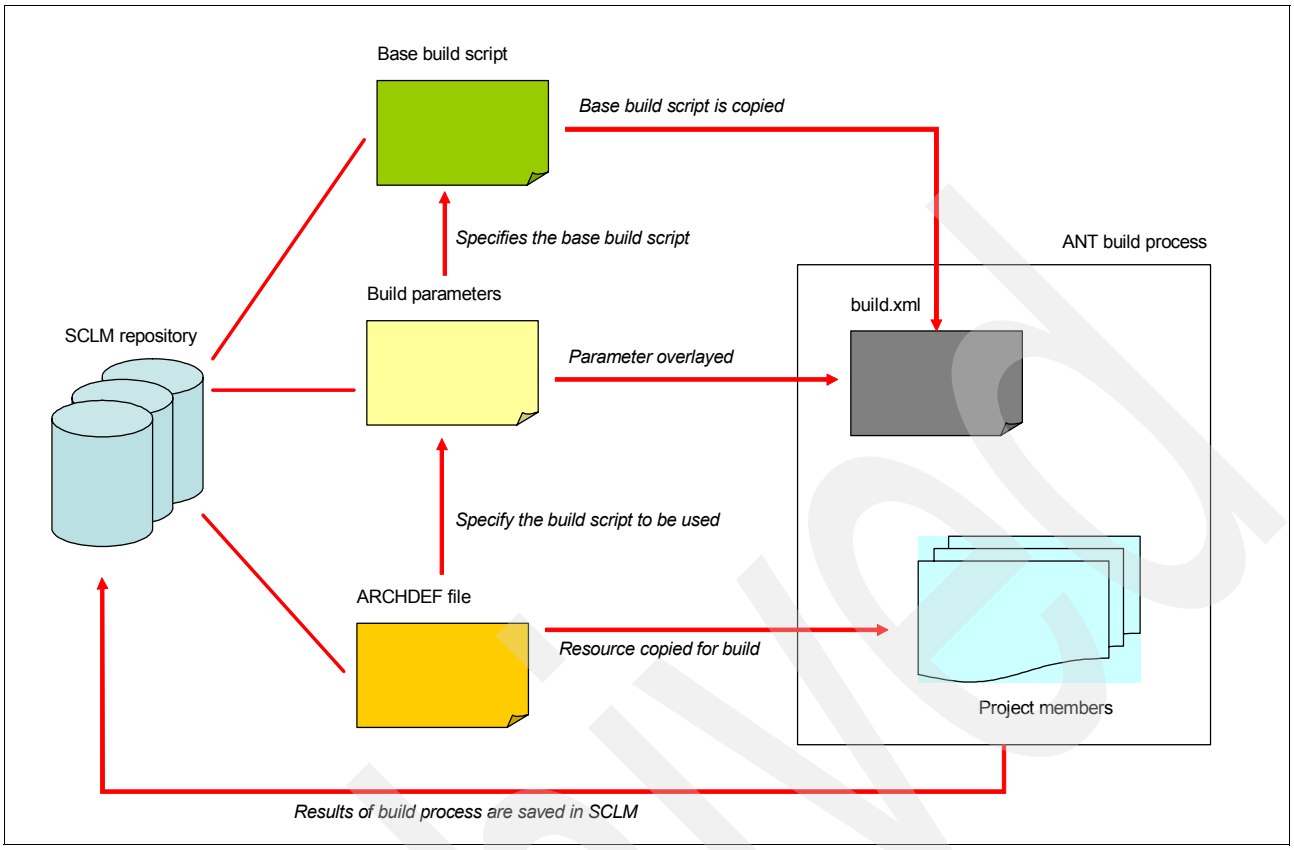


Figure 21-16 ARCHDEF build process overview

If the steps described above are followed correctly, then the build process should succeed and produce the output. A typical output from a Java project is a JAR file that is stored under J2EEJAR type in the repository.

Tip: The build process against a Java project typically takes a long time to complete. IBM recommends to build such projects in batch mode, which can be specified in the build action dialog. The status of the batch build process can be monitored using the batch job monitor feature of DT, which periodically checks the status of jobs submitted to the system. The user is notified when the job is completed and its output can be retrieved.

Promoting members and Java projects

When a project is built successfully, members of the project are eligible to be promoted to a higher group in the group hierarchy of the SCLM repository. Members of a project can either be promoted individually (**Team** → **Promote**) or as a group using an ARCHDEF file of the project (**Team** → **ARCHDEF Promote**).

Importing additional members from SCLM

Members from an SCLM repository can be imported using the *Import* action against the project (**Team** → **Import SCLM Project**). This approach is useful when importing a set of members together; for example, importing all members from a type or importing members using an ARCHDEF file.

When a small number of files must be copied to a project from the repository, importing by using an SCLM view might be more convenient. Figure 21-17 illustrates an overview of the process. From the SCLM view, it is possible to browse members in the repository without importing them to a project. Once you have found a member to import, it is possible to import a member to a project directly from the view.

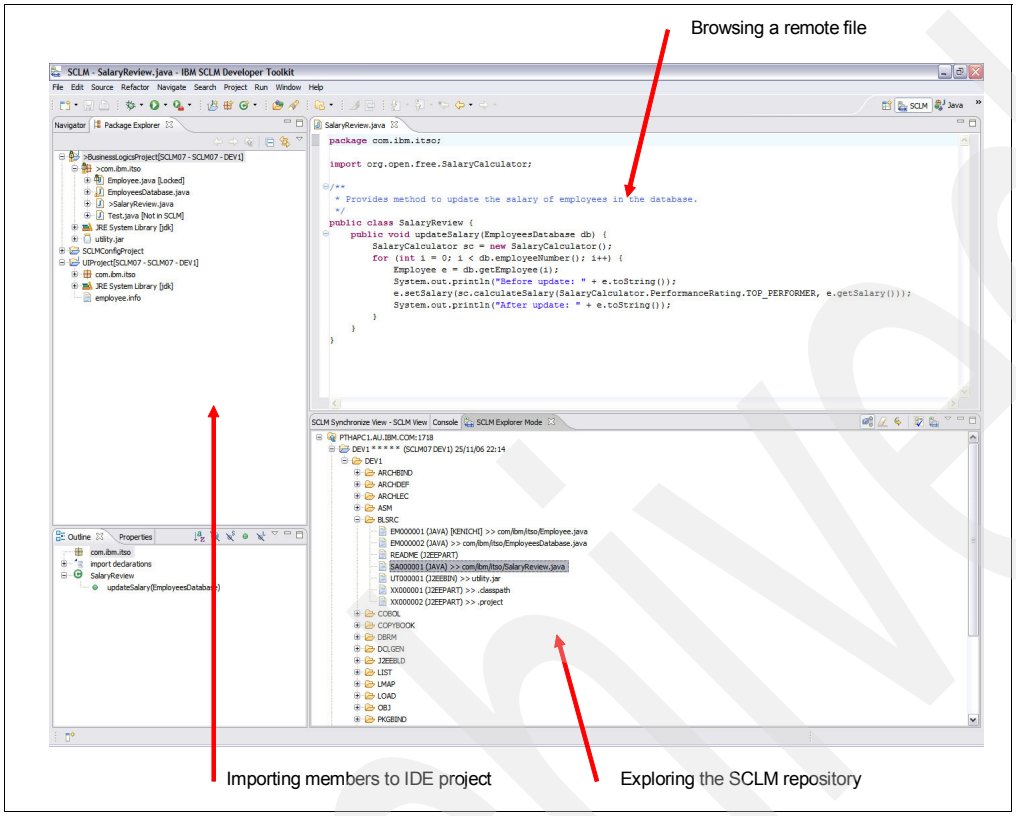


Figure 21-17 Importing members from SCLM view to IDE project

21.5.4 Using DT in a team environment

Next we discuss using DT in a team environment.

Synchronizing an SCLM managed project

Synchronize action provides an easy way to compare the status of members in SCLM versus the status of imported members in the selected Eclipse project. The action is accessible via context menu against a project which is shared with an SCLM repository (**Team** → **Synchronize with Repository**).

Since DT does not enforce the strict mapping from an SCLM concept (such as project and type) to the notion of projects in Eclipse environment, it is your responsibility to synchronize with the right set of members in the SCLM repository. The action provides two methods for selecting the set of members to compare against the members in the selected Eclipse project.

1. **Filter by parameters:** This approach allows you to specify different parameters to be used to filter members in the SCLM repository including *group, type, language, member name, authorization code,* and *change code.* This approach is useful when you have all your members in one SCLM type[4].

[4] The recommendation is to store all members of an Eclipse project under one SCLM type.

2. **Filter by ARCHDEF member:** This approach allows you to select a ARCHDEF member whose contents are read to retrieve the status information from the SCLM repository. This approach is useful when the members of your project belong to multiple types and a set of filters is not sufficient to specify all members of the selected project.

The result of action is displayed in the synchronize view, which provides two different views of the status information. Figure 21-18 illustrates IDE view which shows the status of members which have different status information compared with the corresponding members in SCLM. Table 21-5 summarizes different status information that can appear in the Synchronize view and the recommended actions to follow for each situation.

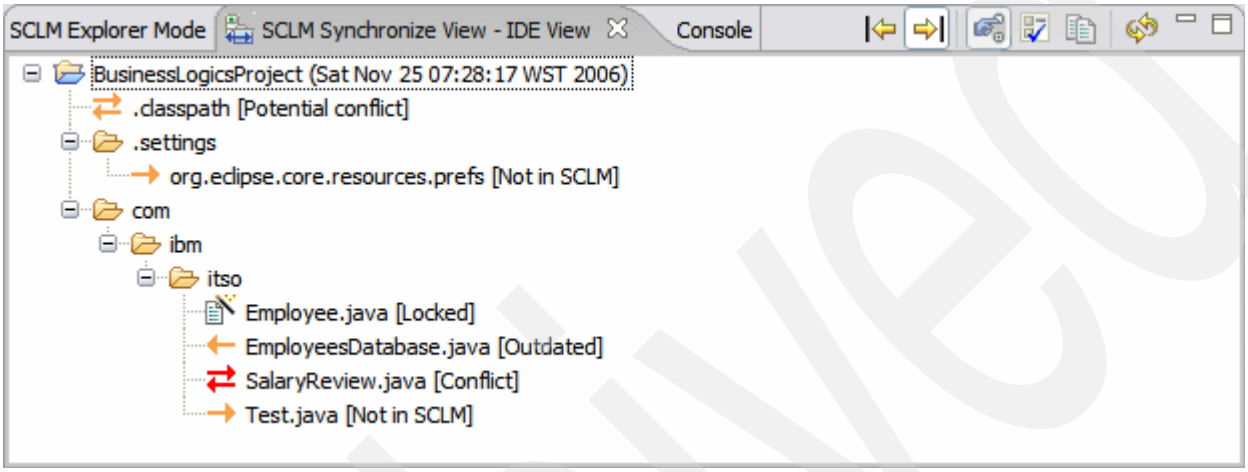


Figure 21-18 Synchronize - IDE view summarizing the status of members in workspace

Table 21-5 shows a summary of synchronization action status.

Table 21-5 Summary of synchronize action status

Status	Explanation
Not in SCLM	A new member in the selected Eclipse project which is not managed by SCLM yet. The member should be added to SCLM so that the rest of team can see your changes.
Outdated	The member in the Eclipse project is older than the member in SCLM. It typically indicates a team member has updated the member. The member should be imported so that you are working with the latest version.
Locked	The member is locked (that is, you have the exclusive right to modify this file). When you finish modifying the member, it should be checked back in.
Potential conflict	The member has been modified locally without locking the member, which may leads to the conflict status if another team member modifies the member in SCLM. It is recommended to put the lock (check out) on the member so that team members know you are editing the member.
Conflict	The member has been modified locally without locking the member in SCLM, and a team member has updated the member in the SCLM repository. It is now your responsibility to merge the changes and update the repository if necessary.

Archived

Figure 21-19 illustrates a view which presents a summary of members that are not in the selected Eclipse project. In this particular instance, the *README* member from *BLSRC* is missing in the selected member. This typically indicates that someone in the team has added a new member in the repository. Both views should be checked after executing the synchronize action to ensure that all differences are seen.

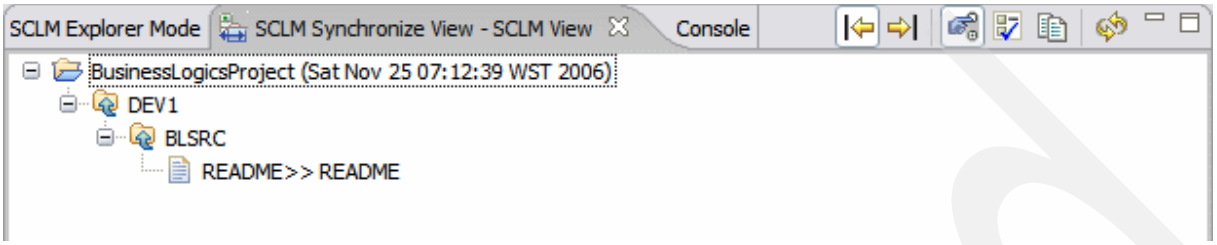


Figure 21-19 Synchronize - summary of members that are missing in the selected project

Archived

22

Deployment with SCLM Developer Toolkit

There are several types of deployment, but here we are only concerned with one: software deployment. To put it simply, software deployment is a process whereby an application is taken from a development environment and placed into an installation environment.

There are other types of deployment, including system deployment, which is concerned with the layout of infrastructure at an installation site. The UML modelling language even features a “deployment'” diagram. For our purposes, however, the term *deployment* will be used in place of *software deployment*.

The concept of deployment is commonly found in the world of Web-based services, where large scale, multi-tiered business applications need to be automatically installed and activated on application servers. Deployment is also a term at the heart of J2EE, where XML documents that expose the resources within particular components are referred to as “deployment descriptors.” In this context, deployment can be complicated, involving transactions with application servers and many steps.

While some literature talks about multiple stages of software deployment such as installation and activation, SCLM DT does not impose a particular model upon the user. A simple file copy between hosts may satisfy the needs of some. Or they may choose to interface with their existing deployment or guaranteed delivery mechanisms.

Although SCLM DT's deployment is designed expressly with the intention of transporting SCLM files, it is actually a convenient way to implement any customized task that makes some use of files in SCLM.

© Copyright IBM Corp. 2007. All rights reserved.

22.1 Types of deployment

A selection of sample scripts is provided with SCLM DT that allows you perform several standard types of deployment. These scripts work “out of the box.” The deployment process in SCLM DT is discussed further in 22.2, “Deployment in SCLM Developer Toolkit” on page 551, but before delving into the *how*, let us take a brief look at the *where*.

22.1.1 SCLM to z/OS USS

This is the most primitive form of deployment, and can be implemented by a single line of script. The chosen file is copied out of SCLM to a location on the USS file system.

This script is used as a building block for custom deployment, since it provides you with access to the file in question outside of SCLM.

SCLM to z/OS USS (remote)

The specified file is copied out of SCLM to a location on the USS file system. Subsequently, the file is transported to the USS file system on another z/OS host.

Secure

In this type of remote z/OS deployment, a “secure” service is used to transfer the file. The utilities used in the sample deployment scripts are **sftp** and **scp** from IBM Ported Tools, and will require the **sshd** daemon to be running. For **scp** and **sftp** to work in non-interactive mode, some additional configuration must be done to enable public-key exchange authentication.

Insecure

In this type of remote z/OS deployment, an “insecure” service is used to transfer the file. An example of such a service is standard FTP. Data travels unencrypted, and is therefore referred to as insecure.

22.1.2 SCLM to WebSphere Application Server

To facilitate the deployment of Enterprise Java applications (J2EE), sample scripts to install an EAR on WebSphere Application Server are given. They provide basic application installation and activation. Many J2EE projects will require further configuration via the Application Server. For this reason a large scale, complex, Enterprise application will probably require further customization of the action script. The good news is that any scripts you currently use to deploy your application into WebSphere Application Server can be reused by the action script.

WebSphere Application Server on same host (local)

This script is for installing an EAR on an Application Server that is on the same LPAR or z/OS host as SCLM DT. It allows you to specify Application Server deployment details such as cell, and node name.

WebSphere Application Server on other host (remote)

For deployment to an Application Server situated on another host, this script is appropriate. It passes the EAR to Application Server, who in turn delivers the EAR to the destination. This deployment works across different platforms, for example you can easily deploy from z/OS to an Application Server installed on Windows.

22.1.3 Custom deployment

If the deployment types listed in this section are inadequate for your requirements, you will need to create new deployment scripts. A more detailed explanation of the scripting interface of SCLM DT can be found in 22.2.1, “Scripting with XML” on page 554.

22.2 Deployment in SCLM Developer Toolkit

In SCLM DT, deployment is managed via the Enterprise Application Deployment dialog shown in Figure 22-1.

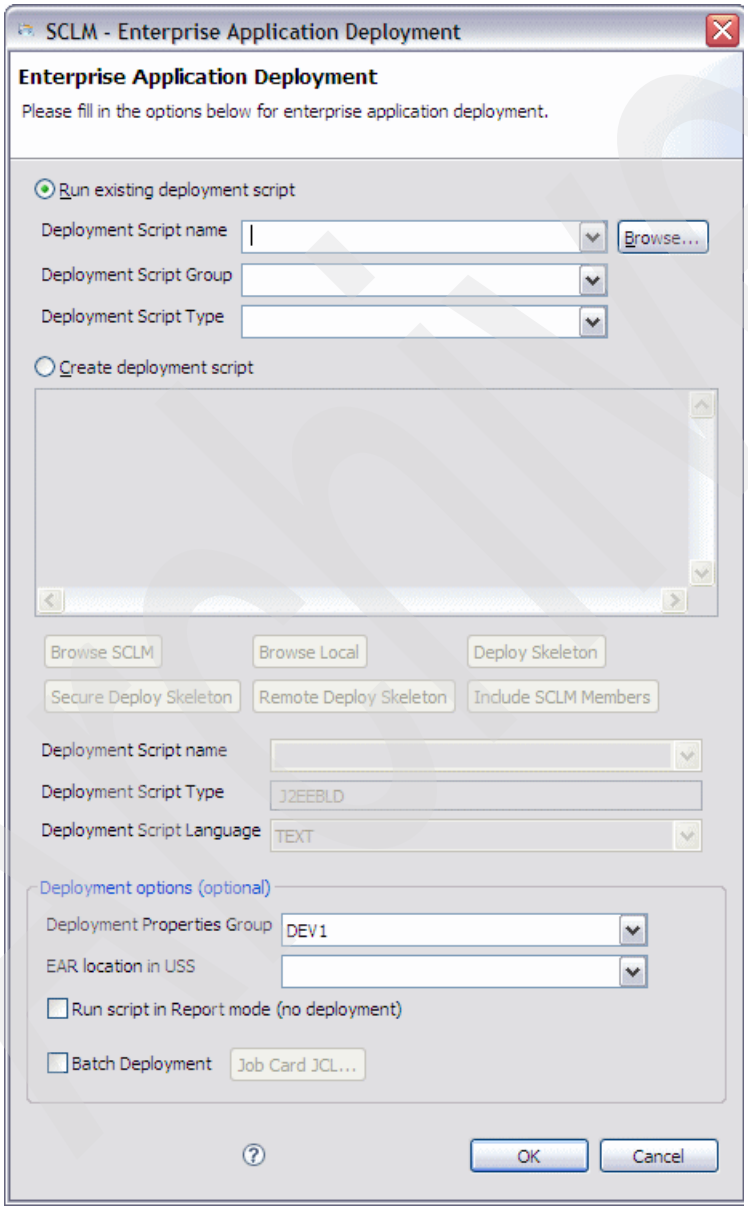


Figure 22-1 Enterprise Application Deployment panel

This dialog can be accessed via the SCLM view by right-clicking a project node, then clicking **Deploy Enterprise Application**, as shown in Figure 22-2.

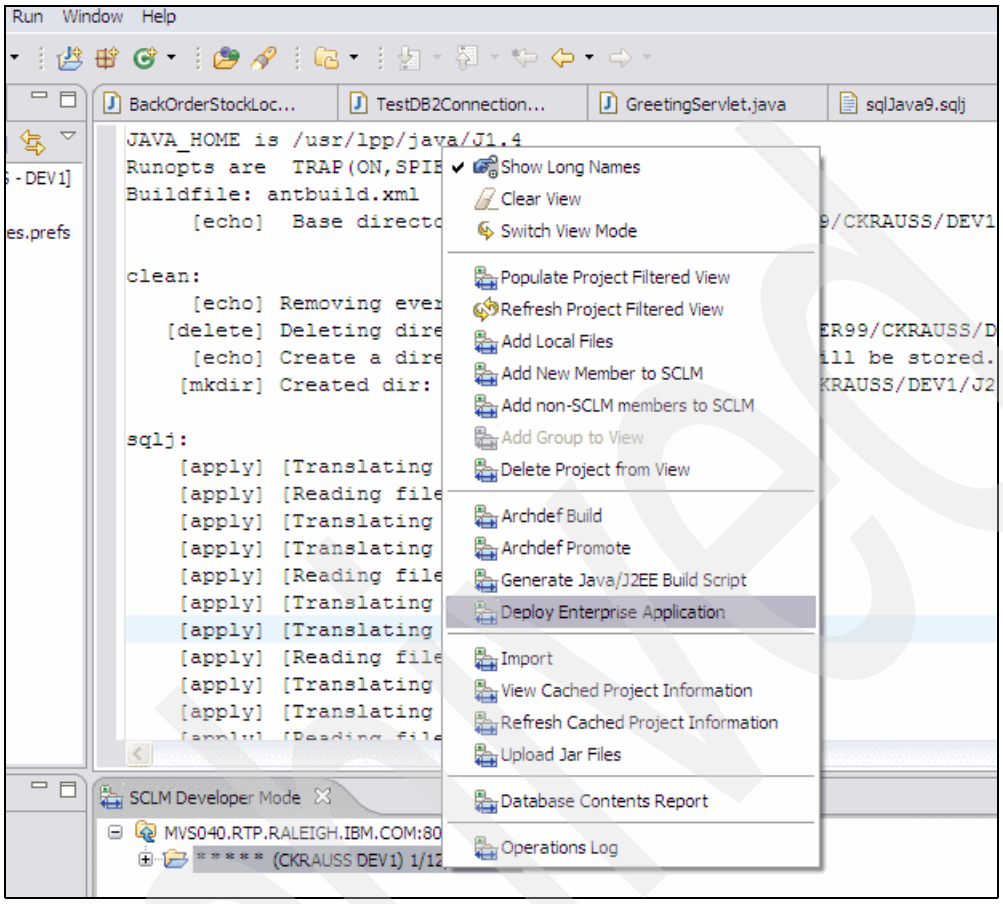


Figure 22-2 Selecting the Enterprise Application Deployment dialog from the project context menu

It is also accessible through the Eclipse project views, again through right-clicking the project node. This time, on the context menu, select **Team** → **Deploy Enterprise Application**. This is shown in Figure 22-3.

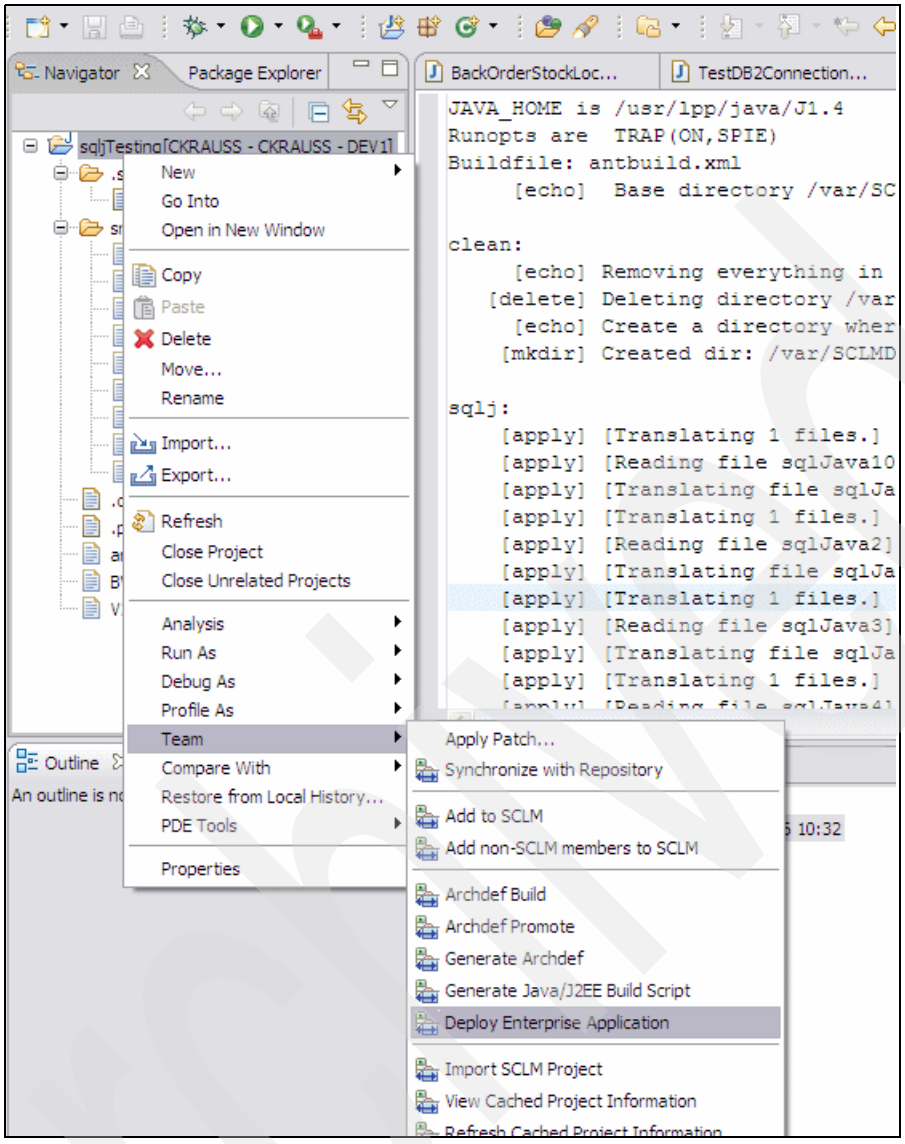


Figure 22-3 Selecting the Enterprise Application Deployment dialog from the Team context menu

The deployment dialog serves as an all-in-one control for deployment. Through the dialog, you can choose to either create a new deployment script, or run one that you have saved earlier.

In our discussion of deployment, there will be references to property scripts and action scripts. For clarity, a definition of each is provided.

Property script: An XML document consisting of a list of property tags. Each property represents some variable in the deployment process, such as the name of the application to deploy, or the host you wish to deploy it to.

Action script: An XML document consisting of a project tag under which the particulars of the operation are defined. At the time of invocation, the properties from the property script are overlayed in a predefined ANTXML section of the action script. This script can reference and call other scripts and programs it needs to get the job done.

22.2.1 Scripting with XML

SCLM DT heavily utilizes XML in its build and deployment processes. Customization invariably involves editing or creating scripts, so it makes sense to briefly discuss how XML and ANT are utilized in deployment.

XML stands for eXtensible Mark-up Language. The purpose of this language is to describe data, and the syntax is quite simple. If you are familiar with HTML but not XML, you will immediately notice the similarities. In many ways XML can be considered a superset of HTML, but they are quite distinct: XML is used for describing data, whereas HTML is used specifically for describing how to display data in a Web browser.

In any XML document, the first line will look something like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

In this example, the XML tag tells the world that the current document is XML version 1.0 using the 8859-1 ISO encoding. The encoding can vary between platforms.

The rest of the document will be made up of sets of tags or elements. Each element is declared using a pair of tags. For example:

```
<name>Joe Smith</name>
```

This specifies a name element, with the value of “Joe Smith.” It is worth noting at this point, that unlike HTML, XML preserves white-space. If we had included a tab between 'Joe and 'Smith, that tab would be preserved when the document is parsed.

Elements can also be nested. For example:

```
<person>
      <name>Joe Smith</name>
      <location>Dakota</location>
</person>
```

On ANT

For a description of ANT, we turn to the online ANT manual and review:

http://ant.apache.org/manual/index.html

“Apache Ant is a Java-based build tool. In theory, it is kind of like **make**, but without **make**'s wrinkles.”

The files ANT uses to specify the configuration details of the application to be built are written in XML. A discussion of ANT tasks, elements, syntax and semantics can be found in the online ANT manual (see the Web site listed above).

22.2.2 Running an existing script

If you have already created a deployment script, and simply wish to execute it again, you may do so by clicking the radio button, **Run an existing deployment script**. The Deployment Script Name, Deployment Script Group and Deployment Script Type text boxes become available for input, so simply specify the details of your predefined script here, and then click **OK**. Deployment will begin.

Archived

22.2.3 Creating a new script

In the event you wish to create a new deployment script, click the radio button, **Create Deployment Script**. The script editor window and associated buttons become available for use. See Figure 22-1 on page 551 for details.

Select script from local file system

If you have written a deployment script already, and it is sitting on your workstation, you can click **Browse Local.** This will prompt you with a file selection dialog that is used to locate and select the script file. Once the file is selected, it is displayed in the script editor window where you have the option of making changes if necessary.

Select script from SCLM

In the case that you only wish to make small changes to an existing script in SCLM, click **Browse SCLM**. You will be prompted with the standard SCLM DT filter member dialog shown in Figure 22-4.

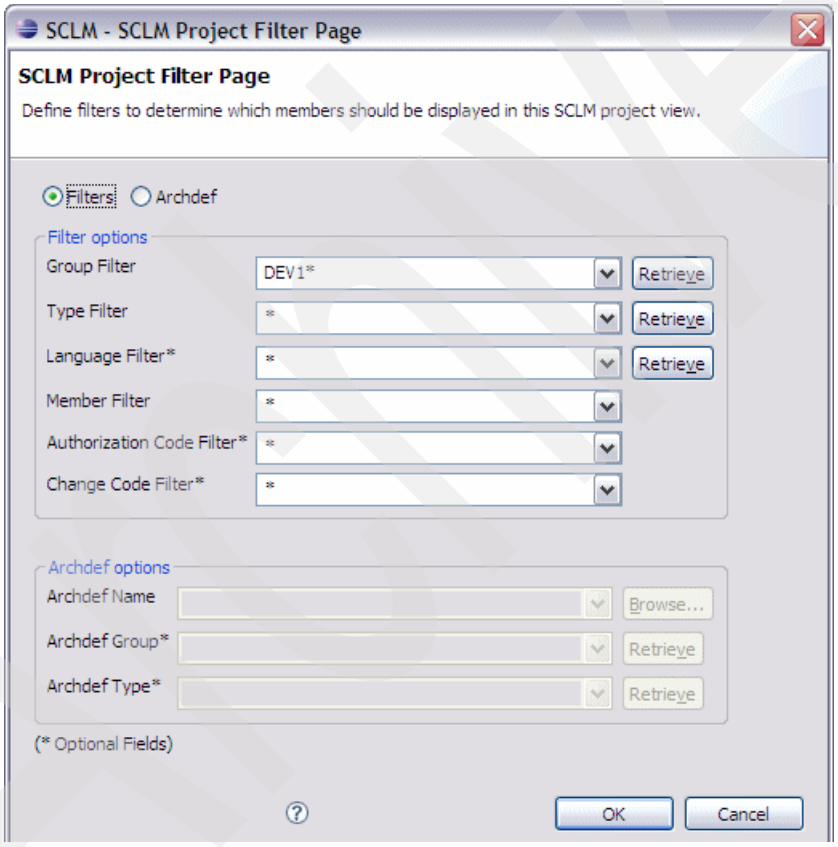


Figure 22-4 SCLM Developer Toolkit Filter panel

From here, locate your script in the usual manner. Once selected, the script will be displayed in the script editor window for you to make changes.

Deploy Skeleton button

The Deploy Skeleton button is used to generate a skeleton script for a standard WebSphere Application Server deployment. When you click **Deploy Skeleton**, the script is pasted into the script editor window. A number of the XML properties in this script will require tailoring. These values are marked with the string “TODO.” See the sample deploy skeleton in Example 22-1.

Example 22-1 Deploy skeleton

```
<ANTXML>
<project name="WAS Deployment" default="deploy" basedir=".">
<property name="SCLM_ANTXML" value="BWBDEPLA"/>
<property name="JACL_SCRIPT_NAME" value="/etc/SCLMDT/CONFIG/scripts/deploy.jacl"/>
<property name="SCLM_ARCHDEF" value="TODO"/>
<property name="EAR_FILE_NAME" value="TODO"/>
<property name="APPLICATION_NAME" value="TODO"/>
<property name="HOST_NAME" value="TODO"/>
<property name="PORT_NUMBER" value="TODO"/>
<property name="CELL_NAME" value="TODO"/>
<property name="NODE_NAME" value="TODO"/>
<property name="SERVER_NAME" value="TODO"/>
<property name="WSADMIN" value="/u/WebSphere/V5R1M0/AppServer/bin/wsadmin.sh"/>
</ANTXML>
```

Table 22-1 shows the sample deploy skeleton properties.

Table 22-1 Sample deploy skeleton properties

Property Name	Description	Default Value
SCLM_ANTXML	The action script to execute with properties defined by this property script	BWBDEPLA
JACL_SCRIPT_NAME	The location of the WAS JACL deployment script supplied with SCLM DT	*/path/*deploy.jacl
SCLM_ARCHDEF	The name of an SCLM ARCHDEF to associate with the script	TODO
EAR_FILE_NAME	The file name (long) of the EAR to be deployed	TODO
APPLICATION_NAME	Name of the application being deployed	TODO
HOST_NAME	The z/OS host name	TODO
PORT_NUMBER	The port number the WebSphere application server is listening on	TODO
CELL_NAME	Name of the application server cell to deploy to	TODO
SERVER_NAME	Name of the application server ‘server’ to deploy to	TODO
WSADMIN	The location of the WebSphere administration shell script on the Z/OS host’s hierarchical file system	Taken from local settings available in Windows...Preferences

Remote Deploy Skeleton button

The Remote Deploy Skeleton button is used to generate a skeleton script for a remote WAS deployment. The difference between this and the standard WebSphere Application Server deployment is that instead of instructing Application Server to install an EAR, we instruct it to pass the EAR along to a separate Application Server.

When you click **Remote Deploy Skeleton**, the script is pasted into the script editor window. A number of the XML properties in this script will require tailoring. These values are marked with the string “TODO.” The sample remote deploy skeleton is shown in Example 22-2.

Example 22-2 Remote deploy skeleton

```
<ANTXML>
<property name="ANT_BIN"  value="/u/WebSphere/V6R0/AppServer/bin/ws_ant.sh" />
<property name="WAS_EAR"  value="TODO"/>
<property name="WAS_HOST" value="TODO"/>
<property name="WAS_PORT" value="TODO"/>
<property name="WAS_HOME" value="/u/WebSphere/V6R0/AppServer"/>
<property name="SCLM_ANTXML" value="BWBRDEPL"/>
</ANTXML>
```

Table 22-2 shows the remote deploy properties.

Table 22-2 Remote deploy properties

Property Name	Description	Default Value
ANT_BIN	Location and name of the ANT service to run	Taken from local settings that specify location of the Application Server
WAS_EAR	The filename (long) of the EAR to deploy	TODO
WAS_HOST	The Application Server's host name	TODO
WAS_PORT	The Application Server's port number	TODO
WAS_HOME	The location of the AppServer of your z/OS Application Server. This is a location on the z/OS USS file system	TODO
SCLM_ANTXML	The action script to execute with properties defined by this property script	BWBSCOPY, which is the SCP implementation. Change to BWBSFTP for SFTP version.

Secure Deploy Skeleton button

The **Secure Deploy Skeleton** button is used to generate a skeleton script for a secure file transfer deployment. This sample uses SCP or SFTP, both of which ship with IBM Ported Tools (copyright/TM). You will require Ported Tools to be present and installed on your z/OS installation for this sample to work. It also requires key generation, and configuration of a non-interactive public key-exchange. Configuring IBM Ported Tools is outside of the scope of this book. The sample secure deploy skeleton is shown in Example 22-3.

Example 22-3 Secure deploy skeleton

```
<ANTXML>
<property name="LOCAL_FILE_PATH"  value="TODO"/>
<property name="REMOTE_FILE_PATH"  value="TODO"/>
<property name="SCLM_ANTXML"   value="BWBSCOPY"/>
<property name="TARGET_HOST_NAME"  value="TODO"/>
<property name="TARGET_USER_NAME"  value="TODO"/>
<property name="SSH_BIN_DIR" value="/bin"/>
</ANTXML>
```

Archived

Table 22-3 shows the secure deploy properties.

Table 22-3 Secure deploy properties

Property Name	Description	Default Value
LOCAL_FILE_PATH	Fully qualified USS file system path name of the file to deploy	TODO
REMOTE_FILE_PATH	Fully qualified USS file system path name of the location to deploy the file to	TODO
SCLM_ANTXML	The action script to execute with properties defined by this property script	BWBSCOPY or BWBSFTP
TARGET_HOST_NAME	Host name of the target z/OS host to deploy to	TODO
TARGET_USER_NAME	User name to utilize when logging on to the z/OS target host	TODO
SSH_BIN_DIR	The USS file system directory in which scp and sftp commands from Ported Tools are installed in	/bin

Choosing SCLM files for deployment

By any standard, deployment is useless unless you have something to deploy. Fortunately SCLM DT allows you to select the application to be deployed through the **Include SCLM Members** button.

Clicking **Include SCLM Members** will present you with the standard SCLM DT Filter member dialog. Go through the process of selecting a member and upon completion, you will notice the following line of script inserted into the script editor window.

```
<property longname="bin/Deployable.ear"
shortname="DE000001" group="DEV1" type="J2EEBLD"
dest="/var/deployment/"/>
```

This line of script causes the file represented by *longname* and *shortname* to be copied to the directory on the USS file system specified by *dest*.

Tip: It is possible to deploy using this single line of script.

22.2.4 Naming your script

When you create a new deployment script, SCLM DT will save a copy of it in an SCLM controlled data set. Some naming details must be provided. These are specified in the Deployment Script Name and Deployment Script Language combo boxes. The Deployment Script Type is also displayed, but may not be edited since deployment scripts are always of type J2EEBLD.

Under Deployment Script Name specify a z/OS-compatible member name. For Deployment Script Language, specify a language that supports character-set conversion (that is, ASCII to EBCDIC). A list of potential languages can be found in the TRANSLATE.conf configuration file. Translated languages are specified using the keyword TRANLANG.

Archived

22.2.5 Running deployment in batch mode

As with a variety of other SCLM DT functions, deployment can be run in batch processing mode. The **Batch Deployment** check box determines the current processing mode. The default is to use foreground processing (Batch Deployment is unchecked).

22.2.6 Managing deployment in an SCLM hierarchy

In staged development, there may be different requirements for deployment at each level. To understand this, it may help to think of an SCLM project with three groups.

There is a development group (DEV1), a testing group (TEST) and a production group (PROD). To begin with, the developers modify code and build the application for their own debugging and testing purposes at DEV1. They wish to test the application on the same server as SCLM for convenience.

Once promoted to the testing group (TEST), the application needs to be handed over to the testers, who do their work on a separate z/OS system or LPAR.

If the testers approve this version of the product, it is promoted to the production group (PROD). At this level, the application needs to be transported to a packaging system for final approval, before becoming available to users.

Can the deployment feature of SCLM DT cater for this type of scenario? Fortunately it can, through the use of level specific tags in the deployment script, and by selecting a deployment properties group at deploy time.

22.2.7 Group tags

In any deployment property script, properties that are specific to a particular group in the hierarchy can be tagged to belong to that group. Through this mechanism, a script may build up conditional behavior. For example, by specifying a different SCLM_ANTXML property at each group, a different kind of deployment will be initiated. The following example illustrates group tags in a “secure copy” deployment. However, the same technique of nesting group tags can be used for any deployment type, including deployment to WebSphere Application Servers.

Let us look at our earlier example of the three tier hierarchy consisting of DEV1, TEST, and PROD. We specified that in each case, a copy of the file to an appropriate USS file system should be performed. The host containing that USS file system varies depending on the current group.

At DEV1, the application should be placed on the local z/OS USS file system. At TEST, it should be transferred to the test host's USS file system. Finally, at PROD, yet another host was involved.

For simplicity sake, we look at a script that fulfils the requirement of DEV1 and TEST.

At DEV1, a simple local deployment is all that is required. A script is provided below, shown in Example 22-4, that performs such a task without hierarchy awareness. (See “Choosing SCLM files for deployment” on page 558 for more details).

Example 22-4 Sample local deployment script to deploy from DEV1

```
<ANTXML>
<property longname="Util.jar" shortname="UT000002"
group="DEV1" type="J2EEJAR"
```

```
dest="/var/SCLMDT/DEPLOYMENT"/>
</ANTXML>
```

At TEST, we require the application to be copied to another z/OS host, where it can be accessed by testers. An example script is shown in Example 22-5. (See , “Secure Deploy Skeleton button” on page 557 for more details).

Example 22-5 Sample secure deployment script to deploy from TEST

```
<ANTXML>
<property name="LOCAL_FILE_PATH"  value="/var/SCLMDT/DEPLOYMENT/Util.jar"/>
<property name="REMOTE_FILE_PATH"  value="/u/bhorwoo/testing/"/>
<property name="SCLM_ANTXML"   value="BWBSCOPY"/>
<property name="TARGET_HOST_NAME"  value="PTHISD1"/>
<property name="TARGET_USER_NAME"  value="BHORWOO"/>
<property name="SSH_BIN_DIR" value="/bin"/>
<property longname="Util.jar"  shortname="UTOOOOO2" group="TEST" type="J2EEJAR"
dest="/var/SCLMDT/DEPLOYMENT/"/>
</ANTXML>
```

So now we must ask the question: how do we combine these two into a single script, while retaining the group-sensitive behavior? The answer is by using group tags. Examine Example 22-6 that implements the behavior of both scripts. It is quite straightforward. The properties for DEV1 are placed inside a pair of DEV1 tags. The properties for TEST are placed inside a pair of TEST tags. Both DEV1 and TEST tags are placed inside ANTXML tags.

Example 22-6 Sample deployment script to deploy from both DEV1 and TEST

```
<ANTXML>
<DEV1>
   <property longname="Util.jar"  shortname="UTOOOOO2" group="DEV1" type="J2EEJAR"
dest="/var/SCLMDT/DEPLOYMENT/"/>
</DEV1>
<TEST>
   <property name="LOCAL_FILE_PATH"  value="/var/SCLMDT/DEPLOYMENT/Util.jar"/>
   <property name="REMOTE_FILE_PATH"  value="/u/bhorwoo/testing/"/>
   <property name="SCLM_ANTXML"   value="BWBSCOPY"/>
   <property name="TARGET_HOST_NAME"  value="PTHISD1"/>
   <property name="TARGET_USER_NAME"  value="BHORWOO"/>
   <property name="SSH_BIN_DIR" value="/bin"/>
   <property longname="Util.jar"  shortname="UTOOOOO2" group="TEST" type="J2EEJAR"
dest="/var/SCLMDT/DEPLOYMENT/"/>
</TEST>
</ANTXML>
```

Tip: When using hierarchy sensitive scripts, if one property is placed inside a group tag, all other properties must be inside group tags too.

22.2.8 Deployment properties group

Now that we know how to write group sensitive deployment scripts, we need the ability to invoke a deployment on a particular group. This is achieved through the deployment properties group control in the deployment dialog. See Example 22-1 on page 551.

The control is a drop-down list that is populated with the various groups from the current SCLM Project. To initiate a deployment on a particular group, carry out all deployment steps as normal, but specify that group using the drop-down list.

What if you are not initiating deployment from Eclipse? For example, let us say you want to deploy in a user exit. In that case you will need to know the arguments that are passed to the deployment service BWBJ2DPY as shown in Table 22-4:

```
BWBJ2DPY PREFIX PROJ PROJDEF GROUP GROUPDEV MEMBER GROUPDPY DEPSEC DEPMODE SUBMIT
```

Table 22-4 Arguments

Argument Name	Description
PREFIX	The users TSO prefix
PROJ	SCLM Project HLQ
PROJDEF	Project definition HLQ
GROUP	Group in which script rests
GROUPDEV	Development group of the project
MEMBER	Member name of the script
GROUPDPY	The name of the “deployment group” (hierarchy sensitive)
DEPSEC	Deployment security (Y/N)
DEPMODE	Mode of deployment
SUBMIT	Use online processing, or batch processing

Below is an example taken from the JCL of a batch deployment job. The project’s name is SCLM10, which is the same name as the project definition HLQ. The TSO prefix is BHORWOO. The development group and script location group are the same: DEV1. The deployment properties group is TEST, there is no build security, and the deployment script's member name is COMBO.

```
ISPSTART CMD(BWBJ2DPY +
BHORWOO SCLM10 SCLM10 DEV1 DEV1 COMBO TEST N NONE
NONE) +
LANG(CREX)
```

22.3 Usage scenarios: Deployment in action

Now that you understand a bit more about deployment and how it is implemented in SCLM DT, it is time to see some working scenarios.

First we show the simplest example: copying a file from SCLM to the z/OS USS file system. Following that, we will take on a more complicated case: deployment to a WebSphere Application Server.

Through following these examples, you should gain enough confidence to begin implementing a deployment that fits your requirements.

Archived

22.3.1 Scenario 1: SCLM to z/OS USS deployment

In this scenario, we consider deployment to z/OS USS.

Introduction

The description of this scenario is as follows; we are developing a JAR that contains some utility services. We want SCLM to place the JAR onto the z/OS USS file system at deploy time, so that we can run tests on it and debug the code.

Preparation

To begin, we require the following components:

- Java source code that produces the JAR
- An ARCHDEF describing the JARs resources
- A build script specifying the Java JAR builder

You will need to create a new Java project in SCLM DT. If you do not know how to create a Java project in Eclipse, take a look at the documentation on the Web site:

`http://www.eclipse.org`

Tip: After creating a new Java project, the .classpath file created might not have a path set for the output location. If you keep it this way, all locally compiled classes will be stored in the same directory as the source. To make the migration process easier, specify a path here such as bin/. This way, in the first dialog of the migration wizard, you can uncheck the bin/ directory to avoid adding pre-built objects to SCLM, shown in Figure 22-5.

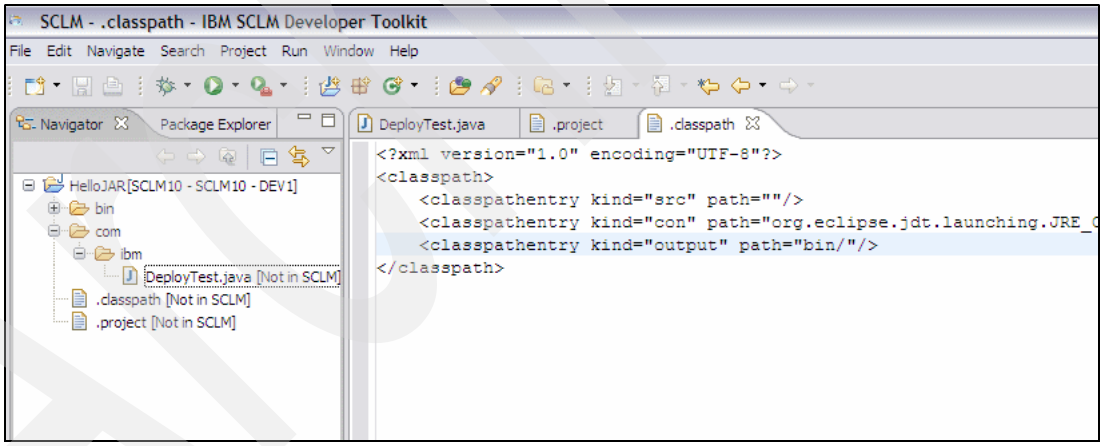


Figure 22-5 Specifying bin/ as the output path in the .classpath file

Create a new class to go into this project. In our example we will use the reliable and timeless "Hello World" program, implemented in a class called DeployTest, as shown in Figure 22-6.

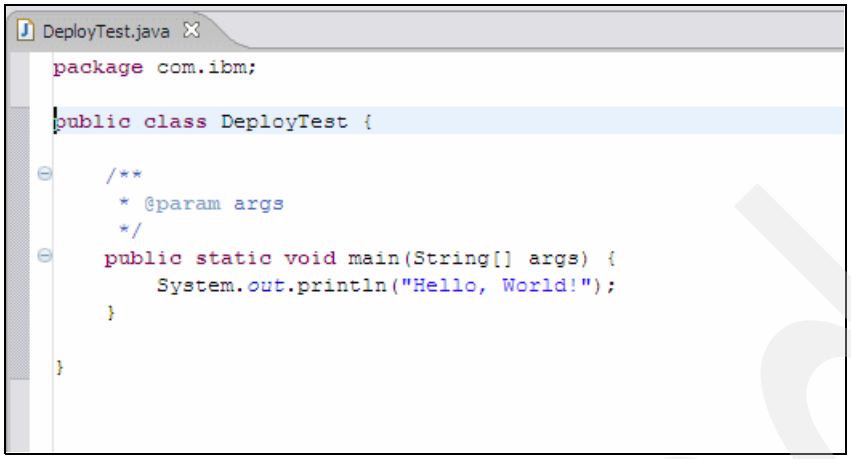


Figure 22-6 DeployTest.java sample program

Map the project to an SCLM project by right-clicking **Team** → **Share Project** as shown in Figure 22-7.

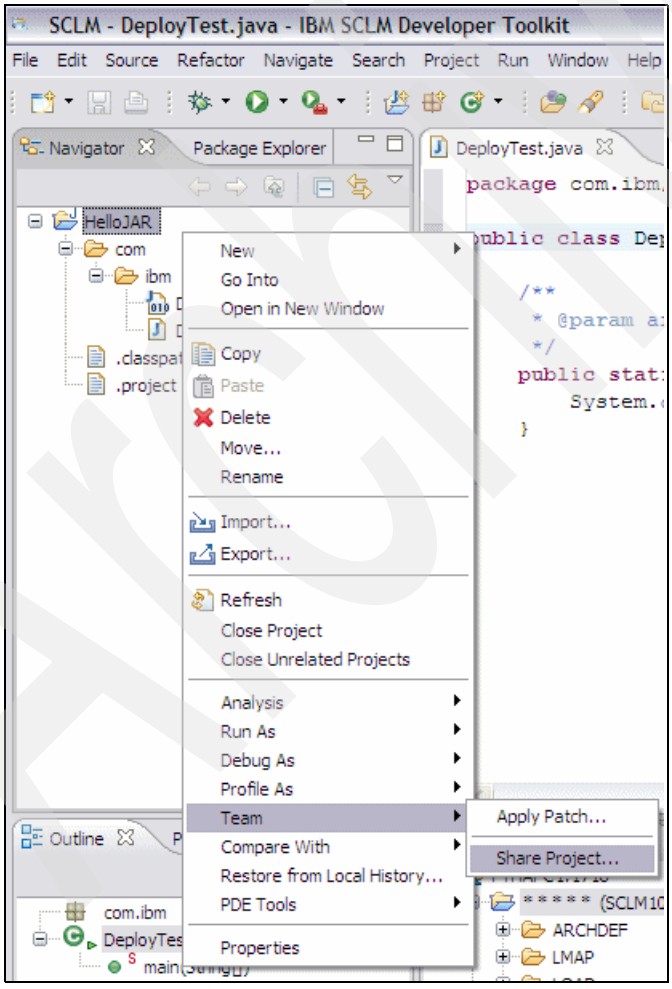


Figure 22-7 Mapping the project to SCLM

Add all the project resources to SCLM by right-clicking **Team** → **Add to SCLM**. This is shown in Figure 22-8.

Archived

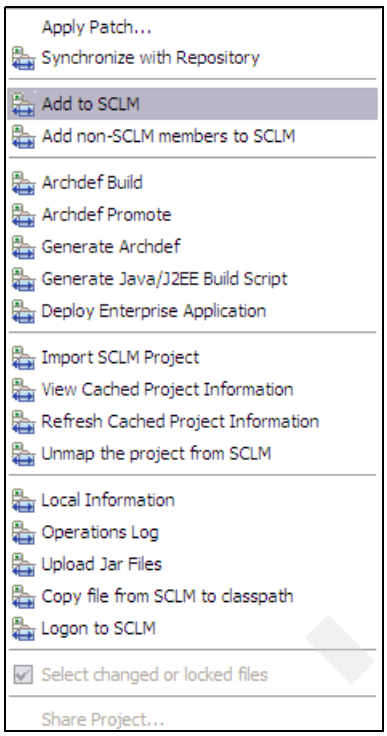


Figure 22-8 Selecting the Add to SCLM option from the Team context menu

Finally, build the project's ARCHDEF by right-clicking **Team** → **Archdef Build**.

By populating the SCLM Developer view, we can now see the built JAR object sitting comfortably in SCLM as shown in Figure 22-9.

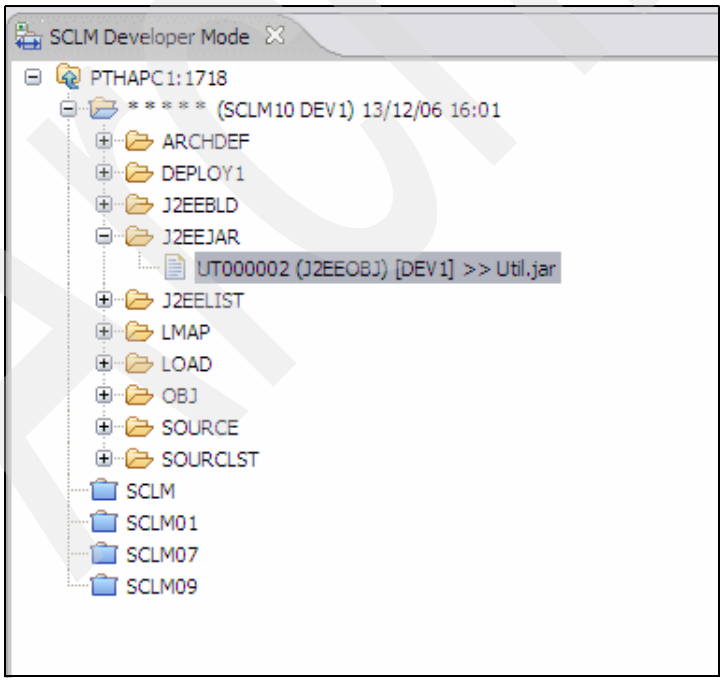


Figure 22-9 Listing the generated jar file in SCLM

We are ready now to deploy the application.

Archived

Creating the script

The first step in creating a new deployment script is to open the deployment dialog. To do this in the navigator view, right-click the project, **Team** → **Deploy Enterprise Application**. In the SCLM Developer view, simply right-click the project node and select **Deploy Enterprise Application.**

You may wish to briefly review “Choosing SCLM files for deployment” on page 558 to familiarize yourself with the user interface.

Click **Create deployment script**, and then click the **Include SCLM Members** button. You are presented with the SCLM member filter dialog. This is where you choose the object you want to deploy. In this example, we filter by type J2EEJAR. The candidates for selection are then displayed as shown in Figure 22-10. We click **Util.jar**, then **OK**.

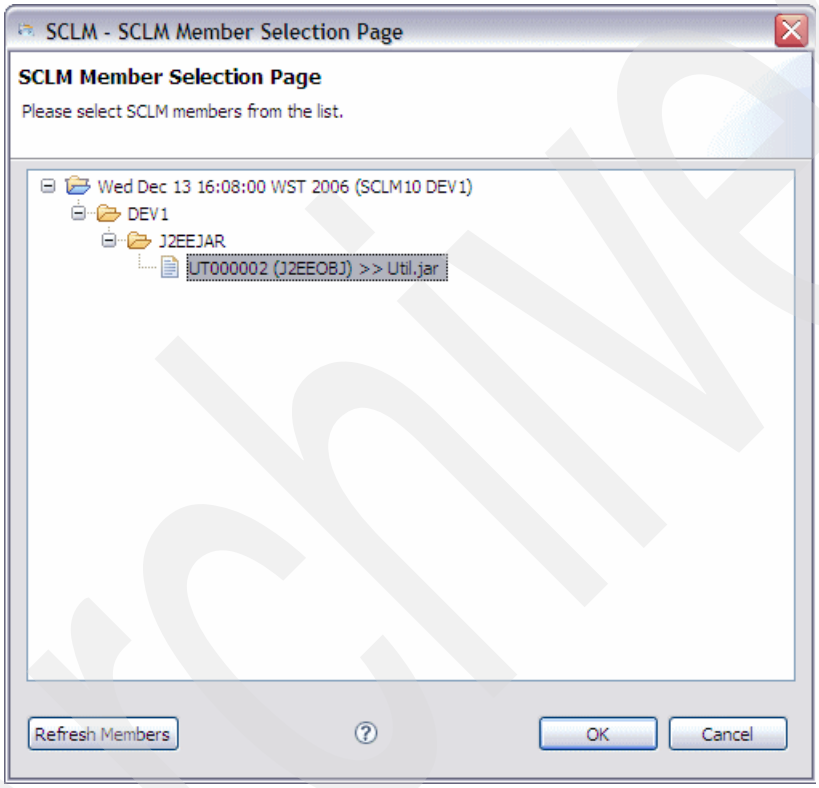


Figure 22-10 Selecting the jar to deploy

The following script is pasted into the script editing pane.

```
<ANTXML>
<property longname="Util.jar"
shortname="UT000002" group="DEV1"
type="J2EEJAR" dest="TODO"/>
</ANTXML>
```

All that has to be done is to replace the “TODO” value with a location on the z/OS USS file system. For this example, we choose to place it in /var/SCLMDT/DEPLOYMENT/ so we specify that location as follows:

```
<ANTXML>
<property longname="Util.jar"
shortname="UT000002" group="DEV1"
type="J2EEJAR"
```

Archived

```
dest="/var/SCLMDT/DEPLOYMENT/"/>
</ANTXML>
```

As discussed in 22.2.4, “Naming your script” on page 558, we now need to specify the name of the member to store the script in, and the member's language. We have chosen to call the script DEP1. We recommend that you always use the TEXT language. Figure 22-11 shows specifying the options to store the script.

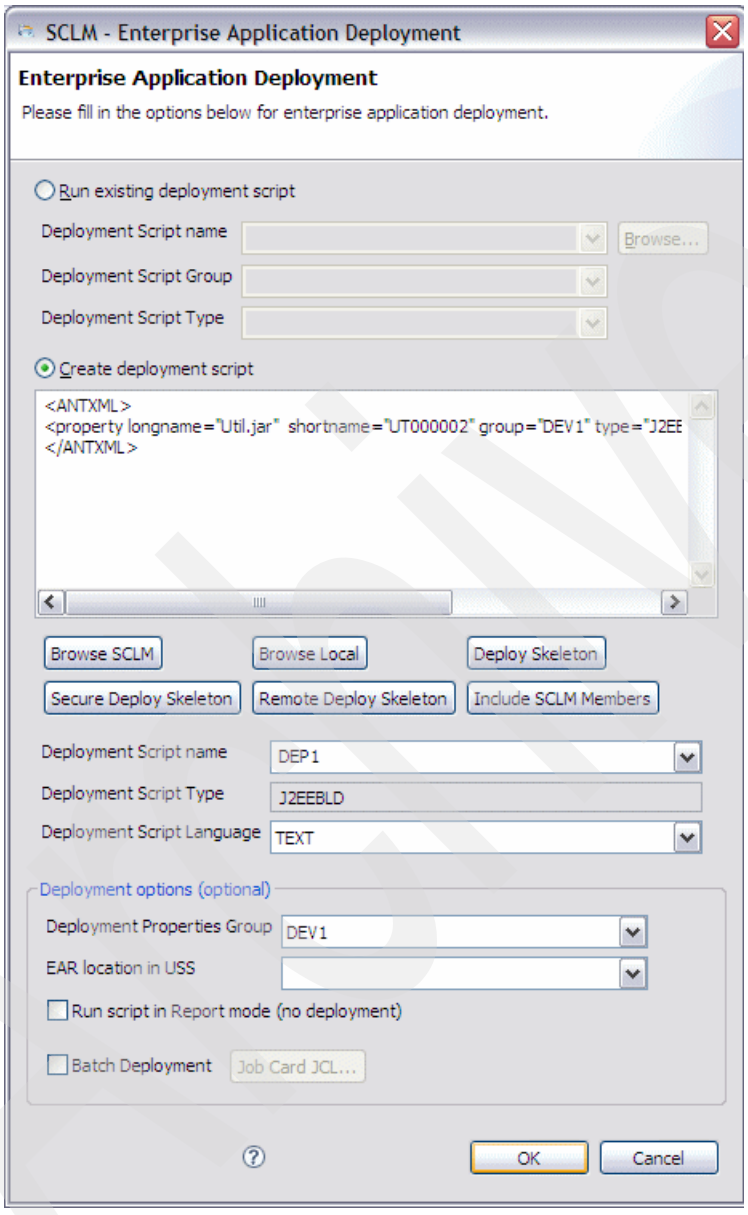


Figure 22-11 Storing the deployment script in SCLM

Tip: Avoid giving a deployment script the same name as an ARCHDEF. Java/J2EE ARCHDEFs will have a build script with the same member name in type J2EEBLD. If you name a deployment script after the ARCHDEF, the build script will be overwritten, causing subsequent builds to fail.

Click **OK** to begin the deployment. A short time later you will be notified of the result. You should see a success dialog, if you have it enabled under **Windows** → **Preferences** → **SCLM Preferences**.

The following lines of text in the log confirm that the copy worked:

```
shortname = UT000002
type = J2EEJAR
Member located at SCLM GROUP=DEV1
Location=SCLM10.DEV1.J2EEJAR(UT000002)
OPUT 'SCLM10.DEV1.J2EEJAR(UT000002)'
'/var/SCLMDT/DEPLOYMENT/Util.jar' BINARY
IGD103I SMS ALLOCATED TO DDNAME SYSXXXXX
```

Invoking the script

Invoking the script is a simple matter of returning to the deployment dialog, and plugging in the member details you provided for your script. Figure 22-12 shows the completed dialog.

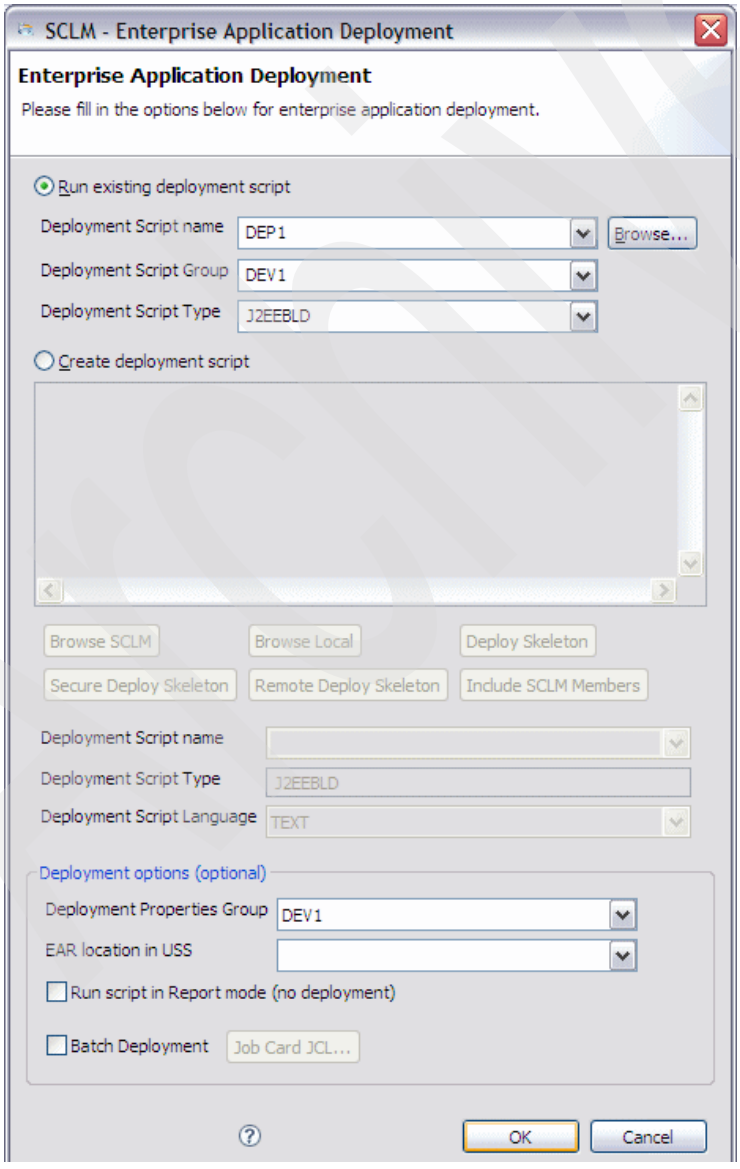


Figure 22-12 Completed deployment panel for scenario 1

Unless you configure deployment from a build exit, this is how you will trigger future deployments when new versions of the JAR become available.

Verifying the results

Performing the role of the so-called “tester” discussed in 22.3.1, “Scenario 1: SCLM to z/OS USS deployment” we wish to verify the process. Our tools for testing? OMVS and the Java interpreter. If you are using SCLM DT as a component of WebSphere Developer for System z, then you can use other components to carry out this task.

Tip: When initiating the Java interpreter through OMVS, be sure your region size is sufficiently large to run Java programs, or the command will fail and you will see **`malloc`** related errors. Also ensure that the Java bin directory is in your PATH environment variable.

Start a 3270 session with the host. From TSO, input the command OMVS. From ISPF, issue TSO OMVS.

You are confronted with the familiar shell prompt. From here you may simply perform the command:

```
java -cp /var/SCLMDT/DEPLOYMENT/Util.jar com.ibm.DeployTest
```

At this point, if you correctly input the program shown previously in Figure 22-6 on page 563, you will be presented with the following console message:

```
Hello, World!
```

22.3.2 Scenario 2: SCLM to WebSphere Application Server deployment

In this scenario, we consider deployment to WebSphere Application Server.

Introduction

Since writing and building J2EE applications is a complex area on its own, we run through this example using a simple EAR project. Our concern is deployment rather than J2EE development. For a more detailed walk-through of J2EE on WebSphere Application Server, you can start with the Redbooks publication, *Experience J2EE! Using WebSphere Application Server V6.1*, SG24-7297, published 31 January 2007.

In addition, the WebSphere Application Server deployment samples are most likely of all samples to require customization to fit your requirements. This is due to the variety of possible installation steps that could be required, depending on the type of project.

Preparation

To begin with, we need an EAR. The example EAR in this scenario is a very basic one. It wraps a WAR file that contains a file called **`hello.jsp`** - a trivial Web page that displays some details about the “requester.”

For HelloWeb, the Uniform Resource Identifier (URI) used to access the application is set in the EAR’s /META-INF/application.xml, as shown in Example 22-7.

Example 22-7 application.xml for HelloWeb application EAR

```
<?xml version="1.0" encoding="UTF-8"?>
<application id="Application_ID" version="1.4"
             xmlns="http://java.sun.com/xml/ns/j2ee"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                 http://java.sun.com/xml/ns/j2ee/application_1_4.xsd">
        <display-name>HelloEAR</display-name>
        <module id="WebModule_1166063054921">
                <web>
                        <web-uri>HelloWeb.war</web-uri>
                        <context-root>HelloWeb</context-root>
                </web>
        </module>
</application>
```

By browsing or editing this file we can determine, or set, the “Web address” that needs to be used to access the application.

The URI consists of:

```
http://(server name) : (port number)/(context-root)/
```

In application.xml, the <module> tag specifies the details of the WAR module embedded within the EAR. In this case the EAR contains a single WAR module, where the context-root has been set to HelloWeb.

The <web-uri> tag specifies the internal URI of the WAR module relative to the root of the EAR. In this case our EAR contains HelloWeb.war in its root directory, so the value is HelloWeb.war.

So, assuming our host name is http://myserver.com and port is 9080, the URI to access the application will be:

```
http://myserver.com:9080/HelloWeb/
```

Much of the information provided in deployment descriptors is vendor specific, but for more information regarding the J2EE specification, see http://java.sun.com.

In Eclipse, use the project wizards to create a dynamic Web application (WAR) and a J2EE EAR. Link them both together.

The source code to **hello.jsp** is provided in Example 22-8.

Example 22-8 hello.jsp sample source code

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>hello</title>
    <meta http-equiv="Content-Type" content="text/html; ISO-8859-1">
    <meta name="GENERATOR" content="Eclipse Platform">
    <%@page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
  </head>
  <body>
    <table align="center">
```

```
      <tr>Hello <%= request.getRemoteHost()%> ! </tr>
      <tr><td> Protocol </td><td><%= request.getProtocol() %></td></tr>
      <tr><td> Auth Type </td><td><%= request.getAuthType() %></td></tr>
      <tr><td> Content-Length </td><td><%= request.getContentLength() %></td></tr>
      <tr><td> Content-Type </td><td><%= request.getContentType() %></td></tr>
      <tr><td> Context Path </td><td><%= request.getContextPath() %></td></tr>
      <tr><td> Remote Address </td><td><%= request.getRemoteAddr() %></td></tr>
      <tr><td> Remote Port </td><td><%= request.getRemotePort() %></td></tr>
      <tr><td> Remote User </td><td><%= request.getRemoteUser() %></td></tr>
      <tr><td> Scheme </td><td><%= request.getScheme() %></td></tr>
      <tr><td> Locale </td><td><%= request.getLocale() %></td></tr>
      <tr><td> Local Name </td><td><%= request.getLocalName() %></td></tr>
      <tr><td> Local Port </td><td><%= request.getLocalPort() %></td></tr>
      <tr><td> Local Address </td><td><%= request.getLocalAddr() %></td></tr>
      <tr><td> Path Info </td><td><%= request.getPathInfo() %></td></tr>
      <tr><td> User Principle </td><td><%= request.getUserPrincipal() %></td></tr>
    </table>
  </body>
</html>
```

Once this is successfully configured in Eclipse, add the two projects to SCLM DT under separate ARCHDEFs. Remember that using the standard build method, the EAR's ARCHDEF must INCL the WAR's ARCHDEF to be a nested component.

Build the EAR ARCHDEF. If you have configured everything correctly, you are ready to create the deployment script.

Creating the script

The first step in creating a new deployment script is to open the deployment dialog. To do this in the navigator view, right-click the project, **Team** → **Deploy Enterprise Application**. In the SCLM Developer view, simply right-click the project node and select **Deploy Enterprise Application**.

If you want, review the Deploy Skeleton button to familiarize yourself with the user interface.

Click **Create deployment script**, followed by the **Deploy Skeleton** button. The following script skeleton, shown in Example 22-9, will be pasted into the script editing pane.

Example 22-9 Creating the deployment script: default values

```
<ANTXML>
<project name="WAS Deployment" default="deploy" basedir=".">
<property name="SCLM_ANTXML" value="BWBDEPLA"/>
<property name="SCLM_BLDMODE" value="Forced"/>
<property name="JACL_SCRIPT_NAME" value="/etc/SCLMDT/CONFIG/scripts/deploy.jacl"/>
<property name="SCLM_ARCHDEF" value="TODO"/>
<property name="EAR_FILE_NAME" value="TODO"/>
<property name="APPLICATION_NAME" value="TODO"/>
<property name="HOST_NAME" value="TODO"/>
<property name="PORT_NUMBER" value="TODO"/>
<property name="CELL_NAME" value="TODO"/>
<property name="NODE_NAME" value="TODO"/>
<property name="SERVER_NAME" value="TODO"/>
<property name="WSADMIN" value="/u/WebSphere/V5R1M0/AppServer/bin/wsadmin.sh"/>
</ANTXML>
```

As with our previous scenario, we replace all instances of the text “TODO” with real values. For this scenario, the following completed script, shown in Example 22-10, was created.

Example 22-10 Creating the deployment script: after tailoring

```
<ANTXML>
<project name="WAS Deployment" default="deploy" basedir=".">
<property name="SCLM_ANTXML" value="BWBDEPLA"/>
<property name="SCLM_BLDMODE" value="Forced"/>
<property name="JACL_SCRIPT_NAME" value="/etc/SCLMDT/CONFIG/scripts/deploy.jacl"/>
<property name="SCLM_ARCHDEF" value="DEPLEAR"/>
<property name="EAR_FILE_NAME" value="HelloEAR.ear"/>
<property name="APPLICATION_NAME" value="HelloApp"/>
<property name="HOST_NAME" value="pthapc1"/>
<property name="PORT_NUMBER" value="8880"/>
<property name="CELL_NAME" value="APC1"/>
<property name="NODE_NAME" value="APC1"/>
<property name="SERVER_NAME" value="server1"/>
<property name="WSADMIN" value="/u/WebSphere/V6R0/AppServer/bin/wsadmin.sh"/>
</ANTXML>
```

As discussed in 22.2.4, “Naming your script” on page 558, we now have to specify the name of the member to store the script in, and the member's language. We have chosen to call the script DEP1. We recommend that you always use the TEXT language.

Tip: Avoid giving a deployment script the same name as an ARCHDEF. Java/J2EE ARCHDEFs will have a build script with the same member name in type J2EEBLD. If you name a deployment script after the ARCHDEF, the build script will be overwritten, causing subsequent builds to fail.

Click **OK** to begin the deployment. A short time later, you will be notified of the result. You should see a success dialog, if you have it enabled under **Windows** → **Preferences** → **SCLM Preferences**.

Archived

Invoking the script

Invoking the script is a simple matter of returning to the deployment dialog, and plugging in the member details you provided for your script. Figure 22-13 shows the completed dialog.

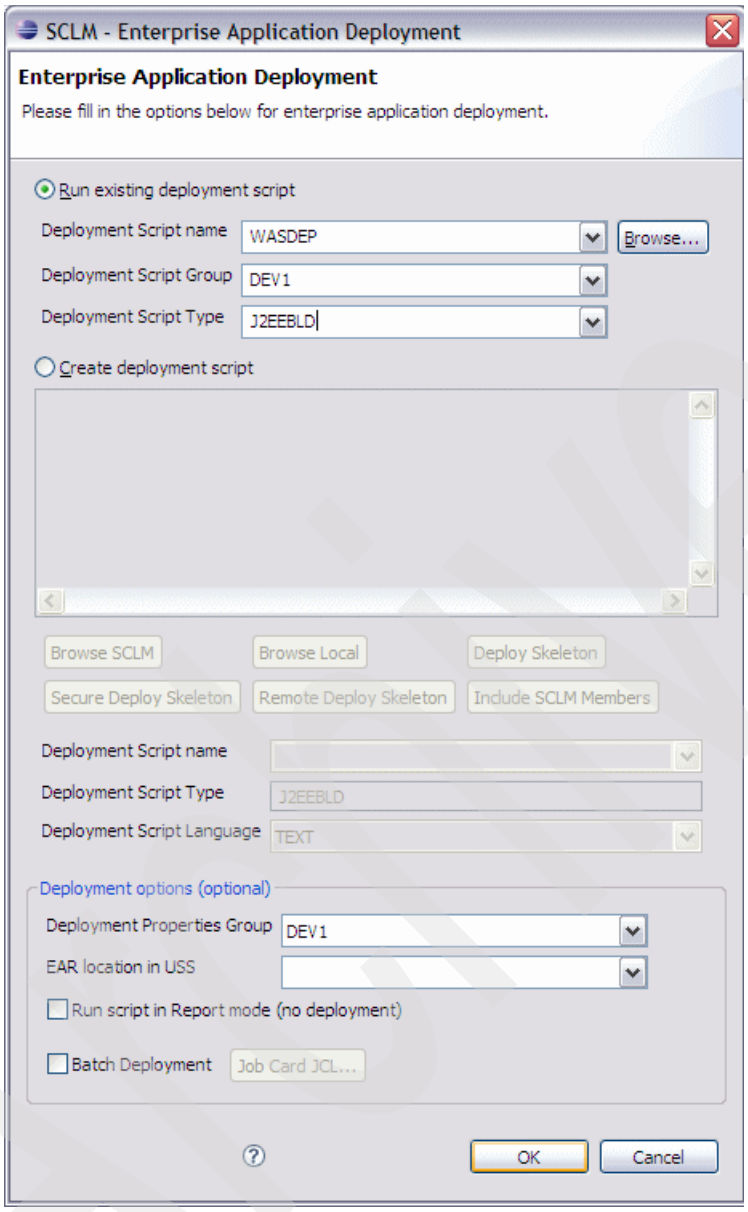


Figure 22-13 Completed deployment panel for scenario 2

Unless you configure deployment from a build exit, this is how you will trigger future deployments when new versions of the JAR become available.

Verifying the results

Verifying the results for this deployment is particularly easy. Just fire up a Web browser with the host name and port number of your Application Server, and append the name of your project.

For our scenario, the result is seen in Figure 22-14.

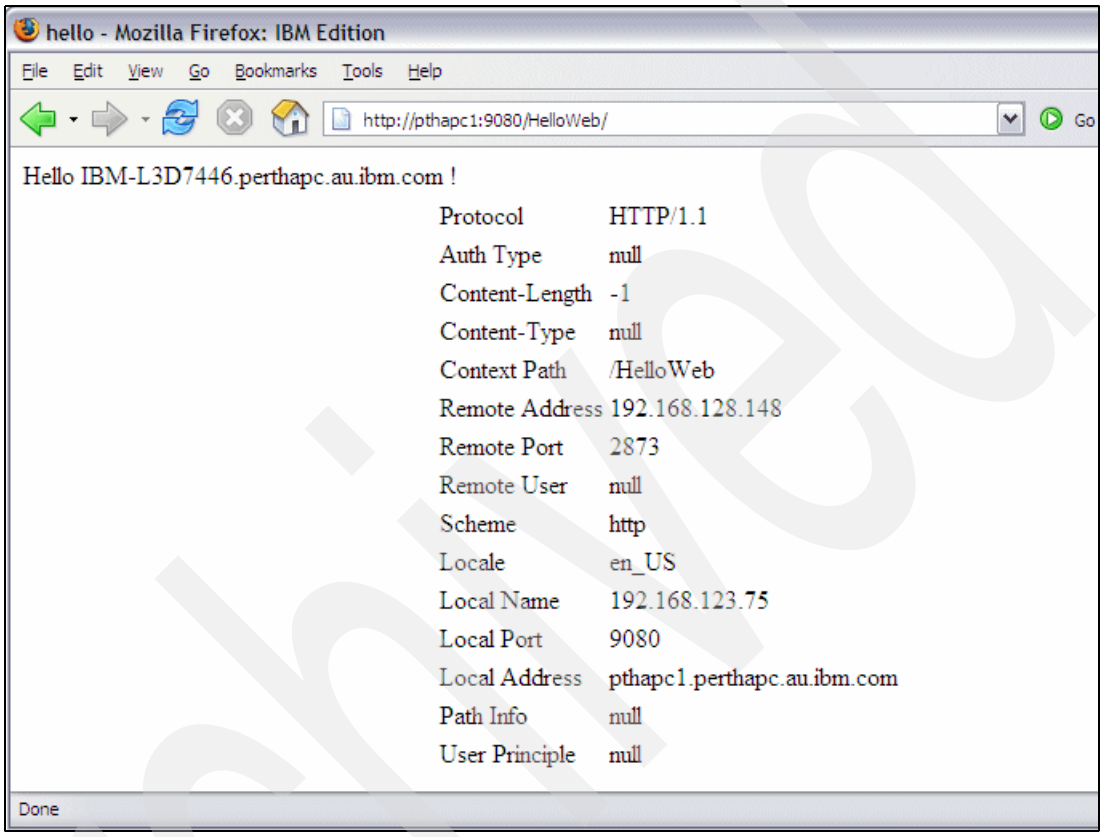


Figure 22-14 Running your application

Archive

Archived

Part 6

SCLM Administrator Toolkit

In this part of the book, we discuss the SCLM Administrator Toolkit as an add-on product to SCLM and a member of SCLM Advanced Edition. We provide a high-level overview of the Admin Toolkit to introduce you to the value the Admin Toolkit provides. We include a more in-depth discussion of all Admin Toolkit functions to further acquaint you with the intricacies of the Toolkit.

We provide step-by-step instructions on how to utilize each of the functions the Admin Toolkit provides, including a section that defines terminology that is specific to the Admin Toolkit, so a new user can better understand the functions and discussions.

We discuss a usage scenario in detail to give you the big picture of how to use the Admin Toolkit in a real-world scenario, and showcase how the features of the Admin Toolkit can greatly benefit the SCLM project administrator and application development lifecycle.

© Copyright IBM Corp. 2007. All rights reserved.

Archived

23

IBM SCLM Administrator Toolkit

In this chapter we provide an overview of the SCLM Administrator Toolkit (Admin Toolkit), the functions and features of the product, and its corresponding terminology.

We discuss the following topics:

- What is the Admin Toolkit?
- The key components of the Admin Toolkit
- Terminology used in the Admin Toolkit
- Using the workstation-based interface
- Using the ISPF-based interface

Archived

© Copyright IBM Corp. 2007. All rights reserved.

23.1 What is the Administrator Toolkit?

The IBM Software Configuration and Library Manager (SCLM) is a program that provides software configuration and library management for the z/OS platform. The SCLM Administrator Toolkit (Admin Toolkit) provides you with a set of tools that make defining and maintaining SCLM projects simpler and easier than ever before.

The administrator functions are accessible from either a workstation-based graphical interface or ISPF panels, thereby allowing users who prefer either type of interface to be equally effective and productive almost immediately. The workflow in both interfaces is the same, meaning that the options in the ISPF panels will appear in the same places in the client, making it easier for a user to navigate in either interface.

When creating a new SCLM project, senior administrators usually have canned JCL to define source code data sets and project data sets. When a new project must be created, the canned JCL is copied to a new data set and then modified to suit the needs of the new project. By using the Admin Toolkit, source code data sets and project data sets are created automatically, eliminating the necessity to create them manually.

Project definitions can be complex and time-consuming to create manually, and mistakes can be easily made. By using the content-sensitive fields in the Admin Toolkit, mistakes are virtually eliminated when creating new projects, and projects can be created in a fraction of the time it takes to create them manually.

Archived

23.2 Key components of the Administrator Toolkit

The Admin Toolkit comprises several automated utilities and wizards that assist the SCLM project administrator in common project administration tasks. In addition to assisting with SCLM project creation and maintenance, the SCLM Administrator Toolkit includes wizards that migrate assets into a new or existing project, create new languages or ARCHDEFs, or administer Enhanced Access Control (EAC) or RACF profiles.

The wizards and utilities contained in the Admin Toolkit are:

- **Project Editor:** Using the Project Editor, users can define data set types, package backout parameters, and groups. If users are modifying an existing project and want to delete a type or group, the Project Editor will, after confirmation from the user, delete any data associated with that type or group. When defining a project lifecycle, or hierarchy, using the workstation interface, drop boxes can be dragged around the display in order to represent project groups and build a network of linked toolboxes, much like an organization chart. Once the definition is completed, SCLM Administrator Toolkit generates an SCLM project hierarchy. A similar process to add or modify groups is also available using the ISPF panels.
- **Project Cloning Utility:** Project definitions and/or entire projects can be replicated using the Cloning utility, including the source code data set contents. In this way, new and alternate projects can be quickly created and then tailored for your needs.
- **Migration Wizard:** Software assets can be easily migrated into an SCLM project from existing libraries using the Migration Wizard or from remote systems using the Remote Migration Wizard. Assets, including those with long file names, can be migrated into SCLM projects and the corresponding ACCOUNTING data will be automatically updated for each asset migrated into your project.
- **Architecture Definition Wizard:** Architecture definitions tell SCLM how pieces of the project's source code fit together and are crucial to the structure of a project. You can automatically generate SCLM architecture definitions from an existing load module or parse existing JCL into an ARCHDEF. A manual option for creating architecture definitions from scratch also exists.
- **Language Definition Wizard:** Language definitions can be added, modified, or removed using the Language Definition Wizard. New language definitions can also be created from existing definitions, existing JCL, or directly via the editor.
- **Enhanced Access Control (EAC) Wizard:** Available only through the GUI, it is an intuitive, easy-to-use utility that enables the project administrator to create, list, modify, and delete RACF data set and RACF profiles. In addition, the EAC Wizard allows applications to be defined, modified, viewed, and deleted, as well as allowing project administrators to view access violations and implement new or updated Rules Files.
- **User Exit Configuration:** Required user exits, including Breeze user exits, can be configured using the Project Editor. If more than one exit must be called for a specific break point, a User Exit Wrapper can be configured to run two or more exits in serial in one break point.
- **VSAM Maintenance Wizard:** An interface to perform project VSAM allocation, deletion, and maintenance tasks, with the ability for users to add their own tasks to the menu. Any REXX script can be a VSAM task and does not have to perform tasks against project VSAM data sets.

23.3 Terminology used in the Administrator Toolkit

The Admin Toolkit assumes that users are familiar with software change control and library management and have some knowledge of SCLM. However, there are certain terms and phrases specific to the Admin Toolkit that should be defined in order to understand some of the material. The following terms are referenced throughout the Admin Toolkit sections of this book:

Build/Rebuild The process of taking the options and parameters saved in PDML, or the '<project>.PROJDEFS.SETTINGS' data set, and generating new project source in the data set '<project>.PROJDEFS.SETTINGS(<project>)'.

Clone The process of creating a new project by copying an existing project's definition, to include optionally copying the underlying application source code. The new projects' definition can then be edited to meet the requirements of a new application's source code development lifecycle.

Copybook Editor A function within the Administrator Toolkit that is the host equivalent of the Refactor function in the client.

Incomplete The status of a project as displayed in the host-based user interface. This means that a projects' Source definition exists in data set '<project>.PROJDEFS.SOURCE(<project>)', but no matching PDML in data set '<project>.PROJDEFS.SETTINGS(<project>)'.

Parse The process of reading in the projects' source located in '<project>.PROJDEFS.SOURCE(<project>)' and generating new PDML.

PDML Abbreviation for "Project Definition Markup Language". PDML is the projects' options and parameters that were set in the Admin Toolkit and saved in UTF-8 format in the data set '<project>.PROJDEFS.SETTINGS(<project>)'.

Refactor The process of rewriting a program to change its structure, with the explicit purpose of keeping its meaning or behavior the same. In the case of Administrator Toolkit, the Refactor tab in the client allows you to view/edit project definition without having to exit the product and start a mainframe emulator. In this way, you can change the structure of how the project is defined without changing how the project administers the underlying application source code.

Script A series of instructions that are passed between the host or the client and USS where actions then occur depending on the instructions passed.

Synchronized The status of a project as displayed in the host-based user interface. This means that the projects' Source definition located in data set '<project>.PROJDEFS.SOURCE(<project>)' matches the projects' PDML located in data set '<project>.PROJDEFS.SETTINGS(<project>)'.

Archived

Unsynchronized The status of a project as displayed in the host-based user interface. This means that the projects' Source definition located in '<project>.PROJDEFS.SOURCE(<project>)' does not match the projects' PDML located in data set '<project>.PROJDEFS.SETTINGS(<project>)'.

This is usually an indication that either the options of the project have been edited in the Admin Toolkit and not 'built' into the projects' Source definition, or there were edits made to the projects' Source definition outside of the Admin Toolkit that have not been 'parsed' into PDML.

As a refresher, there are some other SCLM-specific terms that are used throughout the Admin Toolkit section:

Alternate Project A project that can use part of a "main" projects' definition contained in the '<project>.PROJDEFS.SOURCE' data set, and optionally can use some of the same underlying application source code.

ARCHDEF Abbreviated name for architecture definition. ARCHDEFs tells SCLM where to find the pieces of a program, how it is to be built and where to put the resultant built and completed application program.

Group/Type A data set that exists in the project that holds the projects' application source code data. Such a data set could be '<project>.DEV.ASM' where assembler source code is stored for the group DEV.

Migrate The process of copying an asset from another project or a partitioned data set to an SCLM-managed project's data sets, and updating the ACCOUNTING and AUDITING data sets with information about the newly migrated asset.

23.4 Using the workstation-based interface

The client interface can be downloaded from the host to your PC workstation using FTP or your emulator's file transfer tool. The Admin Toolkit clients available for download are listed in Table 23-1.

Table 23-1 Client

Workstation operating system	Corresponding member to download
Windows 2000 or Windows XP	AUZ.SAUZGUIW(AUZWIN32)
Linux® GTK	AUZ.SAUZGUIK(AUZLNGTK)
AIX® 5L™	AUZ.SAUZGUIA(AUZAIX5L)

Each of the members are self-extracting executable programs that, when run, will result in files being created in directories you specified to install the client.

The SCLM Admin Toolkit client uses the InstallShield MultiPlatform Installer to install the client onto your PC. During the installation, you will be asked if you have an Eclipse-based application already installed that you would like to use. If you already have Eclipse 1.3.2 on your PC from another application, then you can install the Admin Toolkit as a plug-in. If you do not have Eclipse v1.3.2 or higher on your PC, then Eclipse will be installed as well, and the Admin Toolkit will be installed as a stand-alone application.

Once the installation is complete, use the following instructions in this chapter to configure the workstation client to connect to the host. You must configure the client before you begin to use the product.

The SCLM Admin Toolkit is not a smart client, but rather is an installable application. The Admin Toolkit workstation-based client can be installed on Linux, AIX, and Windows operating systems, and have the look and feel of the respective operating system. That is, if the client is installed on a Windows-based workstation, then it will look and behave like an application on Windows.

Graphical metaphors are used in the workstation-based client wherever possible. That is, behavior such as dragging and dropping, double-clicking to perform an action, and right-clicking to receive additional menus is possible in the workstation-based client.

23.4.1 Configuring the workstation-based client

Once you have completed the installation of the appropriate operating system client necessary, you must set up a connection to the host.

If the SCLM Admin Toolkit does not open with the Admin Toolkit perspective, then in the menu bar above, click the option **Window**, then click **Open Perspective**. The Perspective SCLM Administrator Toolkit will be listed as an option in the drop-down window. If it does not, select the option **Other** rather than Open Perspective. An additional window will appear listing the available perspectives to open.

Making a host connection

To make a host connection, follow these steps:

1. In the Remote Systems View in the left hand side of the pane, double-click the icon **New Connection** to expand the node.
2. Click **z/OS System** to bring up the New Connection Wizard to create a new connection to a z/OS operating system.
3. When the window titled, Remote z/OS System Connection, opens, enter a host name. The host name is the connection to the z/OS LPAR. Notice that the field Connection Name is automatically populated with the text entered in the Host Name field.
   - The Connection Name field is the value displayed in the Remote Systems view, so it can be changed to something different.
   - For example, the Host Name to connect to a company could be 10.19.157.180, but the Connection Name can be changed to “My LPAR” instead.
   - The Connection Name is arbitrary and can be called anything.
4. Once the Host name has been provided, click **Next**. The Files window appears with the tab Server Launch Settings displayed. You have the option of using REXEC to communicate with USS, or an RSE Daemon.
   - If you choose to use REXEC, it must be configured and operational with OMVS.
   - If you choose to use the RSE Daemon, it must be configured and operational, and an available port must be specified. The default port value of 4035 is a suggestion.
5. To use REXEC, highlight the **REXEC** radio button and enter the path `usr/lpp/AUZ/v310/rseserver710`. Change the Server Launch Script field to server.zseries.

Note: Check with the system administrator who completed the installation of the Admin Toolkit to ensure that the correct USS path is specified here.

6. To use the RSE Daemon, highlight the **RSE** radio button and set the port to **4035**, which is the default port number specified in the customization jobs.

   Note: Check with the system administrator who completed the installation of the Admin Toolkit to ensure that the correct port number is specified here.

7. Then click **Finish**. The new z/OS connection will appear in the Remote Systems view on the left-hand side of the application.
8. To connect to the new system, expand the connection node, and double-click **SCLM Admin Toolkit**. Enter your user ID and password to log into the host.
9. Once the user ID and password are verified, you are ready to begin using the Administrator Toolkit.

23.4.2 Listing projects in the project filter

To list projects, follow these steps:

1. Once you have successfully logged into the host, expand the SCLM Admin Toolkit node in the Remote Systems view.
   - A default project filter is already provided with a filter of asterisk ('*'), but its filter value can be changed.
   - See the section titled, "Adding a project filter to a host connection" for more information on how to create a new filter for your host connection.
2. Double-click the **Project Filter** node to see all SCLM projects. By default, the project filter is set to an asterisk and displays all SCLM projects available.
   - The Admin Toolkit has real-time access to the SCLM repository, ensuring that all SCLM projects you have authority to see are displayed in the Project Filter.
   - A project is considered a "project" when data sets <project>.PROJDEFS.LOAD and <project>.PROJDEFS.SOURCE contain members with the same project name.
   - Projects with a blue square icon are complete projects. Projects with a split blue square are incomplete projects.
   - The Administrator Toolkit considers a project to be valid if it has a <project>.PROJDEFS.LOAD(<project>) data set.
   - If a project also has a <project>.PROJDEFS.SETTINGS(<project>) data set, then it has been parsed by the Administrator Toolkit.

23.4.3 Adding a project filter to a host connection

To add a project filter, follow these steps:

1. To add project filters with different filter values, right-click on SCLM Admin Toolkit and select New, then select **Project Filter**.
2. A window titled, Filter, pops up asking for the filter value to use. Enter a value and click **Next**.
3. Provide a name for the filter in the field Filter Name and leave the default value in the field Parent Filter Pool. Click **Finish**.
4. You will see your new Project Filter appear in the expanded SCLM Admin Toolkit node in the Remote Systems view.

Archived

23.4.4 Administering a project using the graphical user interface

To administer a project, follow these steps:

1. Expand the Project Filter under the SCLM Admin Toolkit node to see a list of projects.
2. Select the one you want to work with by double-clicking the project name.
3. When the project opens, there will be tabs across the bottom of the project frame for each of the functions of the Admin Toolkit.
   - When editing an existing project or cloning a project, you can randomly maneuver through the tabs across the bottom of the frame to edit and change parameters as you need to.
   - When creating a new project, follow the tabs from left to right to specify all necessary options, thereby completing a new project.
4. All SCLM macros are supported in the Admin Toolkit when creating new projects or editing existing projects. Incomplete project definitions can be saved and completed at a later time.

23.4.5 Creating a new project on the client by cloning an existing one

When the need arises to create another project, it is always more efficient to copy, or clone, an existing project and then edit the project definitions as necessary. This helps create a new project with minimal effort and in a fraction of the time it would take to create a new one from scratch.

To clone a project, follow these steps:

1. Open the project you want to clone.
2. When the project view opens in the main frame of the application, select the option **SCLM** from the menu bar above.
3. When the drop-down menu appears, select **Clone Project**.
4. A dialogue window will appear titled, Clone SCLM Project. The cloned project name will appear in the window as well as a pre-populated field containing the name of the new project.
5. Change the value in the field New Project Name to the name you want to call your new cloned project.
   - The field, Include Project Data, means that the underlying application source code data sets and their contents will also be copied.
   - Otherwise, just the <project>.PROJDEFS.* data sets will be copied.
6. Once you click **OK**, the project is created to include the associated data sets, and a Project Frame will appear displaying the newly created projects' definitions. You can choose to edit the project definitions, or build the new project.
7. To build the project, click on **SCLM** in the menu bar above. in the drop-down menu, select **Build/Rebuild**. The project is now complete and can be used by SCLM.

23.5 Using the ISPF-based interface

The ISPF-based user interface is invoked by executing the CLIST AUZ or by selecting the appropriate option from the SCLM Main Menu.

23.5.1 Listing projects in the ISPF panels

To list projects, follow these steps:

1. Upon entering the Administrator Toolkit Main Menu:
   - Projects can be listed by placing a partially qualified project name in the Project Name field.
   - An asterisk can be used as a wild card character to filter the list of projects that will be displayed.
2. Once the Project List panel displays the resultant projects that match the pattern provided, a project in the list can then be opened for editing.

23.5.2 Navigating through the ISPF panels

To navigate the panels, follow these steps:

1. Depending on whether you are creating or editing a project:
   - If you are creating a new project, you will typically navigate through the panels of the Admin Toolkit sequentially by completing the entries on a given panel and pressing Enter to move to the next panel in the series.
   - If you are editing an existing project, you can use the Quick Navigation Menu (QNAV) to jump from one function to another. This permits you to work in a manner similar to that provided by the project tabs in the Project Editor in the workstation-based client.
2. As you exit from the user interface, the project's options are automatically saved in the PDML that is stored in the '<project>.PROJDEFS.SETTINGS' data set.
3. When all of the necessary options have been specified, you can build the project by selecting the appropriate option from the QNav menu or by using option **L** from the Project List panel.

23.5.3 Administering a project in the ISPF-based user interface

To administer a project, follow these steps:

1. To bring up a list of valid SCLM projects, select option **3** from the SCLM Administrator Toolkit Main Menu and provide a value for the field Project Name. Then press Enter.
2. When the Project List panel appears, select the project you want to work with by typing an **E** next to the project name. When the project opens, the QNav menu will be displayed allowing you to select the function you want to edit.
   - When editing an existing project or cloning a project, you can randomly maneuver through the functions on the QNav menu to edit and change parameters as you need to.
   - When creating a new project, follow the functions from the QNav menu in numerical order to specify all necessary options, thereby completing a new project.
3. All SCLM macros are supported in the Admin Toolkit when creating new projects, or editing existing projects. Incomplete project definitions can also be saved and completed at a later time.

Archived

23.5.4 Creating a new project on the host by cloning an existing one

When the need arises to create another project, it is always more efficient to copy, or clone, an existing project and then edit the project definitions as necessary. This helps create a new project with minimal effort and in a fraction of the time it would take to create a new one from scratch.

To clone a project, follow these steps:

1. Type a **C** next to the project you want to clone and press Enter.
2. A pop-up window will appear titled, Clone SCLM Project. The cloned project name will appear in the window as well as a pre-populated field containing the name of the new project.
3. Change the value in the field New Project Name to the name you want to call your new cloned project.
   - The field,, Include Project Data means that the underlying application source code data sets and their contents will also be copied.
   - Otherwise, just the <project>.PROJDEFS.* data sets will be copied.
4. Once you have filled in the appropriate fields, press Enter.
5. The project is then created to include the associated data sets, and the new project will appear in the Project List panel.
   - You can choose to edit the new projects' definitions by typing **E** next to the new project, or build the new project by typing **L** next to it.
   - Once the project has been successfully built, the project is complete and can be used by SCLM.

23.6 For more information

To learn more about SCLM Admin Toolkit for z/OS, refer to the *User's Guide* and the *Installation and Customization Manual*, at:

http://www-306.ibm.com/software/awdtools/sclmsuite/admintoolkit/library

Archived

24

Introduction to the IBM SCLM Administrator Toolkit

In this chapter we focus on each of the components of the Administrator Toolkit (Admin Toolkit). We give you a more in-depth explanation of the automated functions that the Admin Toolkit provides to an SCLM project administrator.

The topics covered in this chapter include:

- Who should use this product and why
- The Project Editor
- The Language Definition Wizard
- The Clone Project Utility
- The Migration and Remote Migration Wizard
- The Architecture Definition Wizard
- The Enhanced Access Control Manager (EAC) and Resource Access Control Facility (RACF) Data Set Profile Feature

© Copyright IBM Corp. 2007. All rights reserved.

24.1 Who should use this product and why

SCLM project administrators typically create their projects by defining project definitions and languages from assembler macros. They may have some canned JCL to allocate a number of data sets. Over time as SCLM projects change (either new groups are added or deleted, authorization codes change or perhaps new data set types are added), administrators have to change the project definitions and the project data sets.

Project accounting and auditing data sets hold information about revision, history and dependencies, and information about SCLM repositories. These data sets have to be reorg'd from time to time and it is the responsibility of the project administrator to maintain the accounting and auditing data sets as well.

Administration tasks are time-consuming and typically are performed by seasoned programmers. Experienced project administrators may have a well-tuned process for managing their projects, but that usually means new administrators cannot easily step in and maintain project changes.

The SCLM Administrator Toolkit was developed to assist experienced SCLM administrators by minimizing the amount of time it takes to complete administration tasks and by automating or simplifying common tasks they perform. For new SCLM administrators, the Toolkit is intended to make SCLM understandable and simpler to administer, thus reducing the time it takes for a new administrator to become productive and to assist with project administration tasks.

Manual tasks usually performed by senior programmers are automated by the Admin Toolkit, reducing the time it takes to maintain existing projects. The user-guided interface contains content-sensitive fields so that conflicting parameter values cannot be entered when creating new projects. Content-sensitive fields help to eliminate definition errors within projects and gets the project built and into production faster and easier than defining a project manually.

Using the Administrator Toolkit to automate most manual processes frees project administrators to move on to other tasks.

24.1.1 Two user interfaces

IBM SCLM Administrator Toolkit V3.1.0 provides two interface options. The host-based user interface is standard with the installation of the Admin Toolkit and installs on z/OS operating systems V1.6 or higher. The optional workstation-based user interface can be installed on Windows operating systems running Windows 2000 or Windows XP, Linux operating systems running Linux GTK, and AIX operating systems running AIX 5L.

Both the ISPF interface and the workstation-based client interface have the same task flow in that each contains the same options on the same panels. This allows users to navigate through either interface with more comfort, should their preferred interface be unavailable.

ISPF-based user interface

Using standard ISPF panels, the ISPF-based user interface is intuitive with the ability to execute many actions from one panel using a combination of Primary Commands and Line Commands.

Primary Commands are those options that are entered on the command line. line Commands are options entered next to objects in a list. Figure 24-1 displays an ADmin Toolkit panel that contains both Primary Commands and Line Commands.

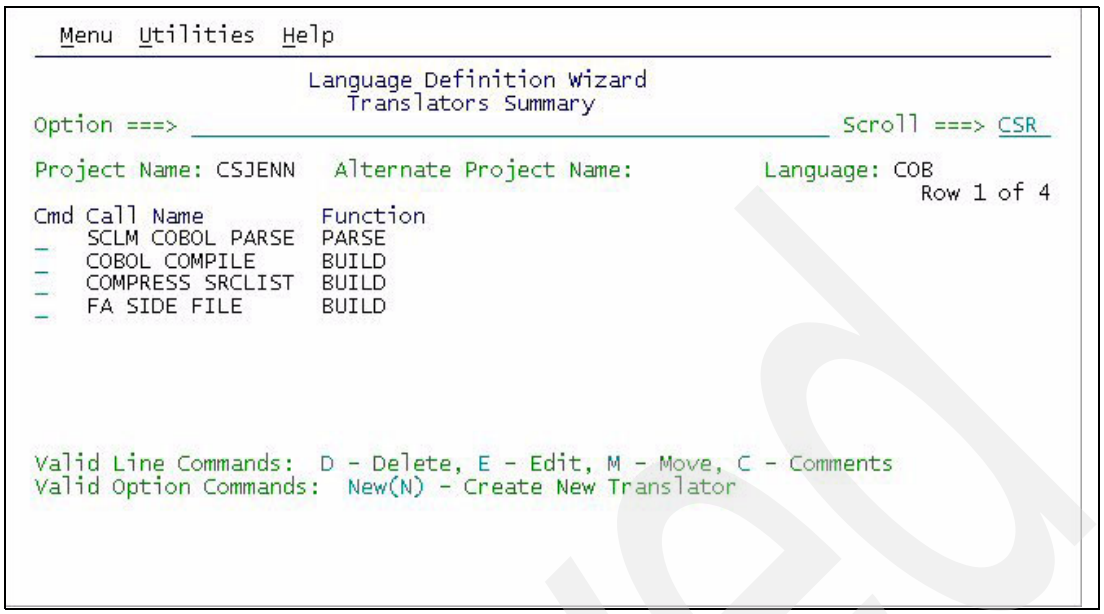


Figure 24-1 Panel displaying both Primary Option Commands and Line Commands

Primary Command **N** is specified on the Command Line to navigate to additional panels which are used to manipulate portions of the project, in this case, adding a new translator to a language definition. Line Commands **D**, **E**, **M**, and **C** are specified next to an object within the Translator Summary panel to perform a specific action against that object.

As you move from panel to panel in the ISPF interface by pressing Enter, the panel options are saved in the project's PDML in data set '<project>.PROJDEFS.SETTINGS(<project>)'. Even if panel options are not changed, the project's options are still updated in the SETTINGS data set.

Navigating from one function to another in ISPF can easily be done by using the Quick Navigation Menu, or QNav menu. The QNav menu contains a numerical list of functions that are used to create an SCLM project. On any of the main function panels in the Admin Toolkit, you can press PF4 to display the QNav menu and jump to another function.

Workstation-based user interface

The workstation-based interface can be installed by FTP'ing the appropriate member from data set 'AUZ.SAUZGUIW' for Windows, data set 'AUZ.SAUZGUIK' for Linux, or data set 'AUZ.SAUZGUIA' for AIX to your PC's hard drive. The PDS members are self-extracting executable programs. Once they are downloaded into your PC's hard drive, the .bin extension should be renamed to .exe and executed as a program.

The InstallShield Multiplatform Installer will install the Admin Toolkit into a suggested directory that can be changed by the user. The appropriate version of Java as well as a pre-existing installation of Eclipse v1.3.2 or higher will be searched. If Java is not found or an older version of Java is located, Java 1.4.2 will be installed into the Admin Toolkit directory. If no pre-existing installation of Eclipse is found for the Admin Toolkit to be installed as a plug-in, or if a version of Eclipse older than v1.3.2 is found, then Eclipse will be installed as well, and the Admin Toolkit will be installed as a stand-alone application.

Navigating through the workstation-based user interface is performed through a series of tabs in the Project Editor view within the Administrator Toolkit's perspective. Each of the tabs is a function that is used to create an SCLM project and can be accessed by simply clicking the tab. Figure 24-2 shows the Project Editor view.

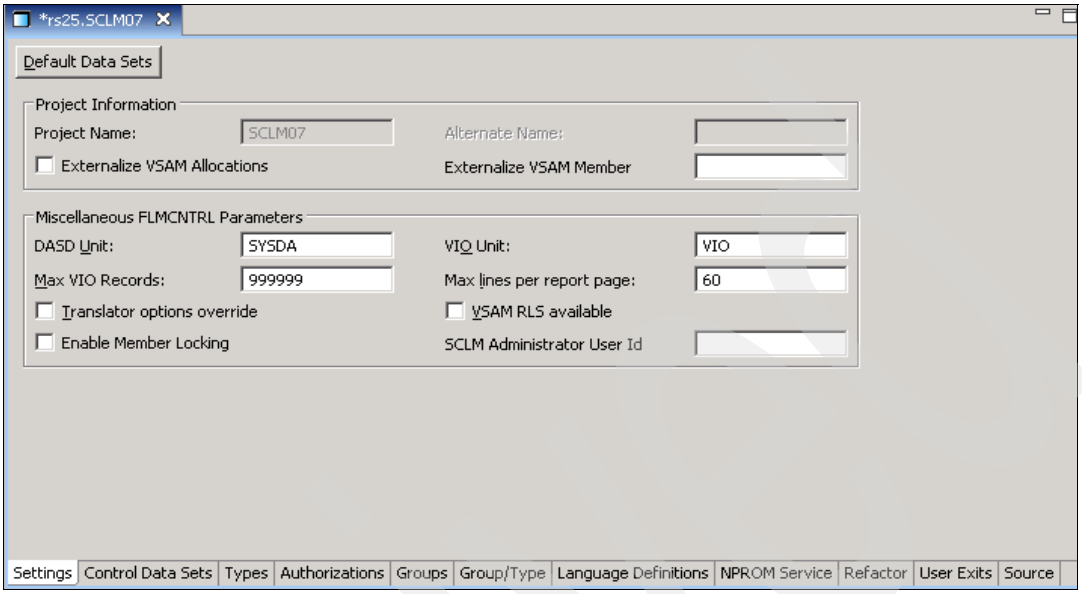


Figure 24-2 Use the tabs at the bottom of the Project Editor view to navigate through the functions

As you edit options in your project using the client, the changes are not automatically saved as they are in the ISPF user interface. Project changes are only saved when the **Save** option is selected under File from the Menu Bar or when the disk icon is selected on the Tool Bar.

Archive

24.2 Project Editor

The Project Editor is used to edit existing projects or to create new projects. When invoking the Project Editor in the client, all SCLM macros are editable for both new projects and existing projects.

Using the Project Editor, you can define data set types, package backout parameters, and groups. If you need to modify an existing project and want to delete a data set type or project group, the Project Editor will, after confirmation from you, delete any data associated with that type or group.

When defining a project hierarchy using the workstation interface, drop boxes representing project groups can be dragged around the display to build a network of linked toolboxes, much like an organization chart. Once the definition is completed, Admin Toolkit generates an SCLM project hierarchy.

24.2.1 Specifying project settings and control data sets

The Project Editor in the client contains two tabs that help define the project itself. The Settings tab contains the Default Data Sets, which tell SCLM where the project definition source is located, as well as what SYSLIB data sets contain language definitions or macros that will be used by the project.

Figure 24-2 on page 590 shows the Settings tab with the Default Data Sets button that, when clicked, displays the PROJDEFS data sets used by the project to contain project-specific definition information.

Figure 24-3 shows the Default Data Sets Dialog window that is displayed as a result of clicking the button **Default Data Sets** from the Settings tab. The data set names are suggestions only and the low level qualifier can be edited. It should be noted that the high level qualifier should remain the project name.

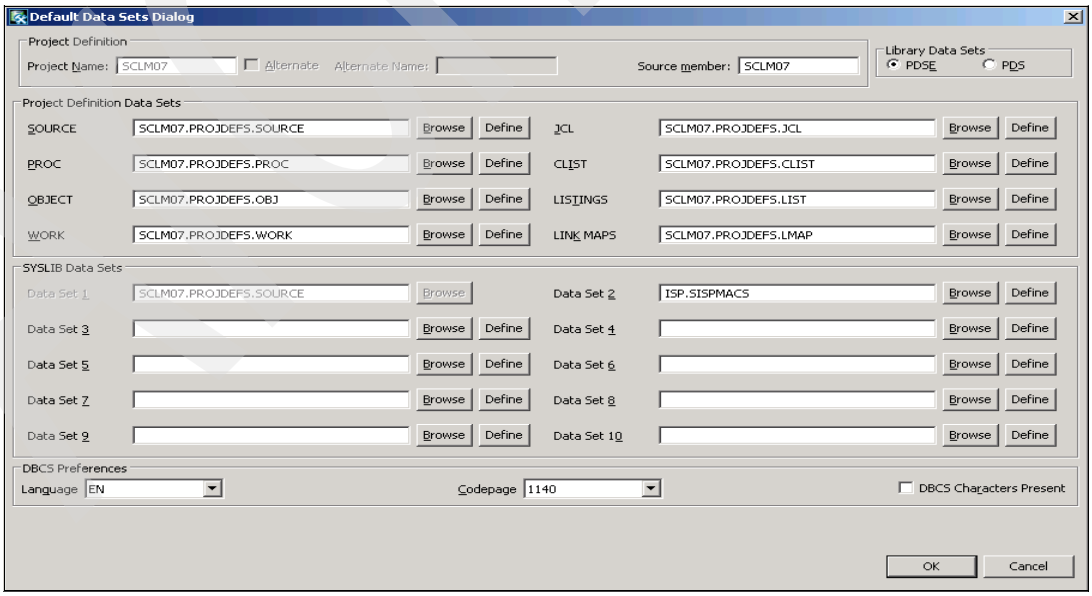


Figure 24-3 Default Data Sets Dialog window from within the Settings tab

The second SYSLIB data set is ISP.SISPMACS by default, but may be edited to reflect your shop's naming standards. If additional macro data sets are needed, they may be added in fields Data Set 2 through Data Set 10.

Figure 24-4 shows the Control Data Sets tab that contains the project control information, as well as alternate control information. Select the appropriate radio button to edit or add Primary Control Information or Alternate Control Information.

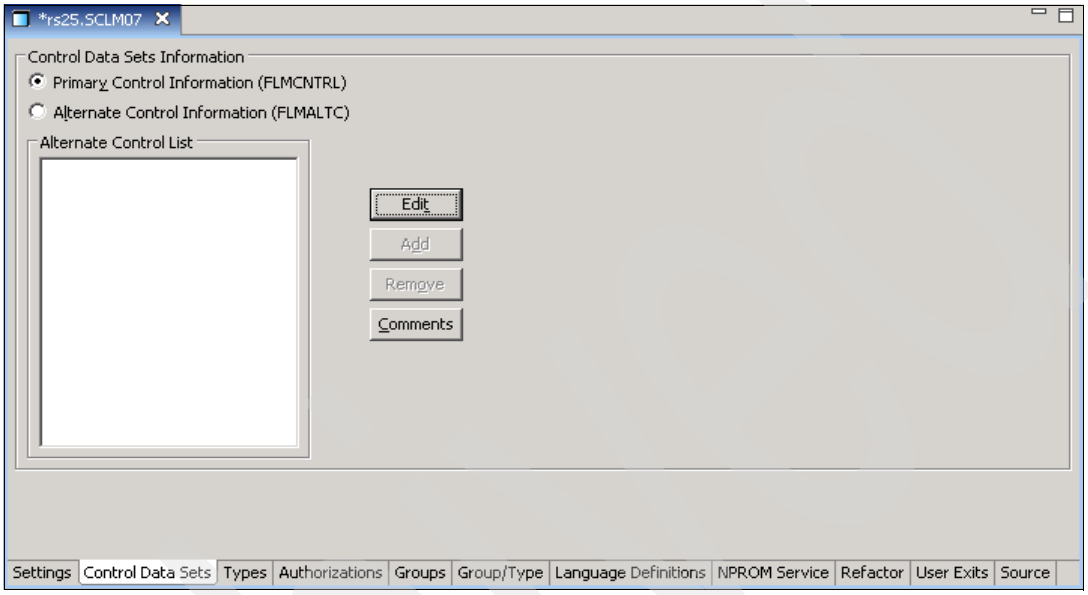


Figure 24-4 The Control Data Sets tab allows you to specify project control information

By clicking the **Edit** button, as shown in Figure 24-5, you can edit the project's primary control information, such as accounting and auditing files, and the versioning data set pattern to use and the number of versions to retain.

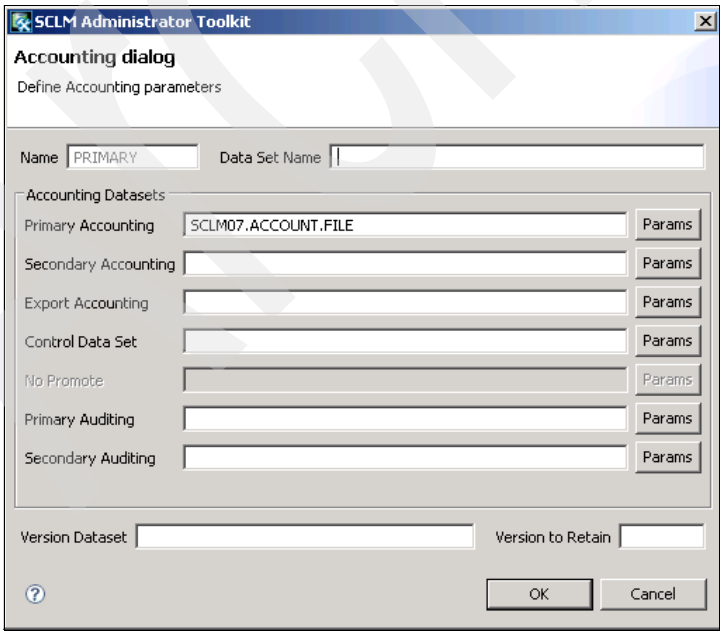


Figure 24-5 All project control information can be edited

24.2.2 Specifying data set types

When developing an application, different data set types will be used to contain the application's source code. That is, Java source code will be kept in a separate data set from COBOL source code. To define the types of data sets to allocate for your project, select the Types tab in the Project Editor.

Figure 24-6 displays two frames. The frame on the left contains the predefined data set types that are available to add to your project. The frame on the right contains the types of data sets that have been included in your project.

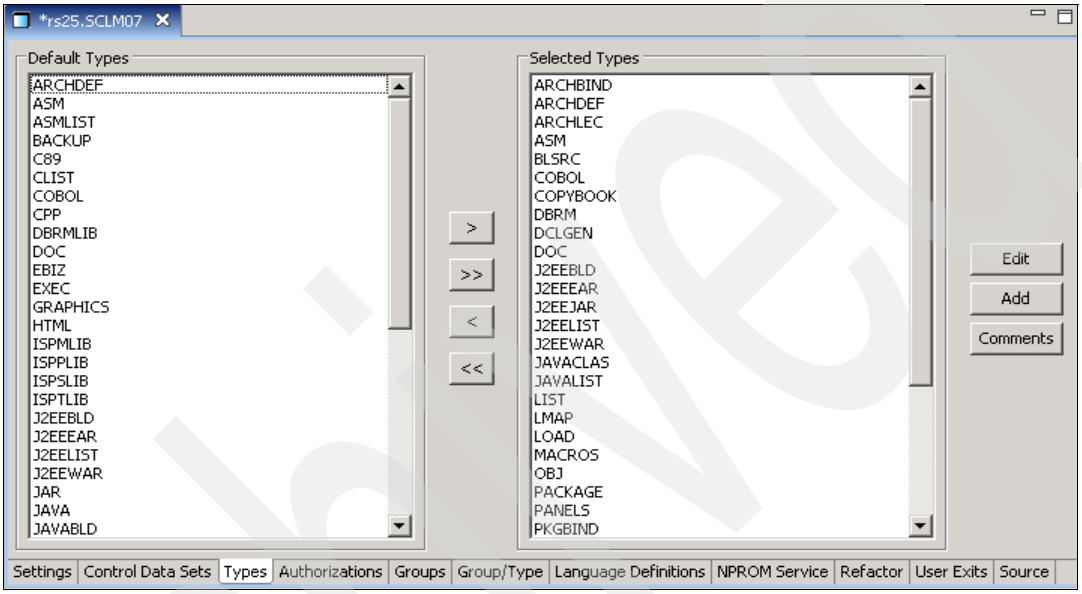


Figure 24-6 The Types tab allows you to select data set types to include in your project

To add a data set type to your project, highlight the data set in the left-hand frame and click the right arrow button (**>**) to move it to the right-hand frame. To remove a data set type from your project, highlight the data set in the right-hand frame and click the left arrow button (**<**) to move it to the left-hand frame.

To add or remove multiple data set types, highlight each of the data set names while holding down the Control key on your keyboard. Then click the right arrow button (**>**) or the left arrow button (**<**) to move the data sets from one frame to the other. To add or remove all default data set types, simply click the double right arrow (**>>**) to add all of them, and click the double left arrow (**<<**) to remove them.

24.2.3 Specifying groups

Groups within a project are connected much like an organizational chart in that the project hierarchy tells SCLM where to move the application's source code as it is built. This process is called "promoting code". When SCLM promotes source code, it must know information such as which development group is allowed to promote code upward to another group, which members are to be restored, and if any alternate project control options are to be used with a particular group.

In the client interface, Group boxes for your project can be created by providing a value in the field Group Name, and clicking the button, **Add Group**. Once you have add the groups to be included in your project, the group boxes can be dragged around the workspace in the Groups tab to create your project's hierarchy. Figure 24-7 shows the workspace in the Groups tab with a graphical representation of the promotion hierarchy of your project.

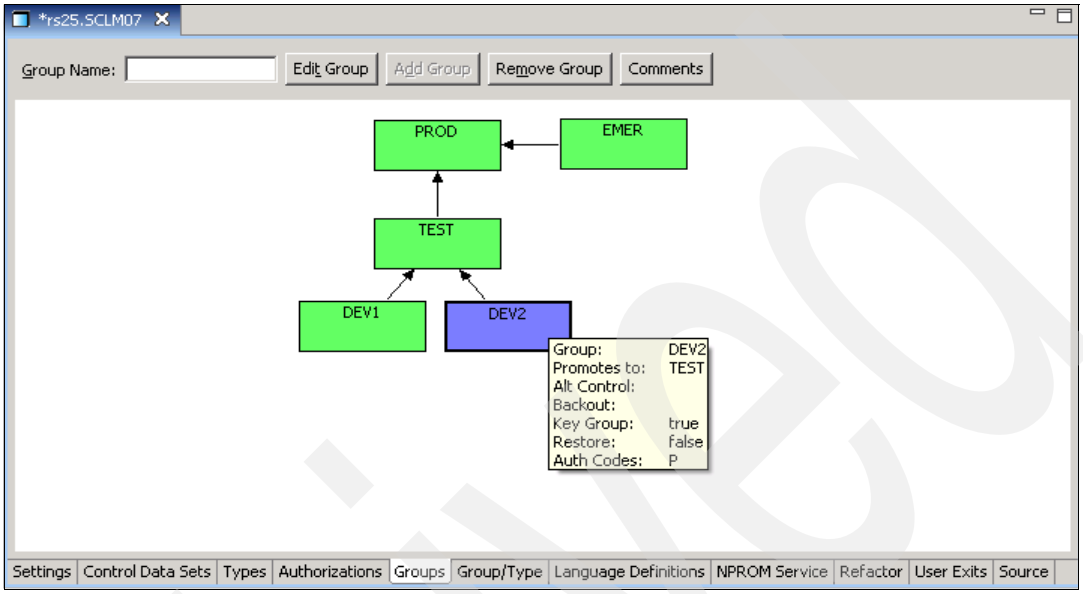


Figure 24-7 The Groups tab allows you to drag-and-drop boxes to create a hierarchy

You can change the direction of promotion in your project in one of two ways. The first way is to click a group box so that it is highlighted in blue. Right-click the highlighted group box, then right-click another group box that you want to lower group to promote code to. An arrow will appear to indicate the direction of promotion. To change the promote direction, repeat the process.

The second way to establish the promotion hierarchy is to double-click a group box to bring up the Group Dialog window, displayed in Figure 24-8. The promotion hierarchy, as well as other group definition information, can be defined from this Dialog window.

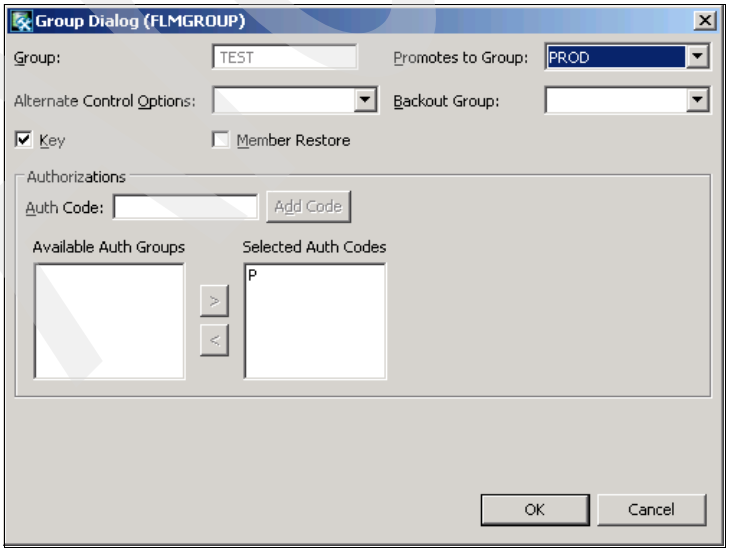


Figure 24-8 Group options are specified in the Group Dialog window

The Groups tab also allows you to view a group's definition at a glance by clicking a group box to turn it blue, then hovering your cursor over the blue group box. A small pop-up will appear with the group's definition information, as shown in Figure 24-7 on page 594.

To delete a promotion direction, double-click a group box to see the Group Dialog window. Then set the field Promotes To Group to a blank value, and click **OK**.

24.2.4 Specifying group and type data sets

In the subsection "Specifying data set types" on page 593, we discussed how to select the types of data sets to include in your project. In the Group/Type tab, you select the combination of data sets you want to include in your project.

Figure 24-9 looks similar to the Types tab in that there are data set types in the left-hand frame that are available to add to your project, and data set types in the right-hand frame that have been selected to add to your project.

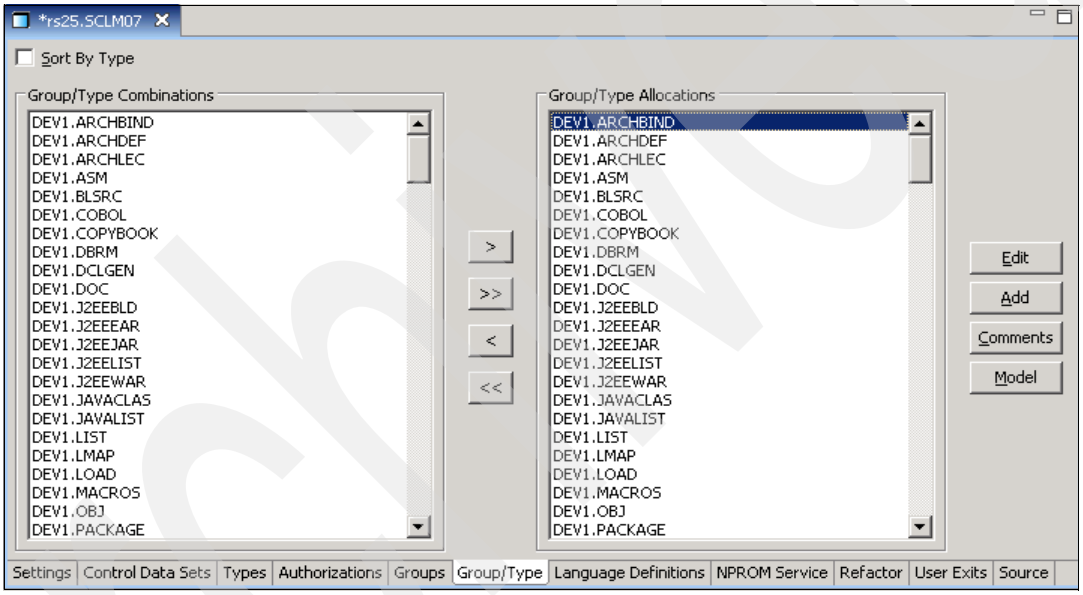


Figure 24-9 Group/Type combination data sets are selected from the Group/Type tab

The difference between the Group/Type data set combinations in the left-hand frame and the selected Group/Type data set allocations in the right-hand frame is that the data sets in the right-hand frame will be created by the Admin Toolkit and included in your project.

In our example, the available Group/Type combinations have all been selected to be allocated for our project, so the Group/Type data sets in the left-hand frame and the right-hand frame are the same. This means that an ARCHBIND type of data set, an ARCHDEF type of data set, and an ARCHLEC type of data set will be created for each of the groups.

However, there may be projects in which not all types of data sets will be needed by each group. For example, group DEV1 may not need an ASM data set type, but group DEV2 does. In this case, you can remove Group/Type data set DEV1.ASM from the right-hand frame by highlighting the data set name and clicking the left arrow button (<), preventing it from being created. Or, for example, group DEV2 may not use a COBOL data set type, but group DEV1 will. In this case, the Group/Type data set DEV2.COBOL may be removed from the right-hand frame so that it is not allocated.

24.2.5 Specifying language definitions

Language Definitions tell SCLM how it should process an asset in the project during a build, a promote, etc. Language Definitions can be very complex and involve many components in order to properly handle all components of a project. The information provided in this subsection is a basic explanation of using the Language Definition tab in the client interface. For a more in-depth description of using the Language Definition Wizard, see the section “The Language Definition Wizard” on page 609 in this chapter.

The Language Definitions tab, as shown in Figure 24-10, allows you to edit, add, or remove language definitions by clicking the appropriate button on the right-hand side of the tab. In addition, you can activate and inactivate languages by highlighting the desired language definition and clicking the **Switch Active** button.

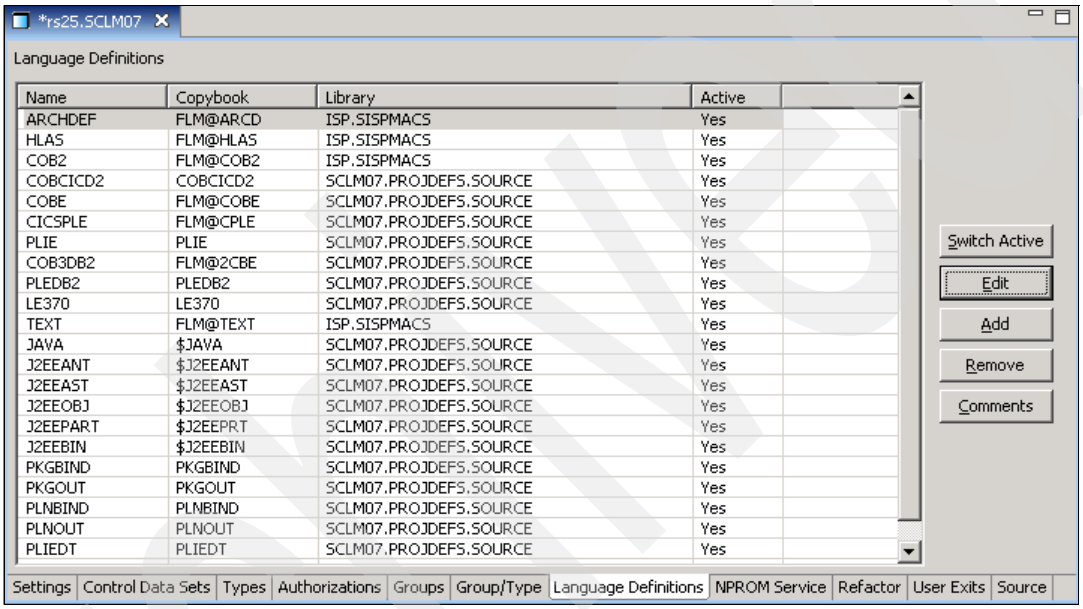


Figure 24-10 The Language Definitions tab at-a-glance

Archived

To edit an existing language definition, simply highlight a language definition in the list and click the **Edit** button. The Language Definition Wizard window appears, as shown in Figure 24-11, and allows you to edit all SCLM macro parameters by navigating through the Wizard using the buttons Back, Next and Finish at the bottom of each Wizard window. At any time, you can cancel out of the Language Definition Wizard without making any changes to your language definition by clicking the **Cancel** button.

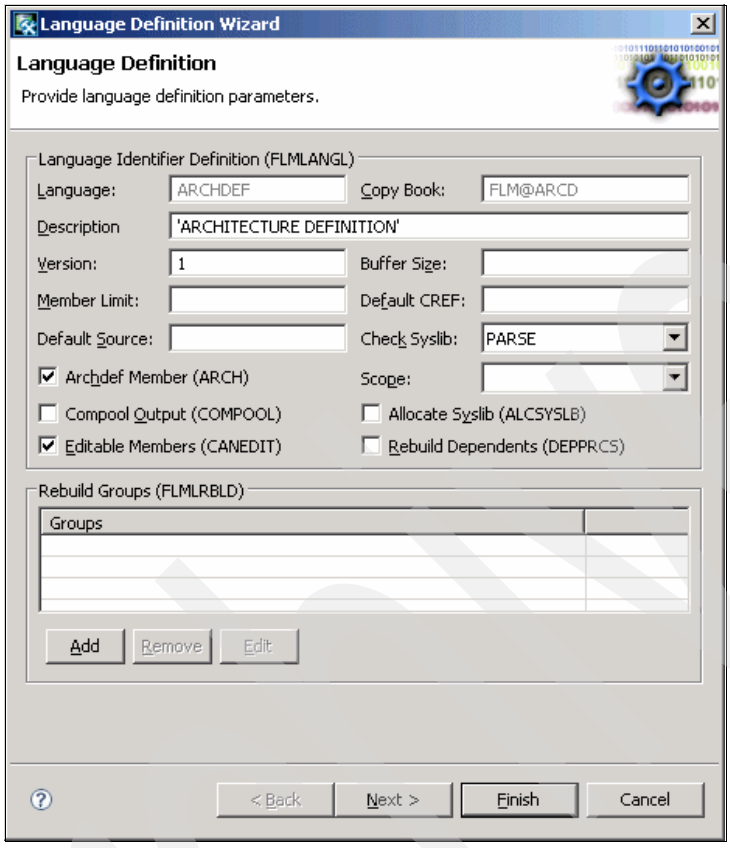


Figure 24-11 Edit an existing Language Definition

Archived

To add a new language definition to a project, simply click **Add** on the right-hand side of the panel as shown in Figure 24-10 on page 596 to display the Language Definition Wizard. The Wizard will allow you to create a new language definition in one of three ways, as shown in Figure 24-12.

The first way to add a new language definition to your project, and perhaps the most efficient, is to copy an existing language definition and then edit the definition's parameters to suit the needs of the project. In some cases, no editing will be required, in which case, the language definition is simply added to the project with a reference to the data set it can be found in.

The second way to add a new language definition to your project is to create it from existing JCL. If you have third party vendor tools and your languages exist in a JCL job, the Admin Toolkit can parse through the JCL to create a language definition for you. Once the parse has completed, the newly parsed language definition will be displayed for your review. Any edits or changes to the language definition can be made at that time.

The third way of creating a language definition is to create it from scratch by navigating through each of the Language Definition Wizard panels using the navigation buttons Back, Next and Finish as shown in Figure 24-12 and Figure 24-13. While this is a more tedious method to create a language definition, the Admin Toolkit's Wizards all contain content-sensitive fields which prevent conflicting parameters from being entered.

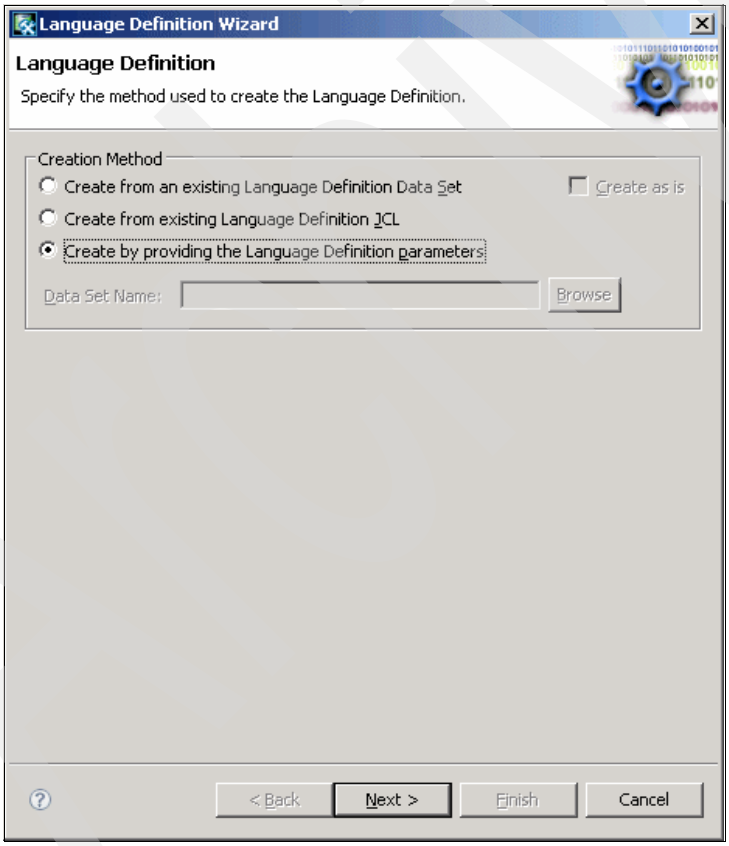


Figure 24-12 Creating a new Language Definition can be done 3 ways

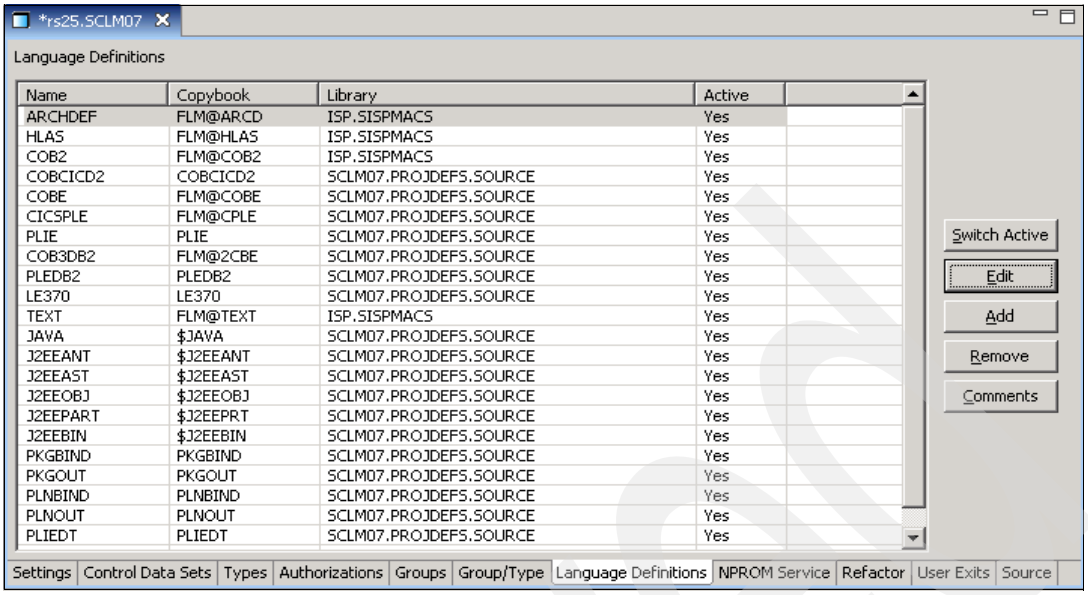


Figure 24-13 Column names help identify the language definitions in your project

The Language Definitions tab contains columns to help identify the languages in your project. The column Name is simply the name of the language definition in your project.

The column Copybook is the name of the PDS member of the language definition. The column Library tells the Admin Toolkit where to find the language definition. It may be the case that the language definition is a generic definition provided in the ISPF macro library, the project's source library or another library. This column lets you know at a glance where the language definition is located.

The column Active tells you if this language is in use or if it is commented out. If it is in use, that is, if the value is Yes, then it is utilized by base SCLM when building the associated application source code. If it is not in use, that is, if the value is No, then the language definition is commented out in the project definition source and base SCLM will not use that language definition to build artifacts.

The Switch Active button on the right-hand side of the frame allows you toggle between uncommenting and commenting out a language definition from the project source. This permits you to have an incomplete Language Definition in your production project without creating an error condition or other possible problems.

Archive

24.2.6 Specifying NOPROM service

The NOPROM, or Not Promote, service was an enhancement introduced into SCLM that provides the ability to leave behind (or not promote) an SCLM editable member, but still promote the SCLM components that use that member or were built using that member. If you want an editable member to not be promoted and you are on the appropriate level of SCLM with the correct maintenance applied, you will need to invoke a new service called NOPROM. The NOPROM service will update the accounting status to specify that the member is not promotable.

The required SCLM release level and corresponding maintenance are listed in Table 24-1.

Table 24-1 Required SCLM level to use the NOPROM Service

SCLM Release Level	SCLM Maintenance
SCLM for ISPF for z/OS V1.6.00	5.603aa99999
SCLM for ISPF for z/OS V1.7.00	5.703aa99999
SCLM for ISPF for z/OS V1.8.00	5.803aa99999
SCLM for ISPF for z/OS V1.9	5.900

The example project SCLM07 was not built on an SCLM release level with the NOPROM service available. Therefore you cannot configure the NOPROM service. The NOPROM tab is grayed out as shown in Figure 24-14.

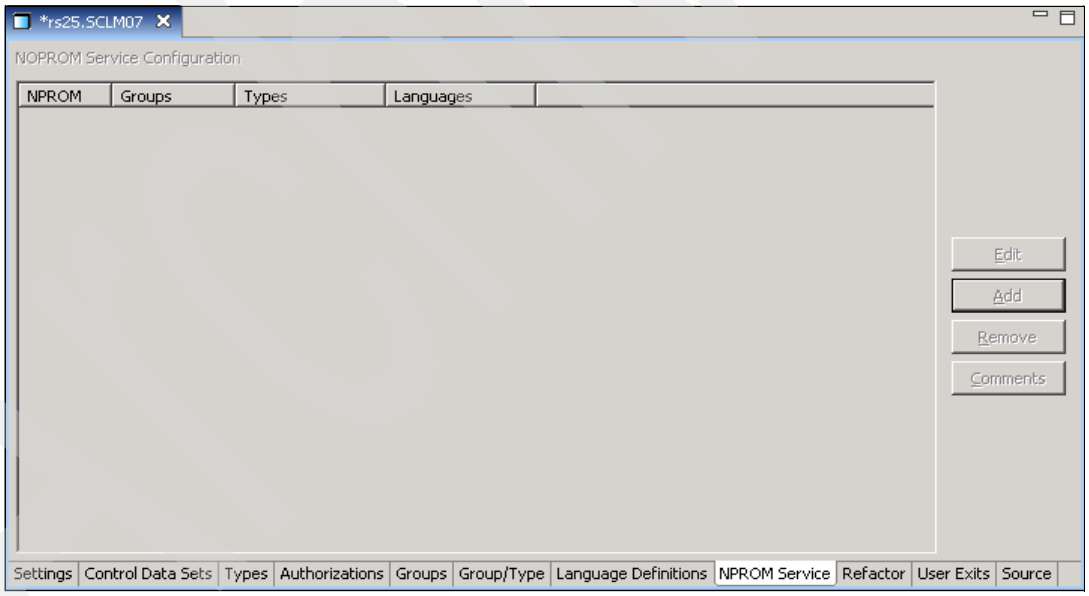


Figure 24-14 The NOPROM Service is not configurable for the example project

24.2.7 Viewing the refactor tab

The term Refactor means to rewrite a program to change its structure, with the explicit purpose of keeping its meaning or behavior the same. In the case of Administrator Toolkit, the Refactor tab in the client allows you to view/edit a project definition without having to exit the product and start a mainframe emulator. In this way, you can change the structure of how the project is defined without changing how the project administers the underlying application source code

As shown in Figure 24-15 you can view and optionally edit the project's source definition using the Refactor tab. By highlighting a function such as a COPY statement or a macro, the buttons Create Copybook and Forward become active allowing you either to create a new copybook to be added to your project, to type in the name of an existing copybook you want to add to your project, or to drill down into the copybooks defined to your project.

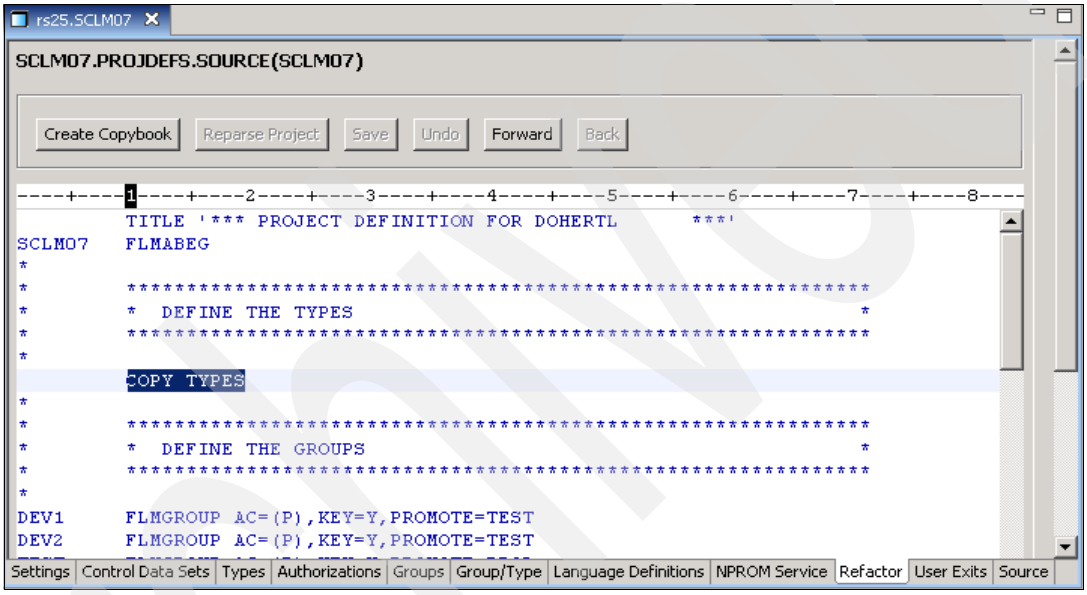


Figure 24-15 The Refactor tab lets you view or edit your project source

In our example, click the **Forward** button to edit the highlighted copybook. The name and location of the copybook you are using is displayed at the top of the Refactor tab.

Once in the copybook, make any necessary changes. In Figure 24-16, a new FLMTYPE data set was added. Once you are done making changes to your copybook, click the **Save** button to save your changes and return to the project source 'SCLM07.PROJDEFS.SOURCE(SCLM07)'.

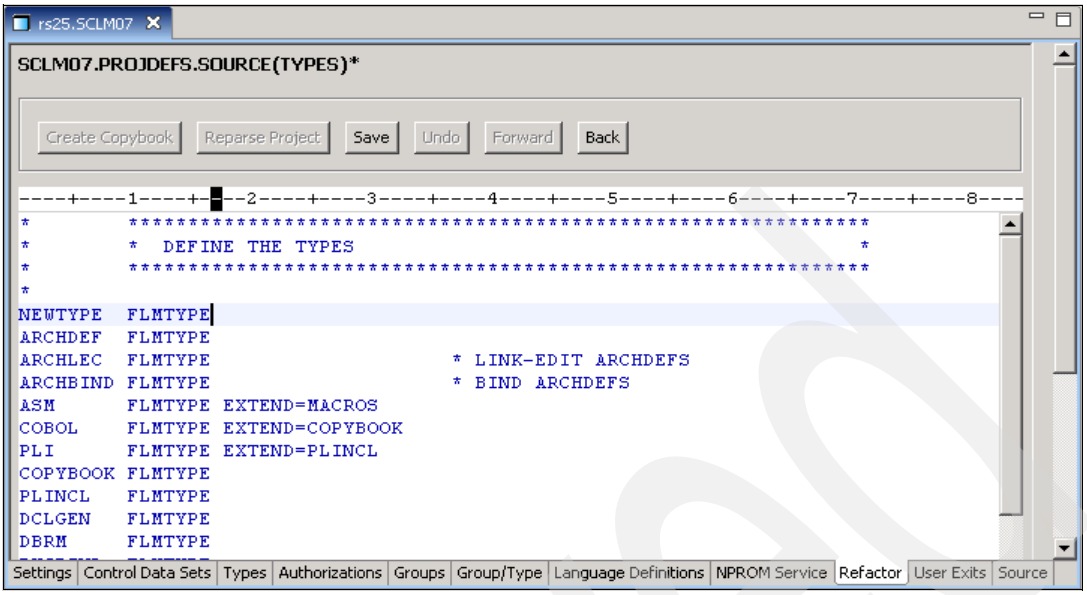


Figure 24-16 Make changes to your existing copybooks

You can drill down into a copybook that is included in your project source code by highlighting the copybook and clicking the **Forward** button. This will display the Copybook Output Selection window. The copybook you drilled down to is displayed for you to edit. In the example in Figure 24-16, a new FLMTYPE data set was added, as well as a comment line describing the edit.

When you have completed the changes to the copybook and are ready to create a new copybook, specify the name of the member that will contain the edited copybook in the field, New Member, at the left of the window and click **OK** as shown in Figure 24-17.

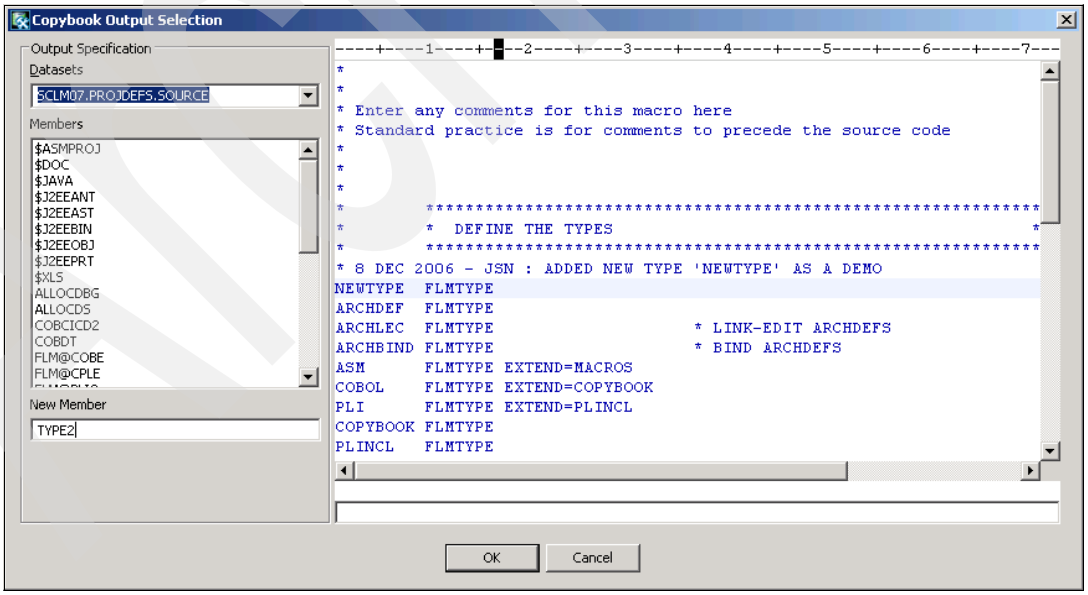


Figure 24-17 Create a new copybook by editing an existing one

When the new member is written to your source data set, the project source definition will be displayed and the new copybook that you created will appear in the project source as shown in Figure 24-18.

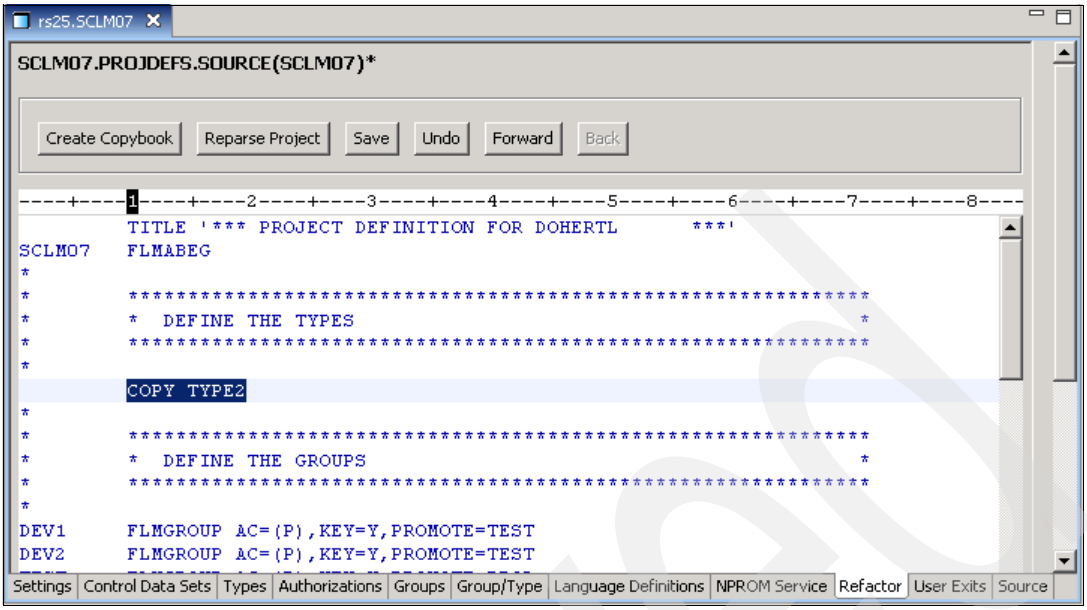


Figure 24-18 The new copybook is inserted into your project

24.2.8 Specifying user exits

Throughout the base SCLM program, there are breakpoints which allow a user exit to be called to perform additional actions, such as notifying users of actions that occur within a project or requiring that a user approve an action. User exits can process any task a user needs to have performed for a project.

For example, a user exit can be called to check a user's authority to delete a group within a project before the delete group service is called. In this way, a user not authorized to administer such changes cannot affect the integrity of a project that is in use.

To configure a project to use user exits in the workstation-based client, click the tab, **User Exits**. The SCLM Admin Toolkit supports all Breeze user exits. By default, the default Breeze data set name is populated in the field, Breeze Data Set, and can be edited to conform to your shop's naming standards. Figure 24-19 shows the User Exit tab and its options.

Archived

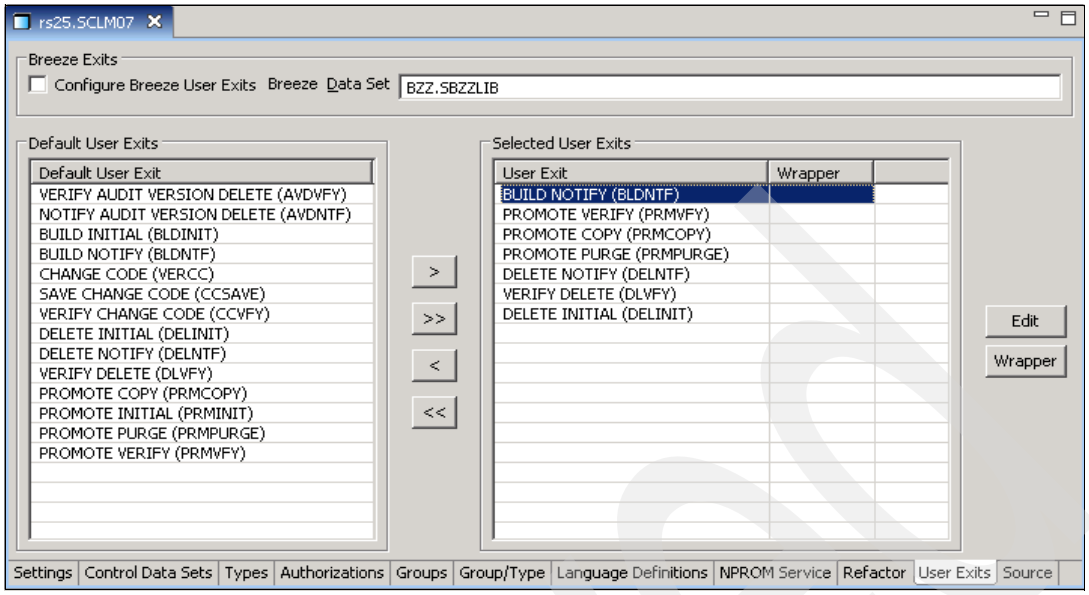


Figure 24-19 Available SCLM exit points are displayed in the Default User Exit frame

The left-hand frame of the window, Default User Exits, lists the available SCLM exit points you can use in your project. The right-hand frame, Selected User Exits, lists those exit points that will be used in your project.

To edit or view a User Exit selected to be used in the project, either double-click a **Selected User Exit** in the right-hand frame of the window, or simply highlight a user exit and click the **Edit** button on the right-hand side of the window.

Figure 24-20 displays the User Exit Dialog window that provides information about what module will be called at the specified SCLM break point, where the module can be found, how the module will be called, and what options are to be used when processing the user exit. All information is editable and may be changed from this window

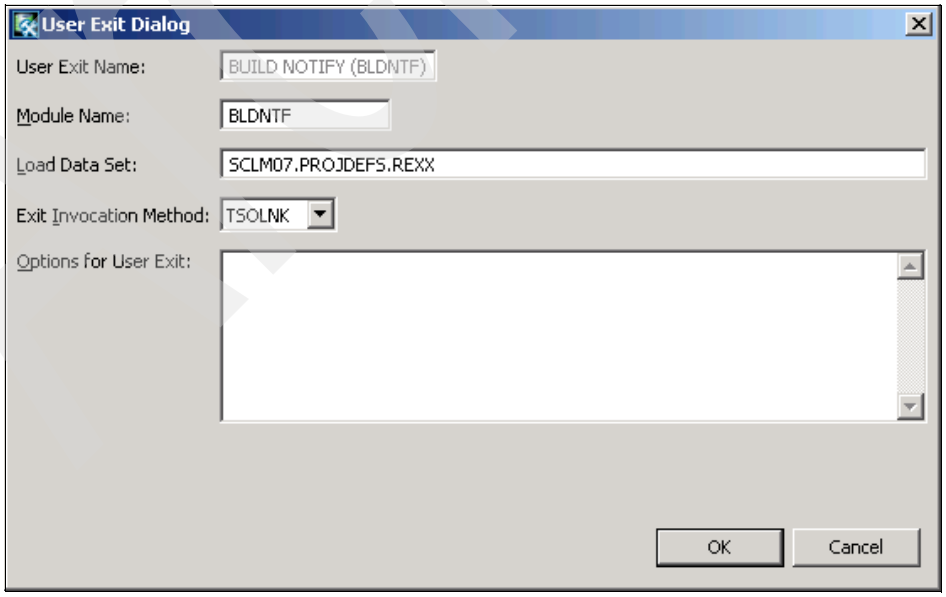


Figure 24-20 User Exit Dialog window displaying a user exit's options

You can add a user exit by highlighting a user exit point in the left-hand frame, Default User Exits, and clicking the button with the single right-hand arrow (**>**) in the User Exits tab shown in Figure 24-19 on page 604. The selected user exit will then appear in the Selected User Exit frame. Once the newly selected user exit is moved into the Selected User Exit frame, you can either double-click the user exit, or simply highlight the new user exit and click the **Edit** button on the right-hand side of the window.

The User Exit Dialog window will appear with blank fields for you to populate with the information that should be used to process the user exit.

In the event that two or more exits need to use the same SCLM program exit point, something called an Exit Wrapper will be built. An Exit Wrapper is simply a parent user exit that contains two or more child user exits that must use the same exit point. SCLM Admin Toolkit generates source code for the parent exit which includes serial calls to the child exits in the order that they appear in the source code. You can designate which user exit should be called first when creating an Exit Wrapper.

An Exit Wrapper can be created in one of two ways. The first way an Exit Wrapper exit will be created is done automatically by the Admin Toolkit when it senses that a user exit has been added to the project using the same SCLM break point as a user exit already defined to the project. In this way, the user is alerted that they have two user exits that are being used in the same SCLM break point and a Wrapper should be used.

The second way an Exit Wrapper exit will be created is by highlighting an existing user exit in the User Exits tab as shown in Figure 24-19 on page 604, and clicking the **Wrapper** button.

The User Exit Definition Dialog window shown in Figure 24-21 defines which user exits are to be included in the Wrapper exit. The order in which the user exits appear in the window is the order in which they will be called by the Wrapper exit. You can change the order in which the user exits are called by highlighting an exit in the list and clicking the **Up** and **Down** buttons.

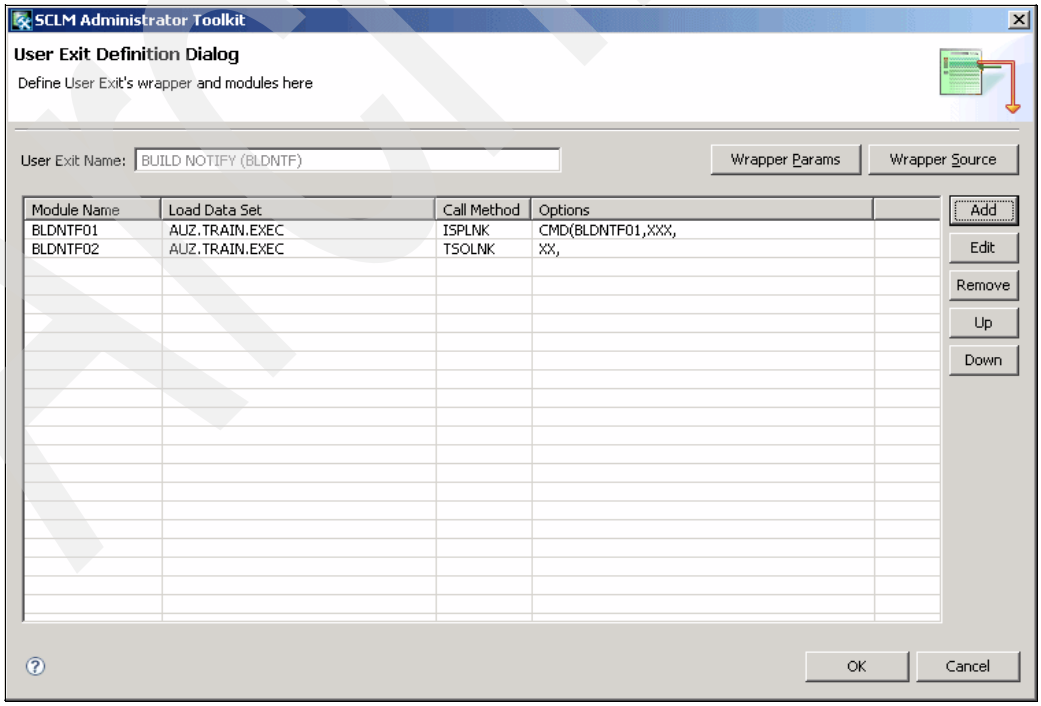


Figure 24-21 User exits that use the same SCLM program break point must use a wrapper

The buttons Add and Remove allow you to control what user exits are contained in the wrapper. The Add button will give you a blank User Exit Dialog window in which you can specify the user exit parameters to use in your project, as shown in Figure 24-20 on page 604. The Edit button allows you to edit parameters for user exits already defined to the project. The Remove button simply removes the highlighted user exit from the Wrapper and from the project.

The Wrapper Params button, on the upper right-hand corner of the User Exit Definition Dialog window, allows you to specify the invocation parameters for the Wrapper exit. When the **Wrapper Params** button is clicked, the User Exit Definition Dialog window is displayed, as shown in Figure 24-22, allowing you to provide information for the location and invocation of the Wrapper exit.

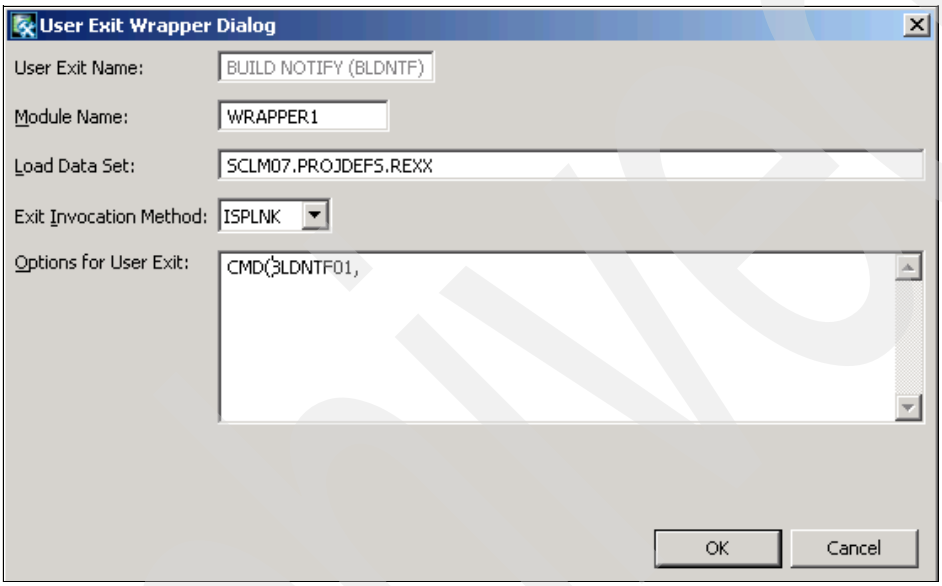


Figure 24-22 The Exit Wrapper parameter window is similar to the User Exit parameter window

Note: If the Exit Invocation Method is ISPLNK for any child user exits contained within an Exit Wrapper, then the Exit Wrapper itself must use ISPF as the invocation method so the connection to ISPF services is not lost.

Once the Exit Wrapper parameters have been specified, then the button **Wrapper Source** in the upper right-hand corner of the User Exit Definition Dialog window in Figure 24-21 on page 605 will allow you to generate, or regenerate, the source code for the Exit Wrapper.

The Exit Wrapper source code shown in Figure 24-23 was generated for our example and placed into the Module Name and Load Data Set specified on the User Exit Wrapper Dialog window.

Parameters can be edited from the User Exit Wrapper Source editor window. Any updates made in the editor will be saved in the PDS on the z/OS host.

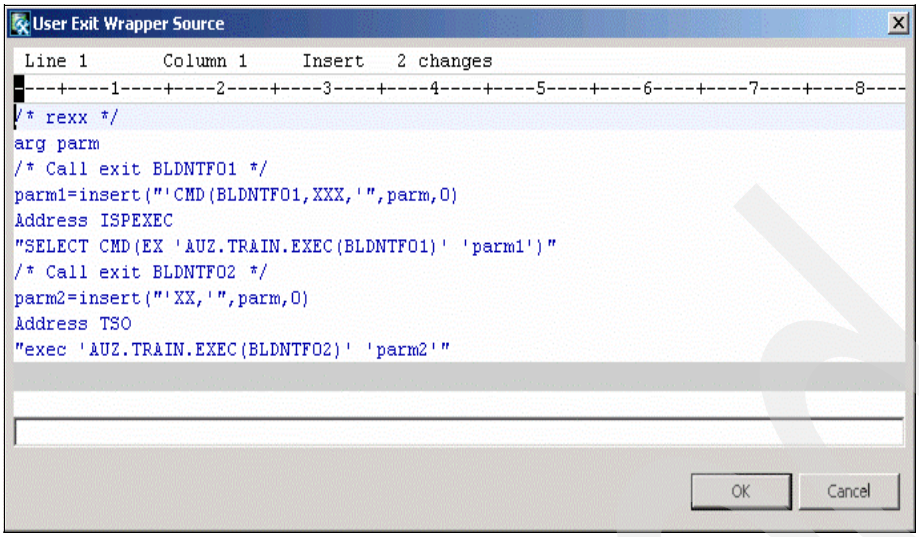


Figure 24-23 The source code of the Exit Wrapper

Once a wrapper has been created, the information on the User Exits tab will change to reflect which user exit in the list has been defined as an Exit Wrapper, as shown in Figure 24-24.

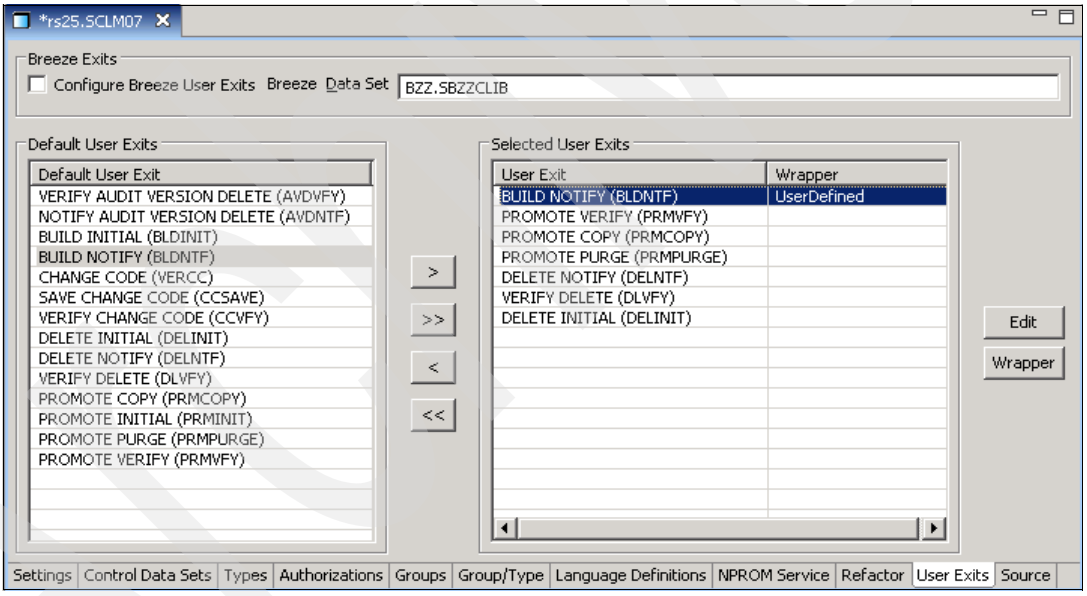


Figure 24-24 The column Wrapper reflects those exits designated as Exit Wrappers

24.2.9 Viewing the source tab

The last tab in the Project Editor is the Source Tab. This tab enables you to view the Project Definition Markup Language (PDML) created by the Admin Toolkit to monitor changes to the project's parameters.

In the workstation-based client, the PDML is viewable in XML language format as shown in Figure 24-25.

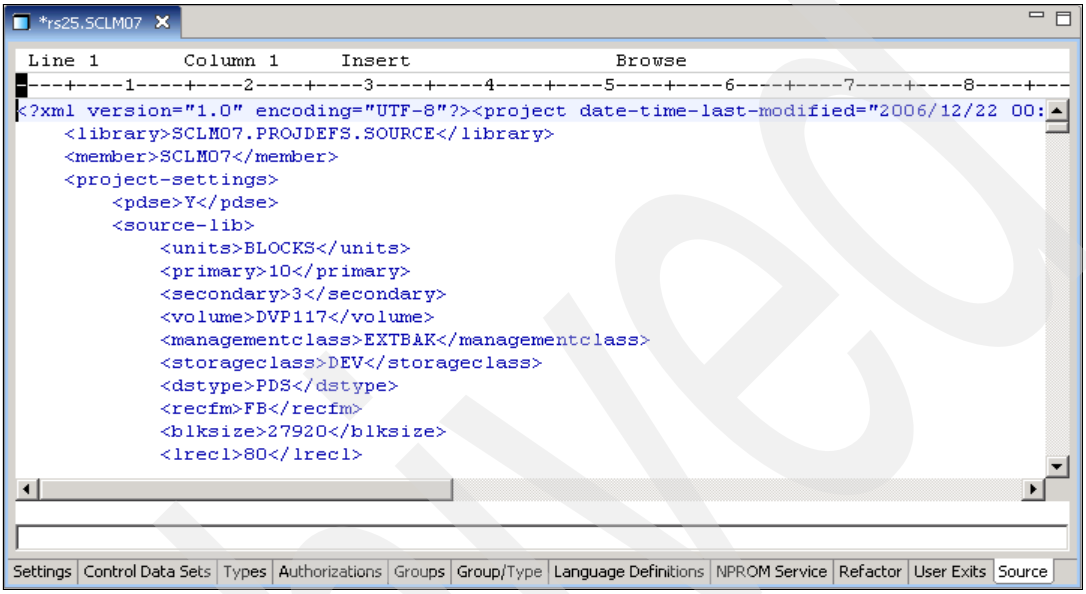


Figure 24-25 Project PDML in XML format

However, when viewing the project's source on the z/OS host, the PDML is in UTF-8 format and is not viewable on the host.

WARNING: The project source PDML is not meant to be edited and should not be modified by the user at any time. Unpredictable results may occur and updates to your project's definition may be lost.

When working with IBM support to diagnose any issue with the SCLM Admin Toolkit, the project's PDML is commonly asked for. The member out of <project>.PROJDEFS.SETTINGS may be FTP'd to your PC and emailed to IBM support.

Archive

24.3 The Language Definition Wizard

Language Definitions are translators that tell SCLM how it is to process the application's source code when builds occur. Language Definitions are an important part of a project, and because of their complexity, they can be difficult to create.

To eliminate the difficulty of creating complex Language Definitions, the SCLM Admin Toolkit has a Language Definition Wizard that contains content-sensitive fields, so that conflicting parameters cannot be entered. With the assistance of the Wizard, creating a language definition is easier than ever before.

Figure 24-26 displays a list of language definitions defined to the sample project's source. Each language definition must have a unique name but does not have to reside in the project's '<project>.PROJDEFS.SOURCE' data set.

For example, language definition ARCHDEF is being called directly from the SCLM macro data set ISP.SISPMACS. Because the SCLM macro data set is not writable, then no edits were made to the ARCHDEF language. If any edits must be made to the ARCHDEF language, then the Admin Toolkit will make a copy of the ARCHDEF language from the SCLM macro data set and save the edited copy in the project's '<project>.PROJDEFS.SOURCE' data set.

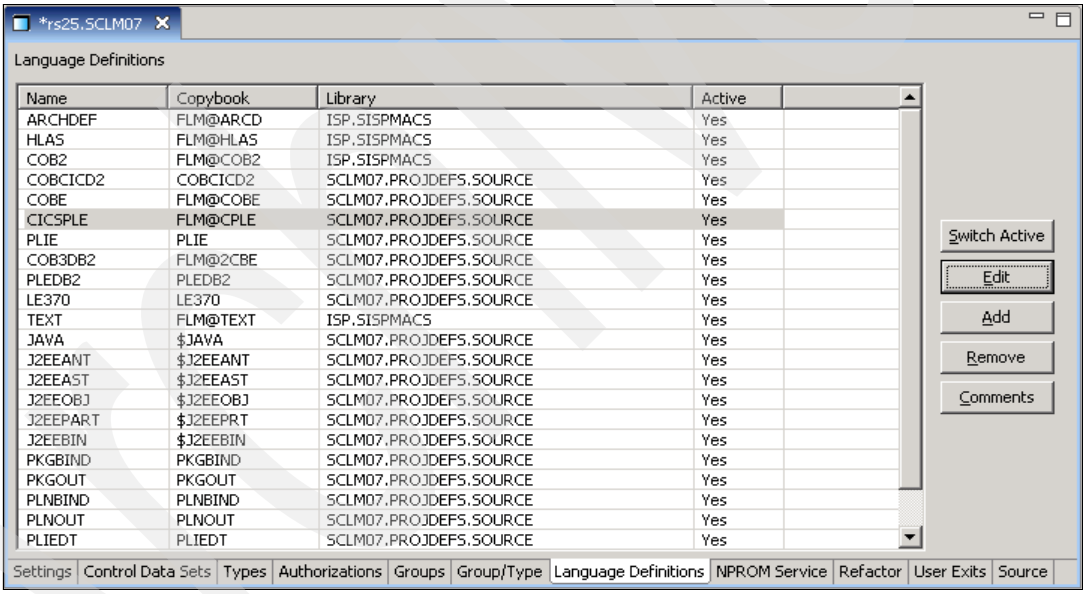


Figure 24-26 The list of languages defined to a project

To create a new language definition, click **Add** to the right-hand side of the tab. To edit an existing language definition, click the **Edit** button. The Language Definition Wizard window opens when either the **Add** or **Edit** buttons are clicked.

All parameters are editable for existing languages as well as new languages, with the exception of the fields Language and Copy Book, as shown in Figure 24-27 on page 611. Those two fields are not editable if you are editing an existing language definition. They are editable, however, if you are adding a new language to a project.

In this Language Definition window, you can provide information for the language identifiers and creating rebuild groups. On this panel you can specify:

- A **Description** for this language definition. It is for your information only and must be enclosed in check marks.
- The **Version** for this language definition. If you change this value, all source members using this language will be rebuilt.
- A **Buffer Size** value specifying the memory required to hold the expected number of $list_info records that SCLM allocates for a parse, verify, build, copy or purge translator. The translator returns dependency information in the allocated memory.
- A **Member Limit** that specifies the maximum number of source members that can be present in any input list presented to the translator. If you specify 0, then there is no limit on the number of source members.
- A **Default CREF** name that identifies the Type that is substituted into the @@FLMCRF variable for include set definitions. This variable can be used in the list of Types to be searched for Includes.
- A **Default Source** name that is a type name. It is used to allocate a hierarchical view that is typically used by the translator to resolve references to SCLM hierarchy members. This parameter is ignored if the FLMALLOC macro with an IOTYPE=I and KEYREF=SREF is not used in the language. This parameter is ignored during a build if a CC, Generic or LEC architecture definition is used to build the source member.
- The **Check SYSLIB** parameter specifies whether or not the System Library is to be checked to determine whether an include is to be tracked, and if so, when. Checking may occur at either parse or build time, or not at all.
- Whether **COMPOOL Output** is required.
- The **Allocate Syslib** option to allocate system library data sets for IOTYPE=I.
- The **Archdef Member** option if the member passed in this language is an architecture member.
- The minimum allowable **scope**. SCLM compares this parameter with the mode specified as input to build and promote functions to allow or disallow processing. The input mode must be of equal or greater value than the language scope.
- If this language can be assigned to **editable members**.
- If you want to **Rebuild Dependents** of this member when some outputs from the translator for this member were not saved.

The FLMRBLD macro at the bottom of the window displays the Rebuild Groups that causes members with a particular language to be rebuilt when they are promoted into particular groups.

Archive

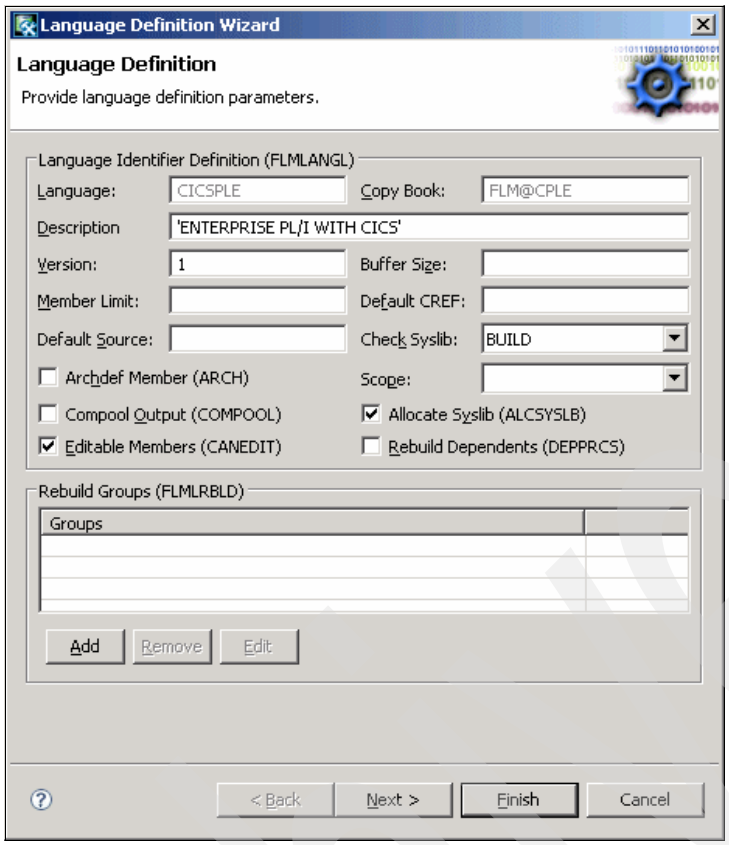


Figure 24-27 The first Language Definition Wizard panel

Provide the FLMTRNSL macro parameters on the Language Definition window, then click **Next**. You will proceed to the Syslib and Include Parameters window as shown in Figure 24-28.

In this Syslib and Include Parameters window, you provide parameters for the System Libraries and Include Sets. In this window you can click the **Add** buttons to specify:

- The FLMINCLS macro:
  - The **data set Name** and a **data set Type** that contains the includes for the Include Set. Two SCLM variables, @@FLMTYP and @@FLMETP, can be used in the Types list.
  - A **Crosslang** value of Y or an N to indicate to SCLM if it is to process the Includes of an included member when the Include member has a different language from the source member, that is, whether or not language boundaries are crossed.
- The FLMSYSLB macro:
  - The **Data Set Name** that contains the macros or includes from outside the project. The data set name must meet MVS data set naming standards.
  - The name of an **Include** set on an FLMINCLS macro (from the list at the top of the window).
  - The **Volume serial number** of a direct access on which the data set is located.

You can add comments to each of the items in a macro list by clicking the **Comments** button. These comments can provide additional information for your use. You can also remove any item from either of the lists by highlighting an item in the lists and clicking the **Remove** button.

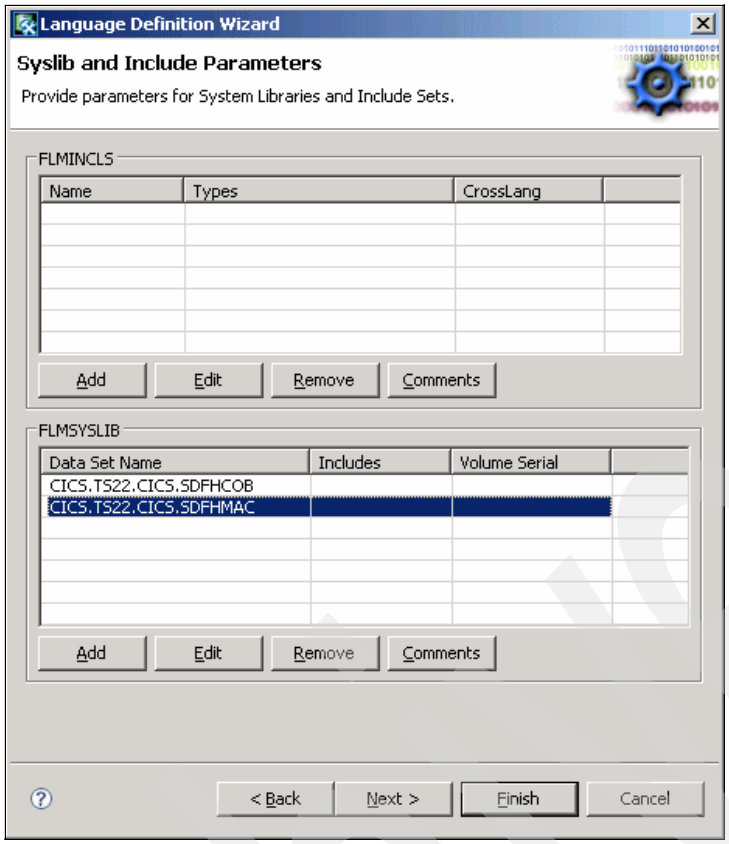


Figure 24-28 The SYSLIB and Include Set window

A Translator Summary, shown in Figure 24-29, is provided in the Language Definition Wizard to allow you to choose a translator to edit. You can edit the translators in any order, or you can choose to click **Finish** to exit the Language Definition Wizard if no edits to the language translators are needed. Select the translator you want to work with and click the **Edit** button.

Archived

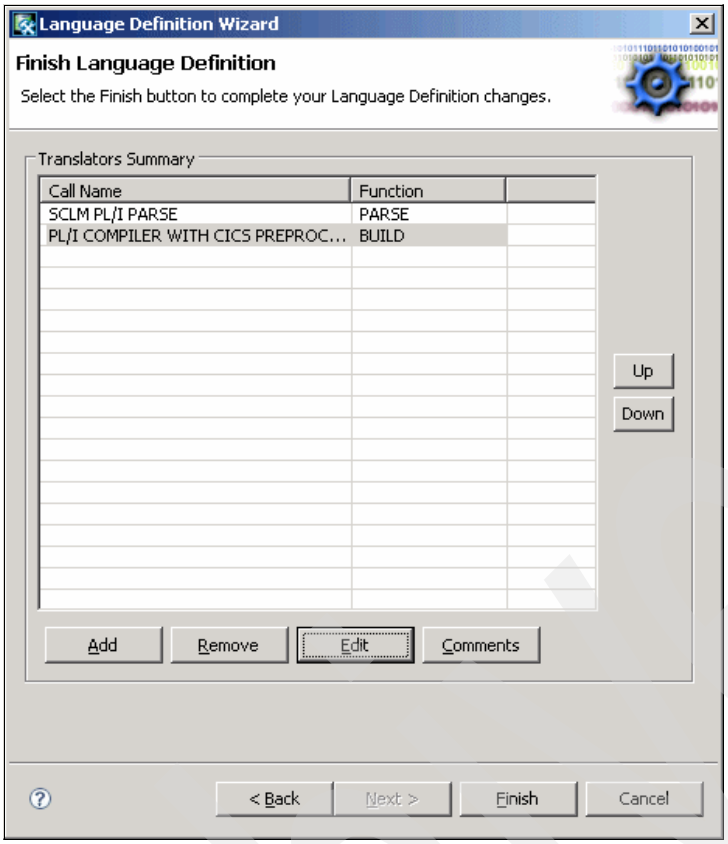


Figure 24-29 Translator Summary window

The Translator Parameters window, Figure 24-30 on page 615, is displayed to allow you to edit or add translator parameters. The Wizard assists you in defining the translator parameter options and data sets that will be used by SCLM. You can specify the following parameters:

- The **function** that this translator will provide:
  - The Parse function gathers statistics and dependent objects that SCLM uses when building or promoting artifacts.
  - The Verify function performs a validation.
  - The Build function assembles, compiles, links or otherwise processes a source member.
  - The Copy function copies an SCLM-controlled artifact to the next higher group in the project's hierarchy.
  - The Purge function purges data that is related to an SCLM-controlled artifact, but the data is not under SCLM control.
- The **Call Name** to name this translator.
- The **Call Method** that will be used to call this translator.
- The field **Compile** is the name of the program that will be invoked when this translator is used. This program can be a parser, a compiler, an assembler or some other user-supplied program. If you selected a call method of ATTACH, LINK or TSOLNK, then the field Compile should be the name of a REXX Exec, CLIST or a load module. If you selected a call method of ISPLNK, then this value must be SELECT to invoke ISPF services.
- The **Step Label** is a name that identifies the job step for this translator.

- The field **Good RC** is the number of what you deem is an acceptable return code for this translator step to complete with.
- **PORDER** is an integer from 0 to 3 that indicates the parameter order to pass to the translator. The following values are allowed:
  - 0 - no parameters are to be passed.
  - 1 - Only the Option List is to be passed.
  - 2 - Only the DDname substitution list is to be passed.
  - 3 - The Option List is to be passed first, then the DDname substitution list.
- The **Task Library DD** name that contains the translator load module. Use this only when you've specified ATTACH as the call method.
- The **No Save External** is a return code value that will indicate whether any translator outputs targeted to an external data set were saved. The BUILD processor determines that the external outputs were not saved by the translator if this value is equal to a return code other than zero.
- The **Max Good RC** value is used in conjunction with the INPLIST parameter to indicate the maximum value of a good return code for each member of the input list.
- Specify a **Version** for the translator. This parameters stored in the account record for each output member saved from the translators. If you do not specify this parameter, SCLM Administrator Toolkit sets it to blank.
- The **Override Options** field indicates that developers can override the default translator options.
- The **Input List Processing** field indicates that this translator supports input list processing.
- The **PDS Data** field indicates that the input members for this translator reside in SCLM-controlled partitioned data sets. If you specify multiple translators for one language definition, then this value must be the same for all translators.
- The **Param Keyword** is used in architecture members to specify additional options for this translator.
- The field **Data Set Name** is the data set containing the translator load module (COMPILE parameter) being invoked. This parameter is not required if the translator load module resides in the system concatenation.
- The field **Options** is a list of options that are passed to the program whose name you specified in the COMPILE parameter. The maximum length of the list, including delimiters, is 255 characters. The options you pass must be acceptable to the program. Delimit the list with single quotes or parentheses. The options can also contain variables to provide dynamic information to the COMPILE program. For example, the variables @@FLMMBR and @@FLMTYPE will be replaced with the name of the member and type of the last SINC, INCL, or INCLD statement in the architecture definition. So if a source member is being built, the variable will be replaced with the name of the source member.

The FLMALLOC macro allocates any data sets that are needed by your translator to process the project's artifacts associated with the language you are editing. The IO Type of data set to be allocated is designated along with the DD name and its corresponding Key Ref value. More is explained on the FLMALLOC macro below.

The FLMTCOND and FLMTOPTS macros can be accessed by clicking the appropriate button.

Archived

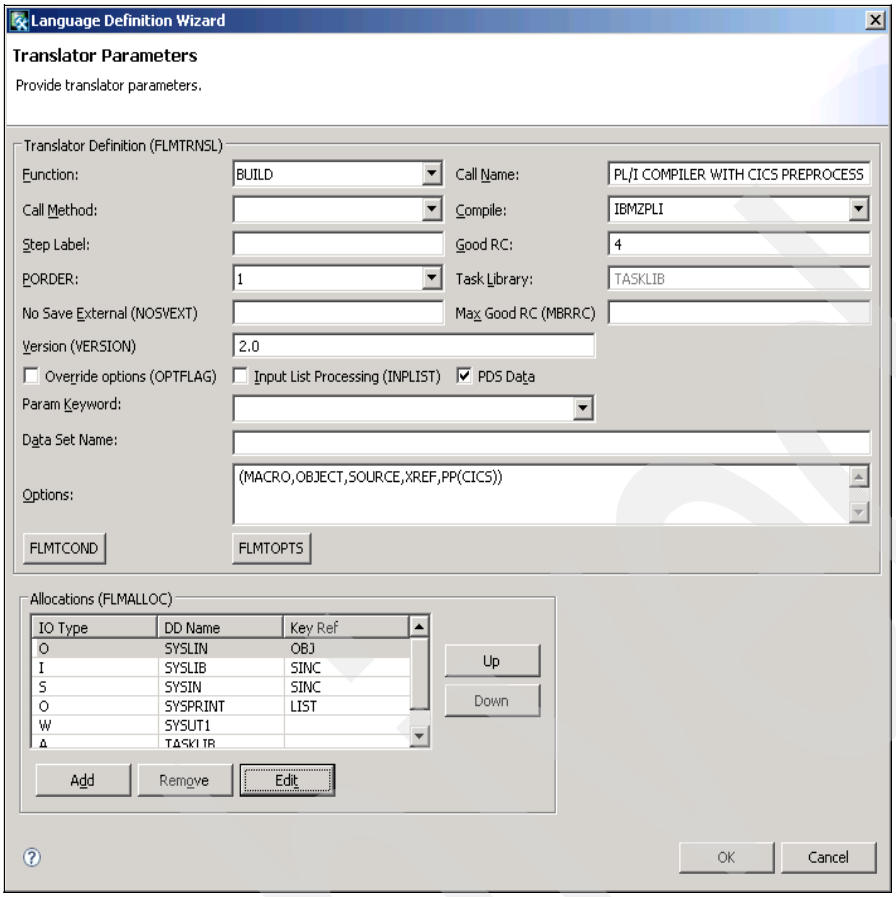


Figure 24-30 The Translator Parameter window

When you click the **FLMTCOND** button, the Condition Dialog window is displayed, as shown in Figure 24-31. The FLMTCOND macro dictates if a translator is to be run or skipped based upon the group for which the build takes place and return codes issued from previous translators in the same language definition.

You can specify the following parameters in this window:

- The **Group Criteria** field indicates the groups to which the conditional criteria are to be applied or excluded, depending on the radio button selected.
- The name of the **Translator** step to which the condition will be applied.
- The **Relation** field is the operator for the return code.
- The **Return Code** is value of the translator's return code that must be matched to make the condition true.
- The **Action** that is to take place when the condition is met; either run the translator or skip the translator.

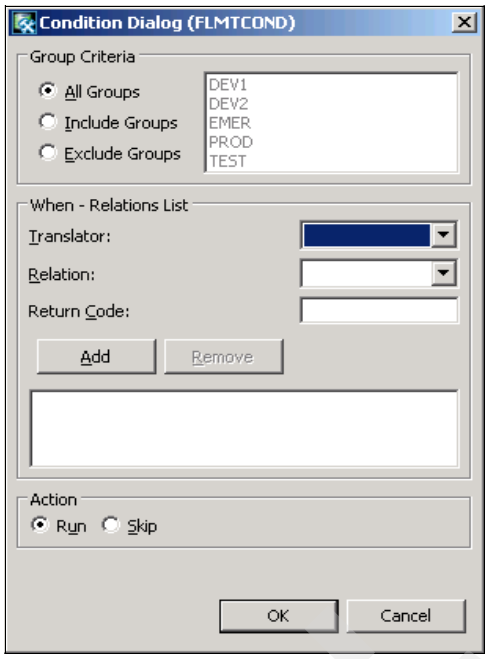


Figure 24-31 The FLMTCOND parameters window

The FLMTOPTS macro is applicable to BUILD translators only and allows you to specify one or more sets of options to use based on groups. You must specify the following on the Options Dialog window as shown in Figure 24-32:

- The **Group Criteria** field indicates which groups are to have the specified options applied to, or excluded from depending on the radio button selected.
- The **Option Sets** to be passed to the translator.
- The **Action** to take for the listed sets of option; either replace the existing options list in the Translator Parameter Window or append the set of options to the options list.

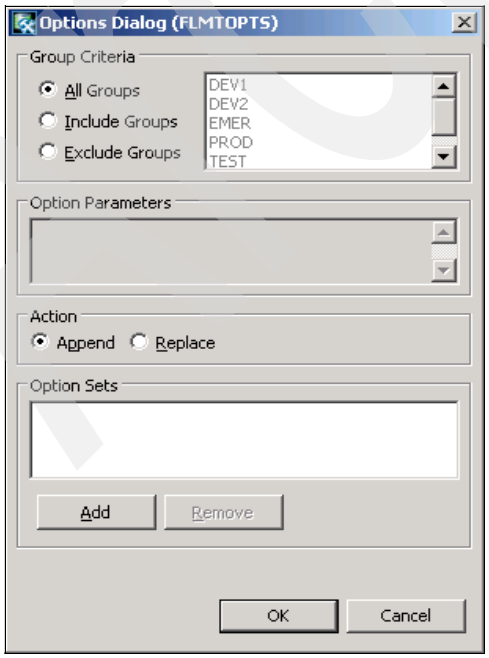


Figure 24-32 the FLMTOPTS parameter window

Archived

The text box Options Parameters displays the parameters of the currently selected options set, and is not editable. When you click **Add** in the Options Dialog window, then a separate pop-up window appears where you can enter option parameters. When you click **OK**, then you will see an Options Set included in the field, Options Set. When you highlight an Options Set, you will see the complete text listed in the Options Parameters text box.

Back on the Translator Parameter window, notice the conflict message that appears in the top of the window when a parameter's option is set to something that conflicts with another parameter's option, as shown in Figure 24-33. In the example below, the option PDS Data was de-selected, causing a problem with the translator. When this type of conflict occurs, you cannot click **OK** to move forward until the parameter options have been rectified.

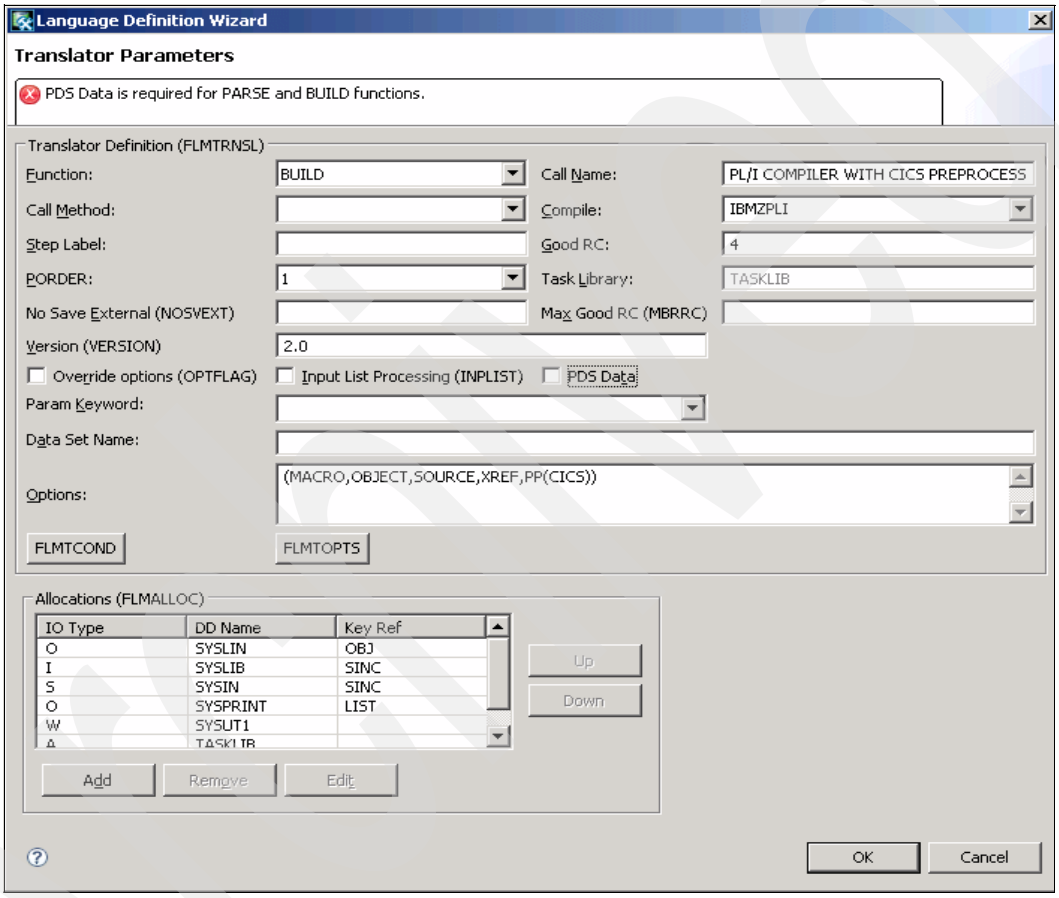


Figure 24-33 You cannot click OK when conflicting parameter options have been entered

The Translator Parameters window is also where the FLMALLOC macro values are specified. The FLMALLOC macro allocates any data sets that are needed by your translator to process the project's artifacts associated with the language you are editing. Using the buttons, **Add**, **Remove**, and **Edit**, you can add data sets to the FLMALLOC macro, remove existing data sets from the translator or edit existing ones.

When the **Add** or **Edit** button is clicked, the Data Set Allocation Dialog window is displayed. Figure 24-34 illustrates how the Wizard ensures users do not enter parameters that are not applicable, as fields in the Wizard windows will be grayed out if they do not apply to the IO Type of the data set being added or edited.

Using Figure 24-34 as an example, the top portions of the Allocations Dialog permits you to specify allocation parameters for the translator.

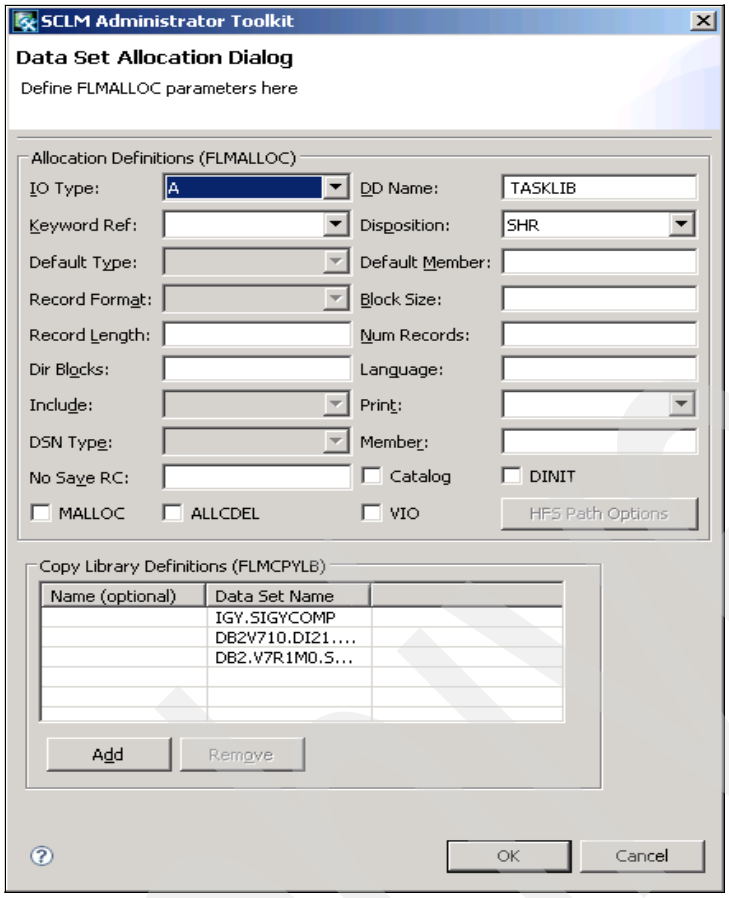


Figure 24-34 Valid fields for IO Type O

You can specify the following parameters on the Data Set Allocation Dialog window:

- The **IO Type** field identifies what kind of data set the translator is to use.
- The **DDname** field tells the translator the name to be used for this allocation.
- The **Keyword Reference** is a reference to a keyword in a build map or architecture definition that is used by other parameters in this macro. Its value differs depending on the specified IO Type:
  - IOTYPE=L - the member name of the macro passing the DDname substitution list for the translator
  - IOTYPE=S - the input members for the translator
  - IOTYPE=I - determines the type name of the hierarchy to allocate
  - IOTYPE=O or P - identifies the location in the hierarchy for the build function to copy the output created by the translator of the translator is successful.
- The **Disposition** of the data set to be used for the allocation in a JCL DD statement.
- The SCLM **Default Type** data set for this allocation
- The **Default Member** name for the output member.
- The **Record Format** that the data set is to be allocated with.

- The **Block Size** that the data set is to be allocated with. This parameter is only valid with IOTYPEs W, O, P, and S.
- The **Record Length** is the logical record length that the data set is to be allocated with. This parameter is only valid with IOTYPEs W, O, P, and S.
- **Num Records** is the number of records this data set is expected to have. This helps to size the data set appropriately. This parameter is only valid with IOTYPEs W, O, P and S.
- The **Dir Blocks** field specifies how many directory blocks to allocate the data set with.
- The **Language** field is used when you want the build output of one language definition to be verified, built, copied, or purged in another language definition.
- The **Include** field specifies the name of an FLMINCLS macro that lists the types to be allocated for this translator. This is only valid for IOTYPE=I.
- The **Print** parameter allows you to copy the contents of a sequential data set to the SCLM listings data set. This parameter is only valid with IOTYPEs W, S, or O.
- The **DSN Type** field value indicates whether to allocate a temporary partitioned data set as PDS or a PDSE. This parameter is only valid for IOTYPE P.
- The **Member** field is the name of a PDS member that contains the output from the translator step. This parameter is only valid for IOTYPE P.
- The **No Save RC** field specifies a return code value that will be set by a translator and, if matched, indicates that SCLM is not to store a translator's output in this data set. This parameter is valid for IOTYPE O and P.
- The **Catalog** field indicates that this data set is to be catalogued. If you check this box, SCLM temporarily allocates cataloged data sets with a predefined high-level qualifier, the TSO-prefix, and deletes the data set after all translators complete their functions.
- The field **DINIT** tells SCLM to create a default member name into this temporary data set. The member that SCLM creates is initialized with a single record containing the string DUMMY FILE, and will have the same name as the Build Map. However, if you specified a value in the Member field, then the member name inserted into this temporary data set is that value, and not a default name.
- The **MALLOC** field tells SCLM to generate a sequential output data set with a specific name. If you use this option, you must also define a data set name in an FLMCPYLB macro and use a KEYREF keyword. This parameter is only valid with IOTYPE O and A.
- The **ALLCDEL** parameter tells SCLM to delete all data sets associated with this translator upon completion.
- The **VIO** parameter tells SCLM to always use VIO when allocating this data set.

The FLMCPYLB macro in the Copy Library Definitions portion of the Data Set Allocation Dialog window allows you to allocate a data set with a specific DD name. You can add new data sets by clicking **Add**, and remove existing data sets by highlighting a data set in the list and clicking **Remove**.

When the **Add** button is clicked, the next empty Data Set Name field will become active, and your cursor will be positioned in the Name column. The Name field is the DD name of the data set to add to the FLMCPYLB macro and is optional. The Data Set Name field is required and can contain SCLM variables.

Once you have completed editing or adding any translators to the Translator Summary window, click **Finish** to return to the Language Definition tab in the workstation-based client.

24.4 The Clone Project Utility

There may be a time when the need arises to copy an existing project to test new edits and changes that you want to make. Alternatively, you can want to create an entirely new project but do not have time to define all of the project parameters that it takes to get a new project operational. In such cases, a utility called the Clone Project Utility allows you to clone an existing project's definition AND the underlying project's application source code in a fraction of the time it would take to create a new project manually, or even through the Project Editor in the Admin Toolkit.

Cloning a project eliminates the need to navigate through the Project Editor's tabs one at a time, specifying each parameter's options as you go. With one click, you can clone a complete working project in just minutes, and optionally choose to also clone the source code. With a time-saving utility, you get a new working project in minutes that you can then tweak for your new project's requirements, or edit the source code via SCLM to meet your changing business needs.

In the workstation-based client, open the project you want to clone. Select SCLM from the menu bar above, then click **Clone Project**, as shown in Figure 24-35.

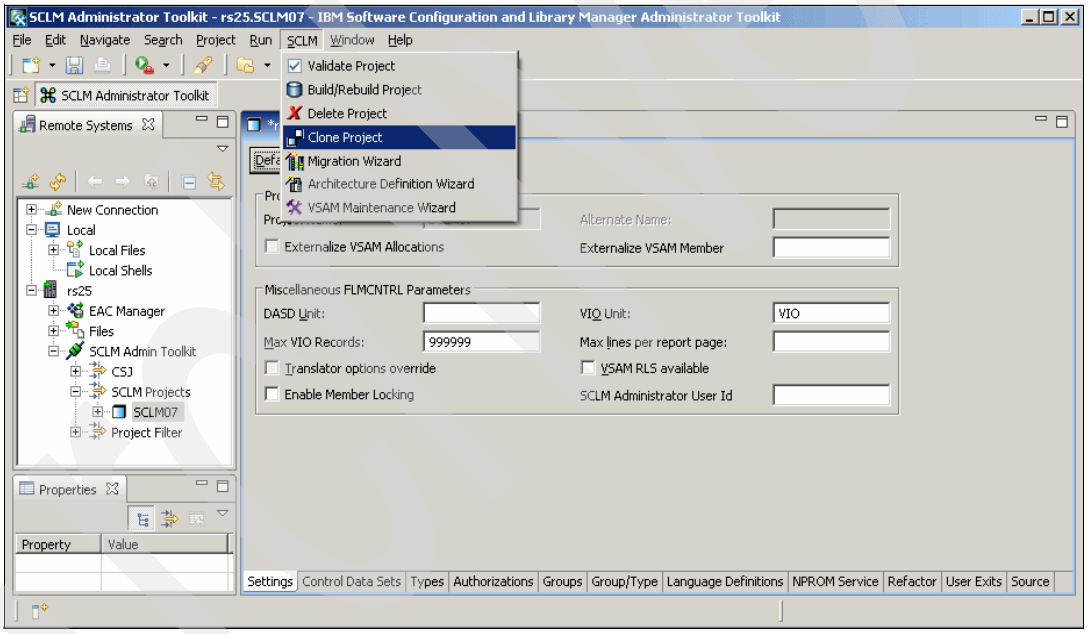


Figure 24-35 Select Clone Project from the SCLM pull-down menu

When the Clone Project Utility is selected from the SCLM pull-down menu, a pop-up window will allow you to specify the project name, and whether you want to clone the project's source code data as well, as shown in Figure 24-36.

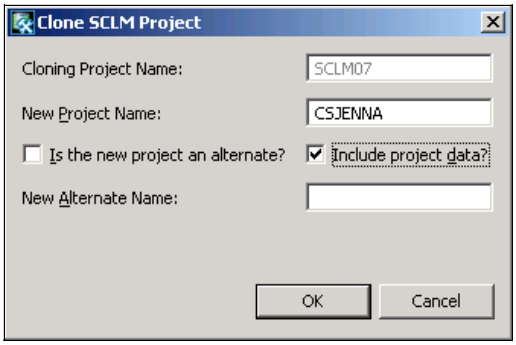


Figure 24-36 The Clone Project options

The field, Cloning Project Name, is the name of the project you are making a copy of. In our example, we will copy project SCLM07.

The field, New Project Name, is the name of the project to be created. In our example, we will create project CSJENNA and copy everything from SCLM07 into CSJENNA.

The field, Is the new project an alternate?, is asking if the new project, CSJENNA, is an alternate of the base project, SCLM07.

The field, Include project data?, is asking if you want to copy the application source data for project SCLM07 into project CSJENNA. By selecting this option, you are permitting the Admin Toolkit to copy all group/Type data sets and its source code from SCLM07 to CSJENNA so that you have an exact copy of the application source code being developed in project SCLM07's to project CSJENNA. This allows you to begin development effort on a new application using the source code that is already available. If the option is not selected, then only the project definition data sets (<project>.PROJDEFS.*) from project SCLM07 are copied.

The field, New Alternate Name, is only applicable if the option, Is the new project an alternate?, is selected. Otherwise, it is ignored.

An alternate project is one that uses a different project definition from the base project. It is basically an alternate view of the base project hierarchy. An alternate project definition is used most commonly to migrate artifacts into a project. Because SCLM only allows artifacts to be migrated into the lowest level of a project's hierarchy, an alternate project allows you to define the hierarchy without the development groups so that artifacts can be migrated into the test level, the maintenance level or the production level.

For example, in the next section concerning the Migration Utility, we will cover in-depth how to clone a project and migrate assets from other sources into our new project. You will gain a better understanding of when an alternate project is used and how it can benefit your development effort. For now, the Clone Project Utility will be presented to create a new completed project.

Once you have clicked **OK**, the SCLM Admin Toolkit processes your request by copying all SCLM07 project data sets and application source code data sets, and creates data sets beginning with CSJENNA to complete the new project.

Figure 24-37 shows the new project CSJENNA in the Remote Systems View frame and opens the project for you to review in the Project Editor frame. Notice that SCLM07, the project that was cloned, is still open. When you are done using project SCLM07, you can close out of the project in the Project Editor frame and make any necessary changes to the new project CSJENNA.

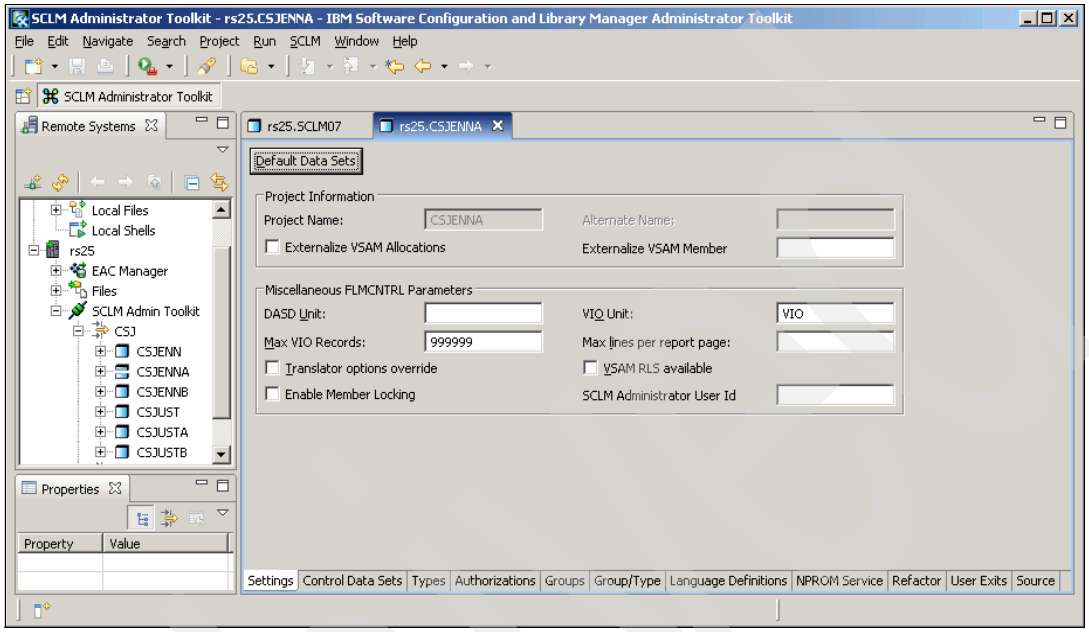


Figure 24-37 The new project appears in the filter list as well as the Project Editor frame

The ISPF Data Set List Utility in Figure 24-38 and Figure 24-39 displays all the CSJENNA.PROJDEFS.* data sets as well as the underlying application source code data sets that were created when project SCLM07 was cloned to create new project CSJENNA.

```
  Menu  Options  View  Utilities  Compilers  Help
DSLIST - Data Sets Matching CSJENNA.*                  CHARS 'PROJDEFS' found
Command ===> _____________________________________________ Scroll ===> CSR

Command - Enter "/" to select action              Message          Volume
-------------------------------------------------------------------------
        CSJENNA.PROJDEFS.AUZLIST                                   TSP105
        CSJENNA.PROJDEFS.AUZOBJ                                    TSP113
        CSJENNA.PROJDEFS.CLIST                                     TSP105
        CSJENNA.PROJDEFS.JCL                                       TSP106
        CSJENNA.PROJDEFS.LIST                                      TSP102
        CSJENNA.PROJDEFS.LMAP                                      TSP10B
        CSJENNA.PROJDEFS.OBJ                                       TSP112
        CSJENNA.PROJDEFS.PROC                                      TSP107
        CSJENNA.PROJDEFS.REXX                                      MIGRAT1
        CSJENNA.PROJDEFS.SETTINGS                                  TSP10D
        CSJENNA.PROJDEFS.SOURCE.BACKUP                             TSP110
        CSJENNA.PROJDEFS.WORK                                      TSP105
        CSJENNA.PROMOTE.MSGS00                                     MIGRAT2
        CSJENNA.PROMOTE.REPORT00                                   MIGRAT2
        CSJENNA.RSQA.AUZ310.IBMTAPE.ISPTLIB                        TSP106
        CSJENNA.RS22.ISPPROF                                       TSP102
        CSJENNA.RS25.ISPPROF                                       TSP103
```

Figure 24-38 Project definition data sets were cloned from project SCLM07

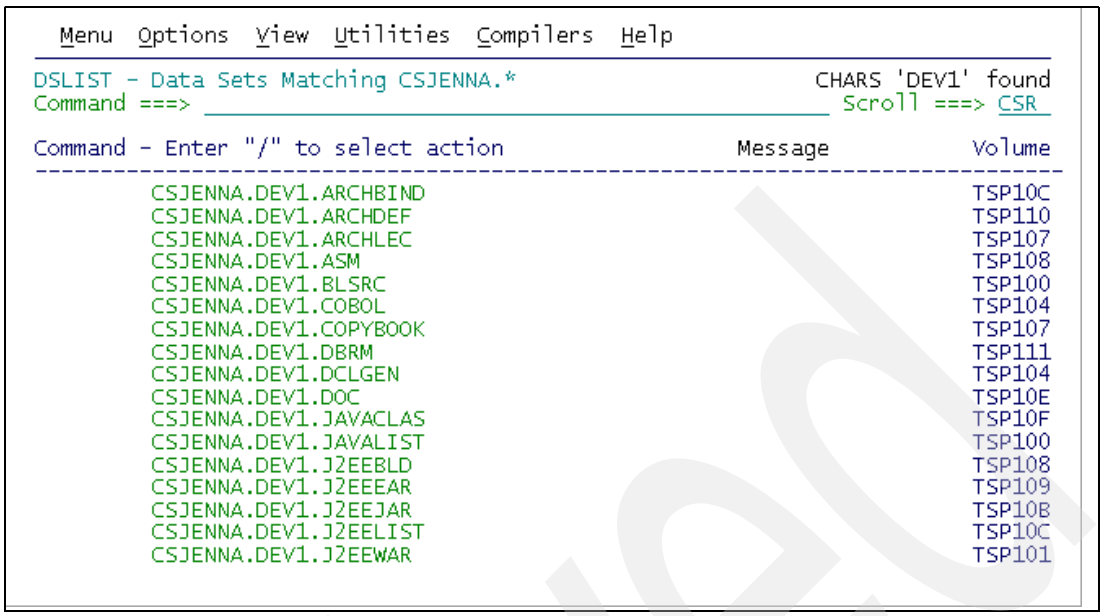


Figure 24-39 Application data sets were cloned from project SCLM07.

24.5 The Migration and Remote Migration Wizard

Migrating artifacts from other host data sets or from remote systems into an SCLM-managed project can be done using the Migration Wizard and the Remote Migration Wizard.

The Migration Wizard allows you to migrate multiple artifacts of the same type into your project, and automatically updates all accounting information and optionally auditing information, if versioning is being used in your project. The Admin Toolkit allows you to migrate both host-based artifacts and workstation-based artifacts into your host-based project. However, the Remote Migration Wizard is only available from the workstation-based user interface.

24.5.1 Migration Wizard for host-based artifacts

To begin using the Migration Wizard to migrate artifacts into your SCLM-managed project from the workstation-based user interface, use the instructions in this section.

Open the project you want to migrate artifacts into. Once the project is open in the Project Editor frame, select SCLM from the menu bar above, then click **Migration Wizard** as shown in Figure 24-40.

Archived

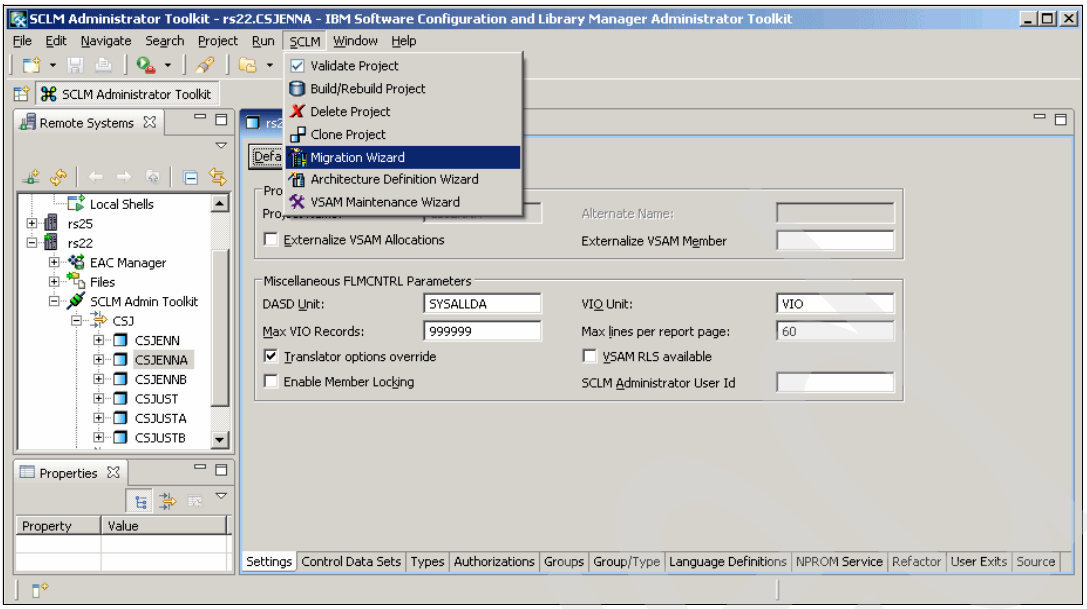


Figure 24-40 Select Migration Wizard for migrating host-based artifacts

The opening window of the Migration Wizard is the Migration Assets window, shown in Figure 24-41 on page 624, which lists the data set and member names of the artifacts which are to be migrated into your project. The button, **Add**, on the right-hand side of the window allows you to add data set and member names, while the button, **Remove**, simply deletes a highlighted artifact from the window, eliminating it from being migrated.

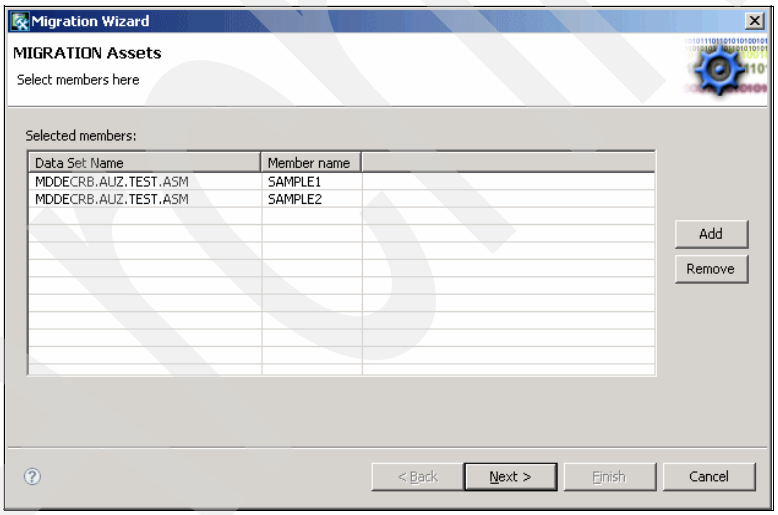


Figure 24-41 The initial Migration Wizard window lists artifacts to migrate

Click **Add** to bring up the Data set Selection window as shown in Figure 24-42. You can add multiple members from the same data set by highlighting them. Hold down the Control key as you highlight the members you want to add to the migration. Multiple data sets may also be included.

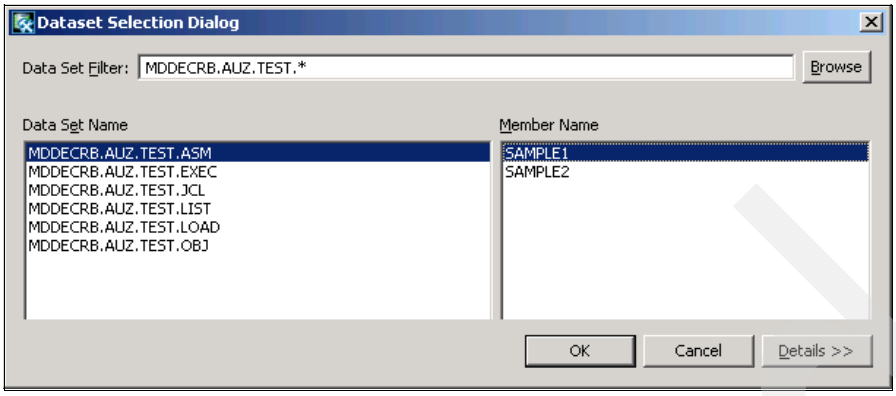


Figure 24-42 Data set Selection Dialog window allows you to select data sets and their members

Specify the following information on the Migrate Asset Options window, shown in Figure 24-43 on page 625:

- The **Group** to migrate artifacts into
- The data set **type** the members should be migrated into
- The **language** that SCLM is to use for the new artifacts being migrated into the project
- The **authorization code** that is to be used for the artifacts
- Optionally, the **change code** to use with the artifacts
- The **Mode** that is to be used to migrate the artifacts
- The **date and time** that is to be used to timestamp the artifacts.

Note: The parameters supplied in this window will be used for all selected artifacts being migrated into the project. If, for example, you must use a different language for an artifact in the list, then that artifact should be migrated separately.

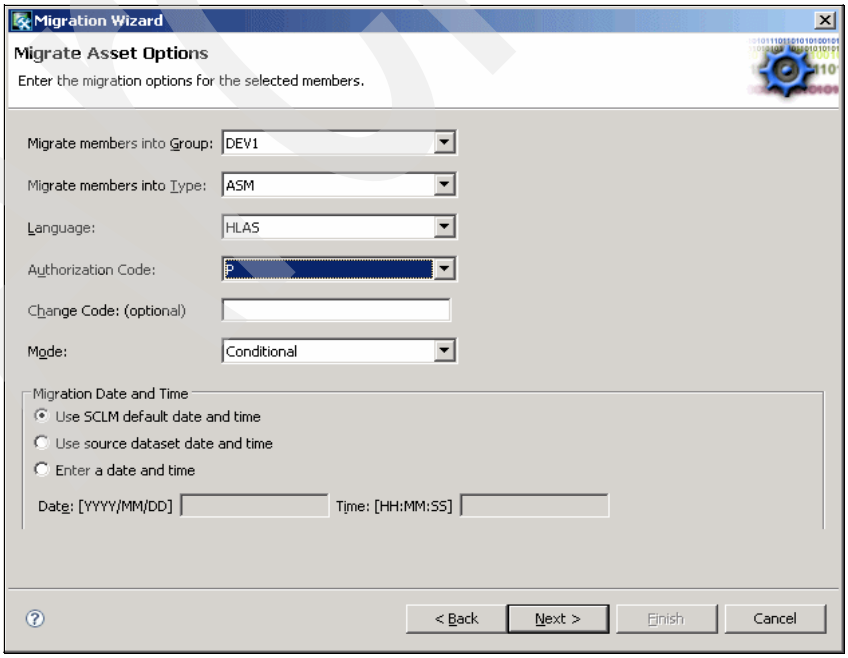


Figure 24-43 Specify the Migration parameters that are to be used for all artifacts

When all options have been set for the selected artifacts, click **Next** to begin the migration. When the migration has completed, you will receive a migration report as shown in Figure 24-44.

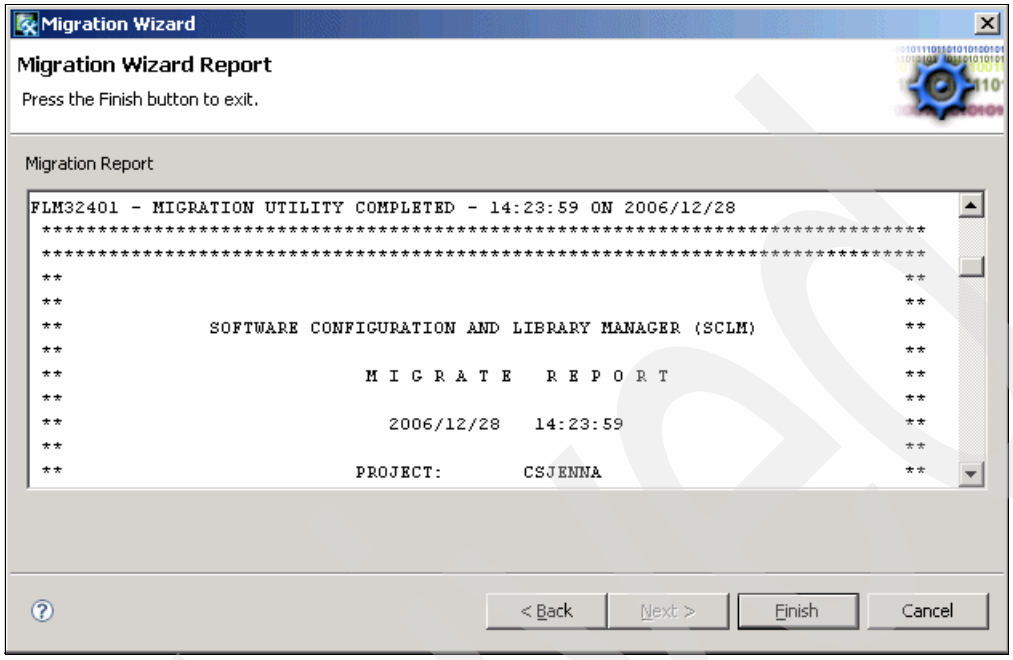


Figure 24-44 The migration report is returned for your review

The Migration Report window may be expanded to view more of the migration report. The scroll bars may also be used to view the complete report.

24.5.2 Remote Migration Wizard for workstation-based artifacts

To begin using the Remote Migration Wizard to migrate remote artifacts into your SCLM-managed project from the workstation-based user interface, use the instructions in this section. The Remote Migration Wizard is only available in the workstation-based user interface.

Open the project you want to migrate artifacts into. Once the project is open in the Project Editor frame, select Window from the menu bar above, then click **Open Perspective**. Click **Other** to view more perspectives to open as shown in Figure 24-45.

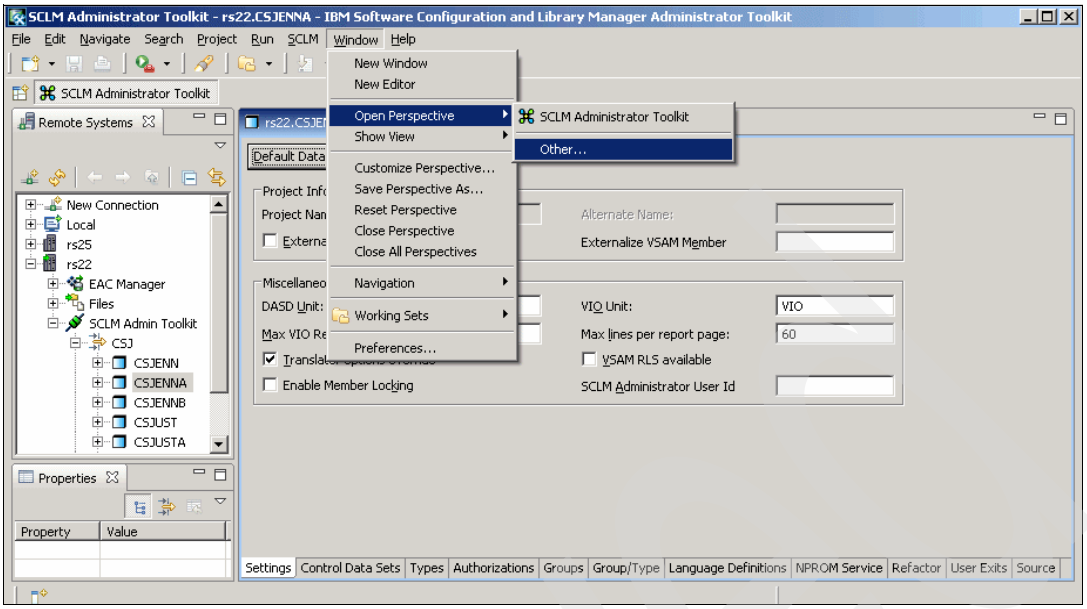


Figure 24-45 Navigate to the Resource perspective to access your remote assets

In this example, artifacts are being migrated from another Java project. Select the **Resource** perspective and click **OK** as shown in Figure 24-46.

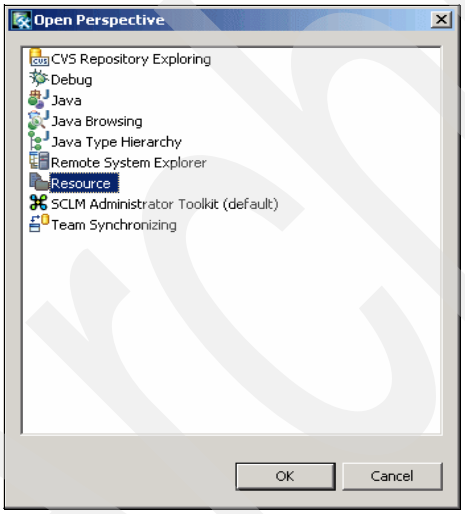


Figure 24-46 Select the perspective that will allow you to access your remote assets

The Resource perspective will open in place of the Remote Systems View perspective. Select the Java project you want to migrate artifacts from to expand it. Find the artifacts you want to migrate by holding the Control key and clicking the artifacts. Right-click the mouse button to see the additional menu appear. Then select **Team**. When Team is selected, the option Migrate to SCLM will be visible to select as shown in Figure 24-47.

Archive

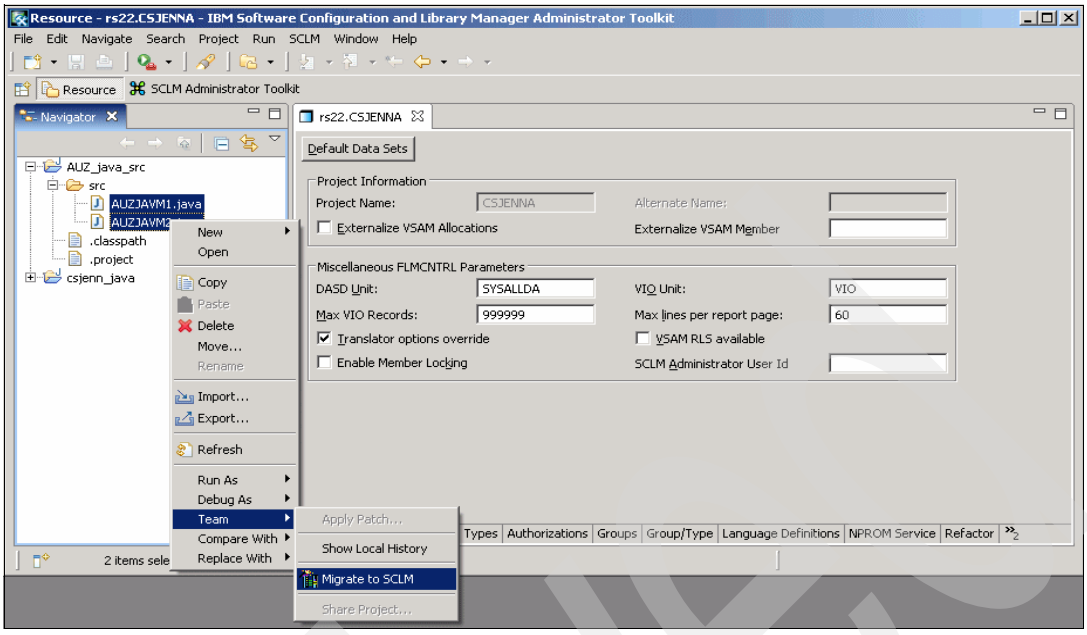


Figure 24-47 Highlight the artifacts to migrate, and right-click to get to the Migrate function

The SCLM Remote Migration Wizard window opens allowing you to select as many artifacts as you want to migrate. When you have finished, click **Next** as shown in Figure 24-48.

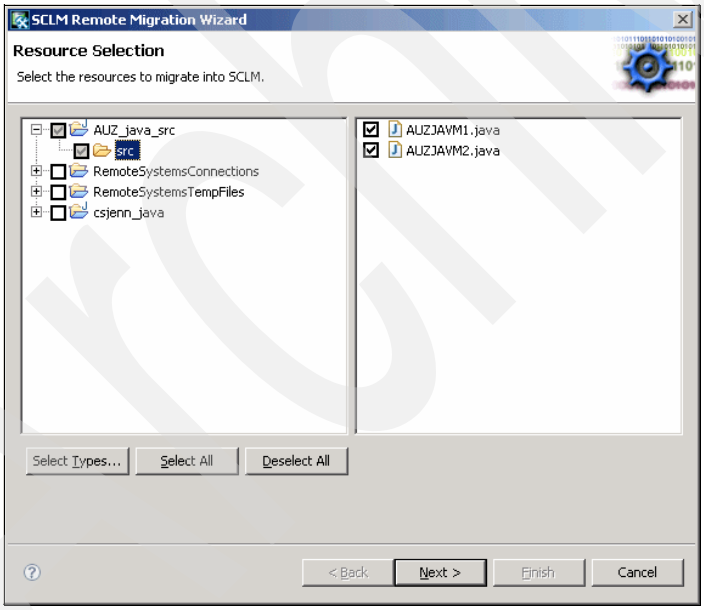


Figure 24-48 Select the remote artifacts to migrate

The window Migration Project opens asking you to select the host system you want to migrate your artifacts to. In addition, select the project name to migrate the artifacts into. Click **Next**. If you are not already logged into the host system in the Administrator Toolkit, you will be asked to log in as shown in Figure 24-49.

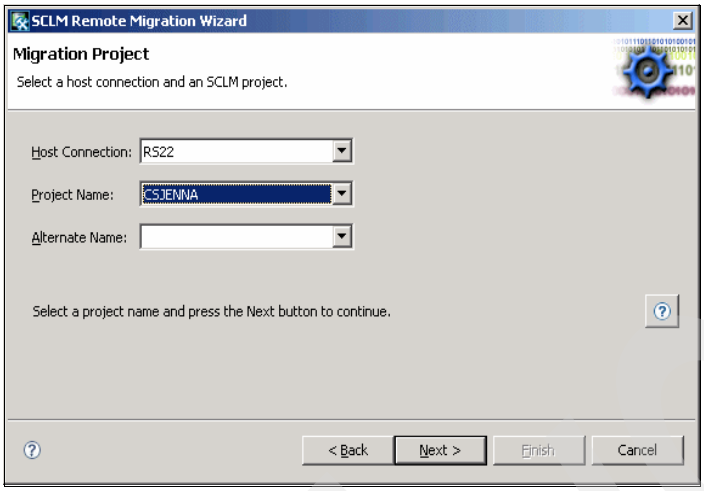


Figure 24-49 Log into the host system you want to migrate the artifacts into

The Migration Options window opens, as shown in Figure 24-50, and appears similar to the Migration Options window for migrating non-remote artifacts as seen in Figure 24-25 on page 608. Provide the following information:

- The **Development Group** to migrate artifacts into
- The **authorization code** that is to be used for the artifacts
- Optionally, the **change code** to use with the artifacts
- The **Mode** that is to be used to migrate the artifacts
- The **date and time** that is to be used to timestamp the artifacts

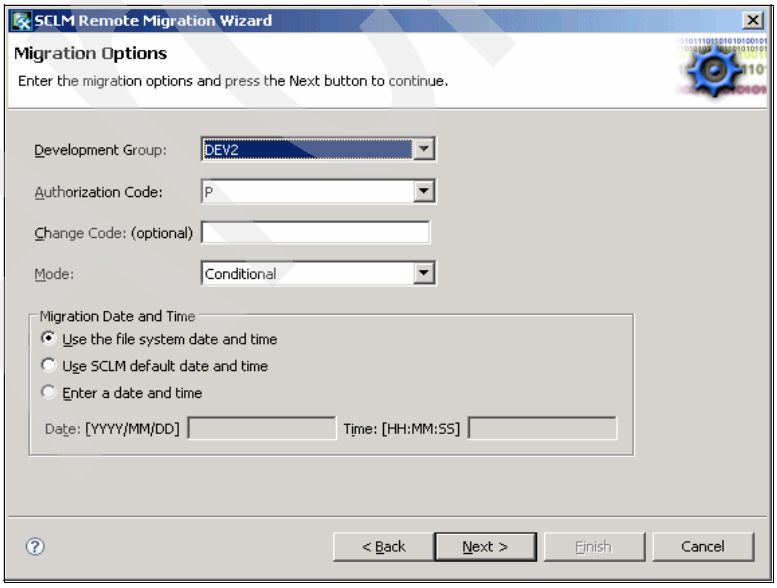


Figure 24-50 Specify the options to migrate the artifacts with

Click **Next** when you have provided all necessary parameters. The window Migration File Extension Mapping opens and will display all types of file extensions of those artifacts being migrated in. For example, if you chose to migrate a *.class file type and a *.html file type, they would be listed underneath the file type Java under the column File Extension.

Specify the SCLM Type of data set that the Java files should be migrated into. In addition, tell SCLM what language it should use for the artifact once it is in the project.

Note: If there is not a suitable language for the artifacts you are migrating in, you must first add a language to the project, then migrate the remote artifacts into the project.

The option Perform File Truncation Checking checks the length of the file name to ensure it adheres to the host system's requirement of keeping data set member names to eight characters or less. If an artifact is found to have a name longer than eight characters, the name is truncated and written to a translation table where its original file name is mapped to its newly truncated name. In this way, artifacts whose file names do not adhere to the 8-character limit can still be migrated into an SCLM-managed project with their unique file name as shown in Figure 24-51.

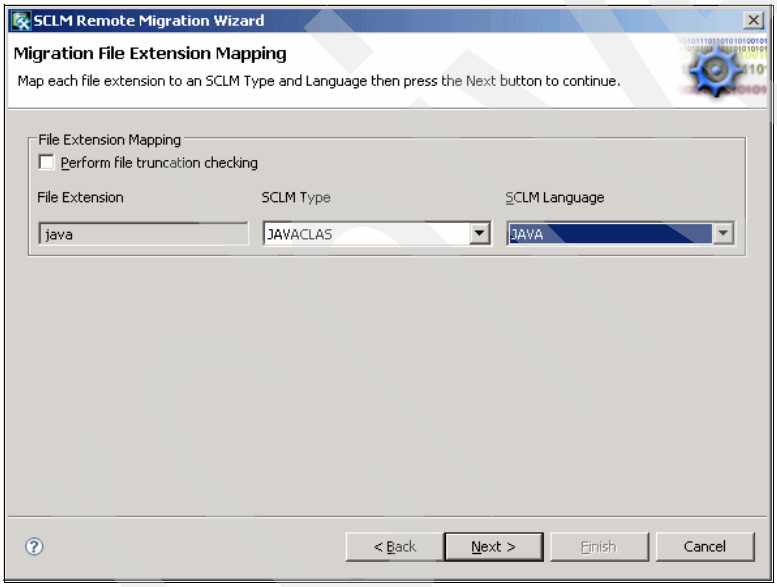


Figure 24-51 Specify the data set type and SCLM language to use for the artifacts

Once the options have been specified in the Migration File Extension Mapping window, click **Next** as shown in Figure 24-52.

The artifact names are written to a translation table if they are longer than 8 characters. They are then written to a host-based file and migrated into the project from the file they were written to. A migration report will be returned for your review once the migration has been completed, as shown in Figure 24-52.

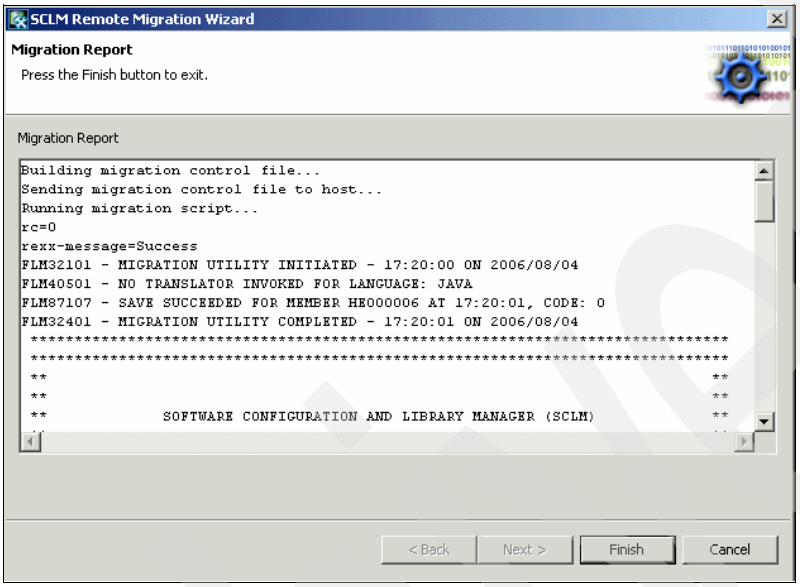


Figure 24-52 The Remote Migration Report

To view the migrated artifacts that now reside in your SCLM-managed project data sets, You can go to an ISPF Data Set List Utility to view the members as shown in Figure 24-53.

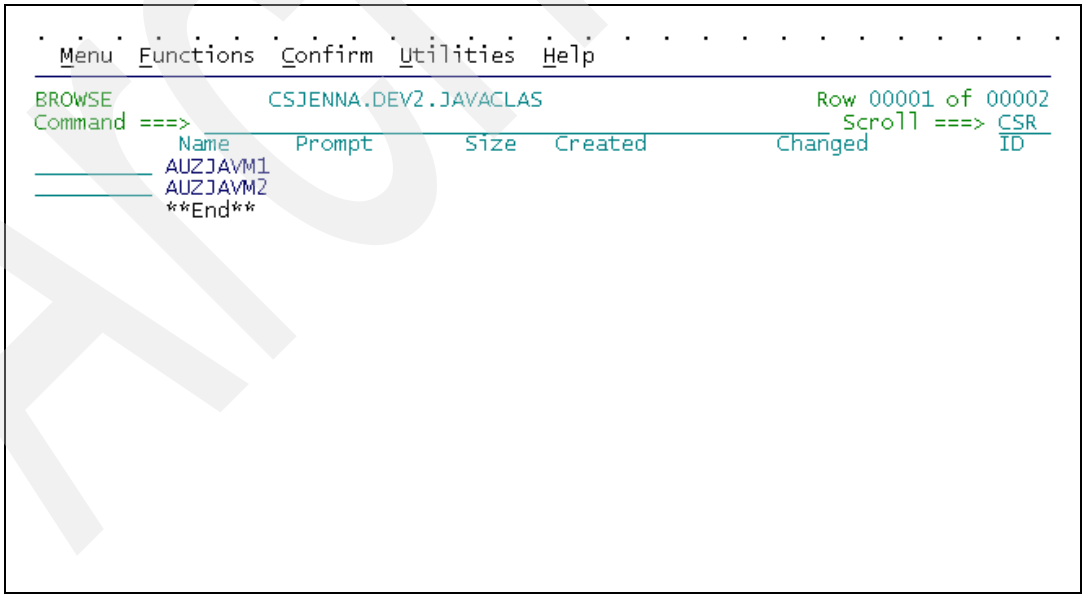


Figure 24-53 The Java members are now in the specified source code data set

24.6 The Architecture Definition Wizard

Architecture Definitions, or ARCHDEFs, tell SCLM how pieces of the project's source code fit together and how base SCLM is to treat specific members within a project when builds and promotes are performed in SCLM.

You can automatically generate SCLM architecture definitions from an existing load module or parse existing JCL into an ARCHDEF. You can also manually create ARCHDEFs from scratch.

24.6.1 Creating an ARCHDEF from a load module

To create an ARCHDEF from a load module means that ISPF's ABMLIST utility will parse through a load module and its CSECTS to create an architecture definition. To use this method to create an ARCHDEF, follow the instructions below.

Open the project you want to create an architecture definition on. Once the project is open in the Project Editor frame, select SCLM from the menu bar above, then click **Architecture Definition Wizard** as shown in Figure 24-54.

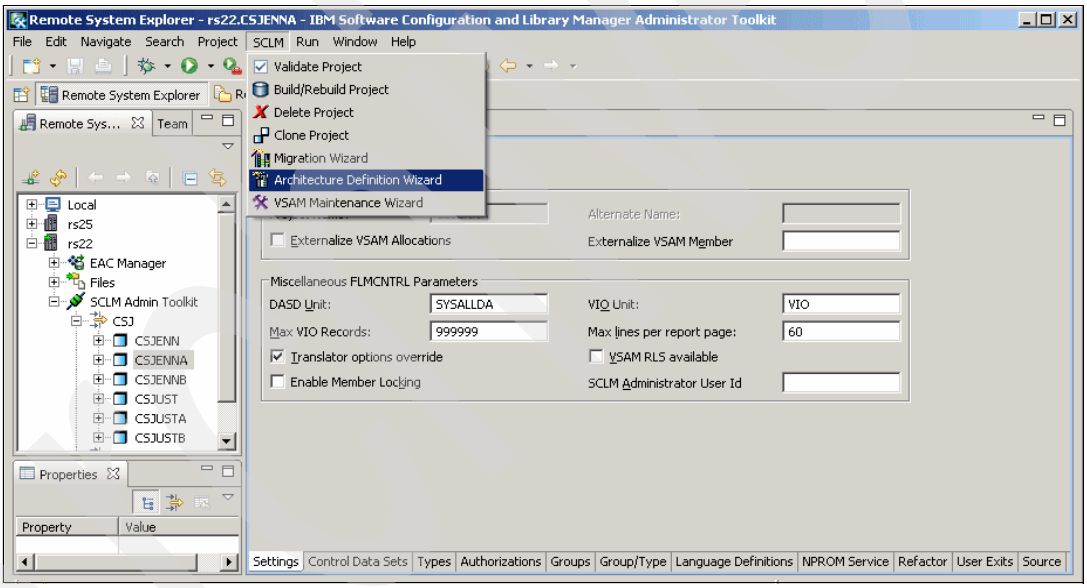


Figure 24-54 Select Architecture Definition Wizard from the pull-down menu

When the Architecture Definition Wizard opens, as shown in Figure 24-55, specify the project group you want to create the architecture definition for, and specify the type of data set the Admin Toolkit is to store the architecture definition into. Select the radio button, **Create New Architecture Definition**, and then click **Next** as shown in Figure 24-55.

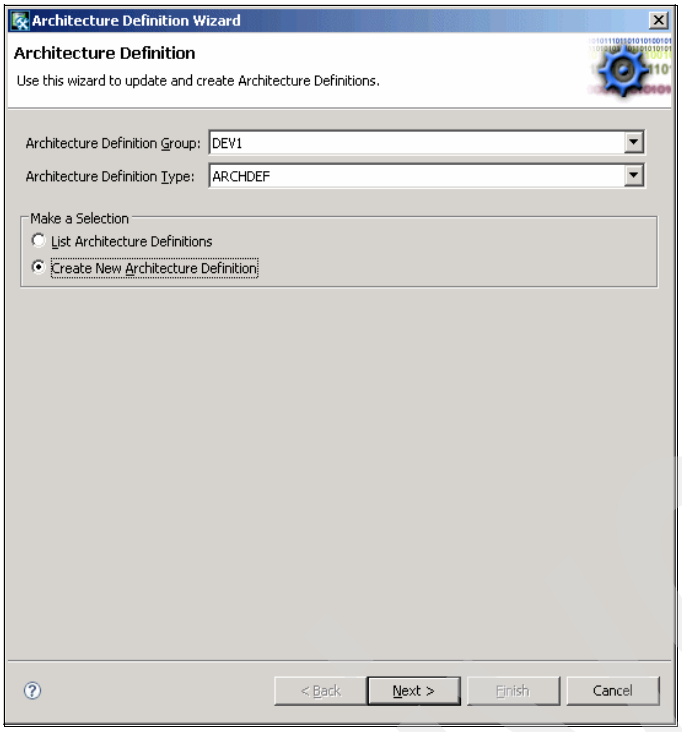


Figure 24-55 Architecture Definition Wizard window

When the window, Create Architecture Definition, is displayed, provide the following information to create an architecture definition:

- The **Architecture Definition Name** should be the PDS member name that gets generated into the data set name provided on the previous panel

Select the check box Multiple Architecture Definition if you will be creating more than one architecture definition at the same time. To view the task flow to create one architecture definition at a time, view the section “Creating an ARCHDEF from JCL” on page 639.

- Specify the **Kind** of architecture definition to create
- Specify the **Language** in the project that the Admin Toolkit is to use to create the architecture definition
- Specify the language to use as the **Linkage Editor Language**
- Tell the Admin Toolkit where SCLM is to put the **output** from building the architecture definition
- Tell the Admin Toolkit where SCLM is to place the **listing** from building the architecture definition
- Select the **Authorization Code** to use with the artifacts defined to this architecture definition
- Specify an optional **Change Code** to associate with this architecture definition
- Specify the **Mode** to use when creating this architecture definition

Select the radio button, **Create from existing JCL**, and click **Next** as shown in Figure 24-56.

Archived

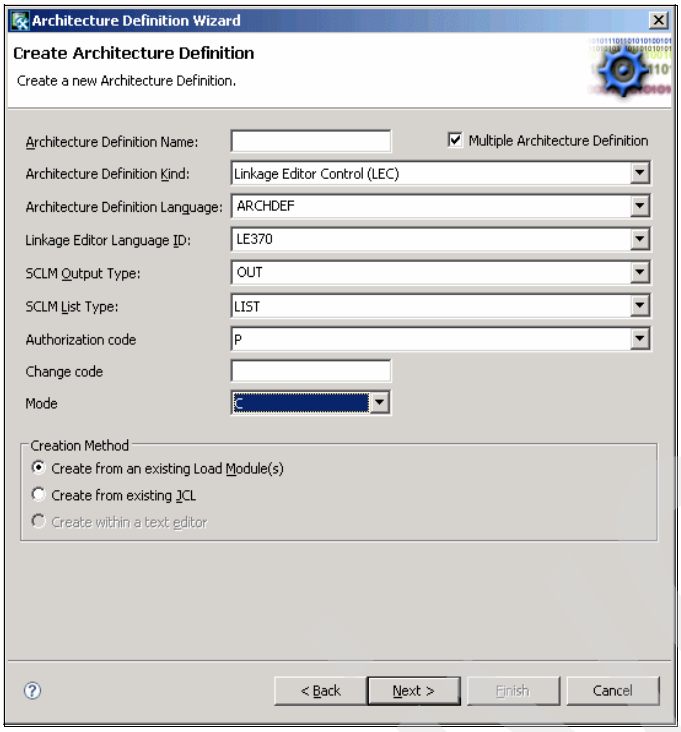


Figure 24-56 The Create Architecture Definition window specifies how to create the ARCHDEF

Note: When the check box Multiple Architecture Definition is selected, the Admin Toolkit will create multiple architecture definitions of the same kind using the same parameters for all ARCHDEFs. If you must specify different parameters for any of the architecture definitions you will create, you must create those architecture definitions separately.

Because the check box, Multiple Architecture Definitions, was selected on the previous window, the window, Multiple ARCHDEF Creation, appears. This allows you to select the project artifacts to include, as shown in Figure 24-57.

Upon initially entering this window, the area, Selected Members, will be blank. To add artifacts to the Architecture Definition Wizard, click **Add**.

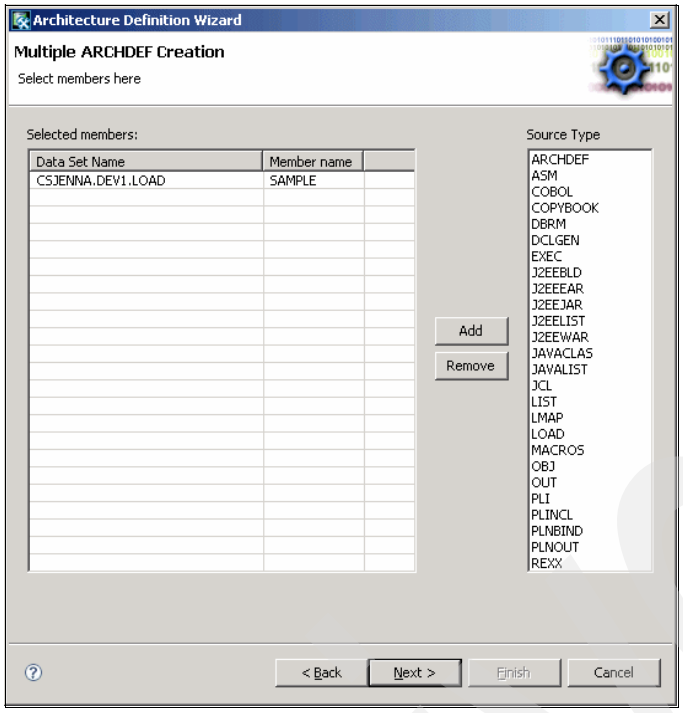


Figure 24-57 Specify the artifacts to create an architecture definition for

When the window, Data set Selection Dialog appears, as shown in Figure 24-58, provide the data set name that contains the artifact you want to include. The data set name may be partially wild carded. Click **Browse** to see a list of data set names matching the pattern provided. These will be displayed in the left-hand frame, Data Set Name.

When a data set in the left-hand frame is selected, the associated members will be displayed in the right-hand frame, Member Name. You can select multiple members from the right-hand frame by holding down the Control key while selecting your choices. When you have selected all artifacts you wish to include, click **OK**.

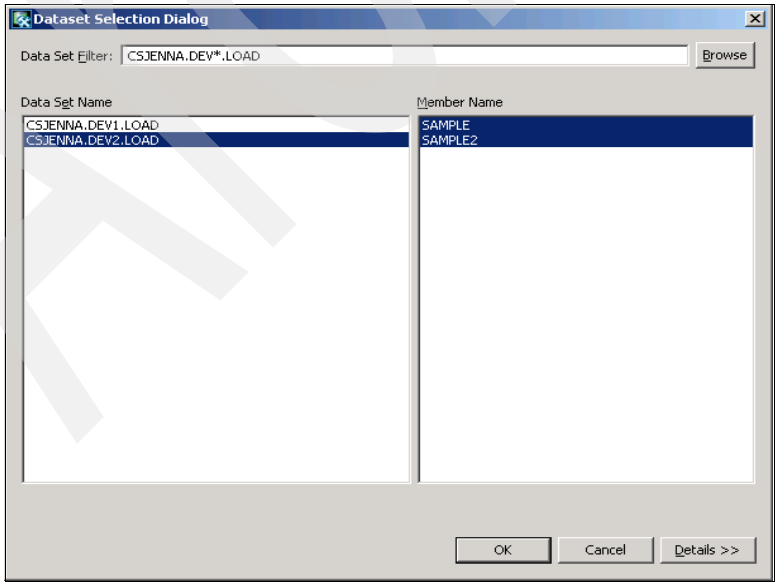


Figure 24-58 Select the members you want to create ARCHDEFS on

If you must include members from multiple data sets, simply click **Add** from Figure 24-57 on page 635, specify the data set name, and then its members.

Once you have included all members to create architecture definitions for, specify the Source Type of the members. You can only create multiple architecture definitions on artifacts of the same kind. In this example, the artifacts are all LOAD module members. If you must create an architecture definition for an artifact that is of another Source Type, you must create it separately.

To Remove an artifact from the Architecture Definition Wizard, simply highlight the data set and member name in the left-hand side of the frame and click **Remove**. When you have finished adding members to the list, click **Next** as shown in Figure 24-59.

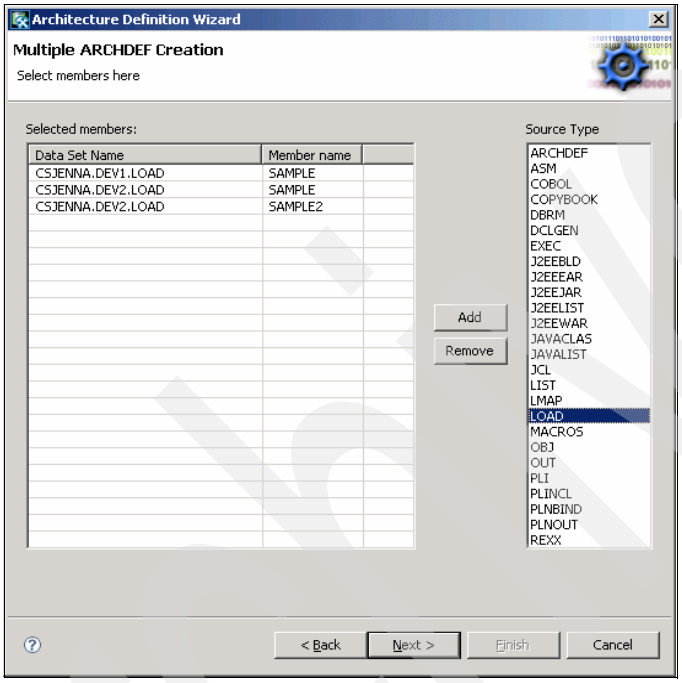


Figure 24-59 Specify the members to include in the Architecture Definition Wizard

The Architecture Definition Wizard creates the ARCHDEFs and returns a report displaying the status of the architecture definitions as shown in Figure 24-60.

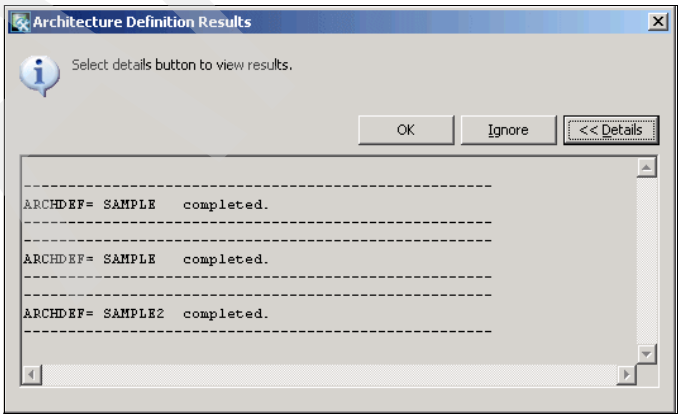


Figure 24-60 The status of creating the ARCHDEFs is returned for your review

To view these or any other architecture definitions that have been created for a project, specify the group and data set type to view, and select the radio button, List Architecture Definition. Then click **Next** as shown in Figure 24-61.

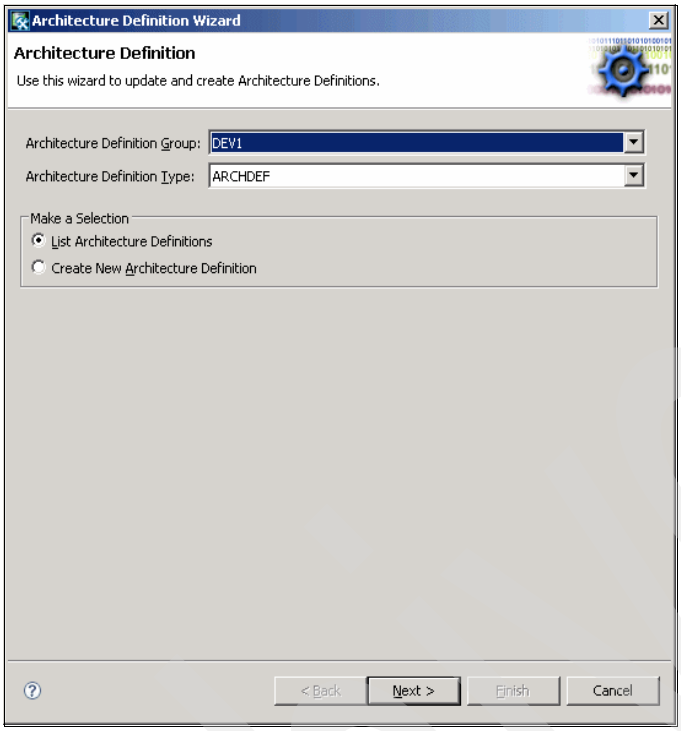


Figure 24-61 Architecture definitions may be viewed

A list of available architecture definitions will be returned for you to make a selection. Highlight an Architecture Definition, then click **Next** as shown in Figure 24-62.

Archived

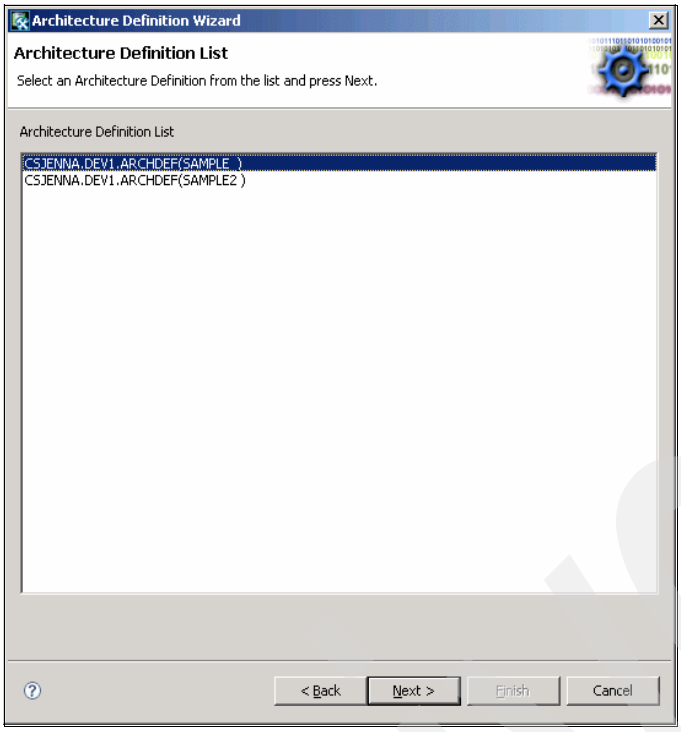


Figure 24-62 Available architecture definitions

The architecture definition is displayed in an editor where changes to the architecture definition can be made as shown in Figure 24-63.

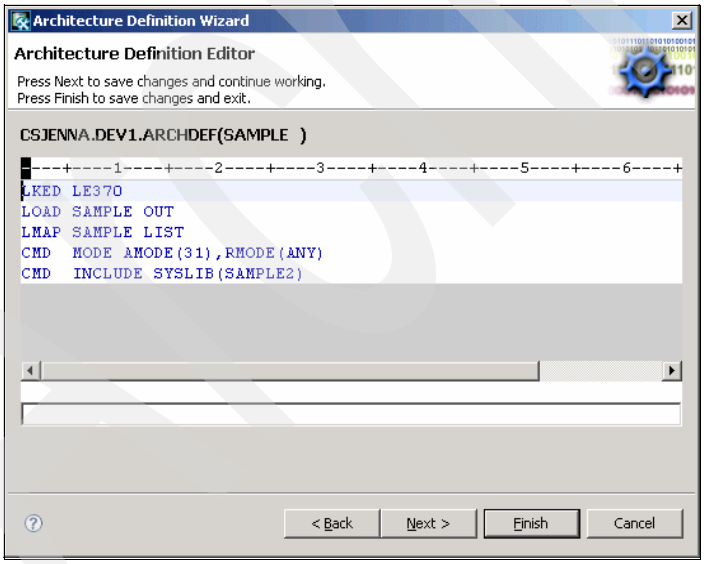


Figure 24-63 You can edit the architecture definition if necessary

Click **Finish** to save your changes and exit from the Architecture Definition Wizard.

24.6.2 Creating an ARCHDEF from JCL

If you have JCL that was used to define the architecture of a project, it can be parsed into an architecture definition for use with SCLM-managed projects. To use this method to create an ARCHDEF, follow the instructions below.

Open the project you want to create an architecture definition for. Once the project is open in the Project Editor frame, select SCLM from the menu bar above, then click **Architecture Definition Wizard**, as shown in Figure 24-54 on page 632.

When the Architecture Definition Wizard opens, as shown in Figure 24-64, specify the project group you want to create the architecture definition for, and specify the type of data set the Admin Toolkit is to store the architecture definition into. Select the radio button, **Create New Architecture Definition**, and then click **Next**.

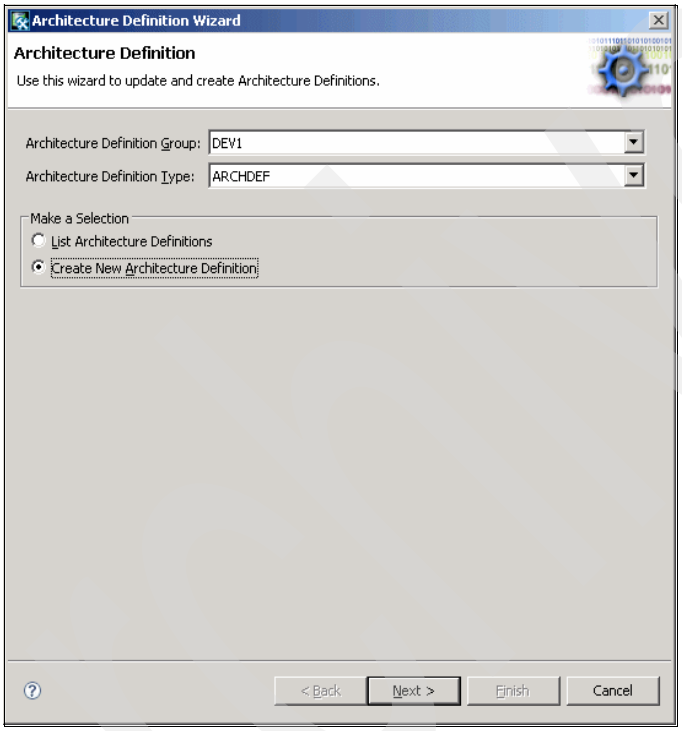


Figure 24-64 Architecture Definition Wizard window

Provide the options to be used to create the architecture definition as shown in Figure 24-65 on page 640. For a description of the parameters in this window, see the section “Creating an ARCHDEF from a load module” on page 632.

When you have completed specifying the options to use in creating the architecture definition, select the radio button, **Create from existing JCL**, then click **Next**.

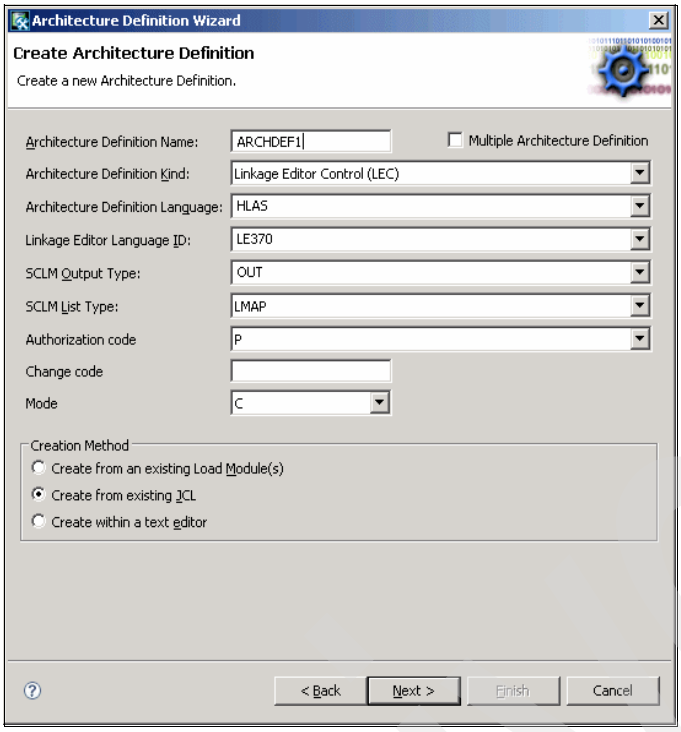


Figure 24-65 The Create Architecture Definition specifies how to create the architecture definition

The Create Architecture Definition window displays a Data Set Name field for you to specify the data set and member name to create the architecture definition for. The data set name can be wild carded with an asterisk. Click **Browse** to locate the data set as shown in Figure 24-66.

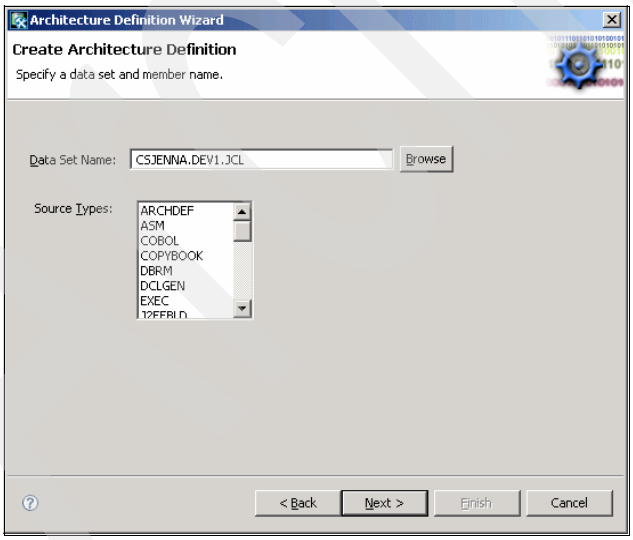


Figure 24-66 The data set name may be wild carded

Figure 24-67 shows the matching data set names to the pattern provided in as well as the members for which to create the architecture definition. When a data set name is highlighted in the left-hand frame, the corresponding members are displayed in the right-hand frame. Select a member, and click **Next**.

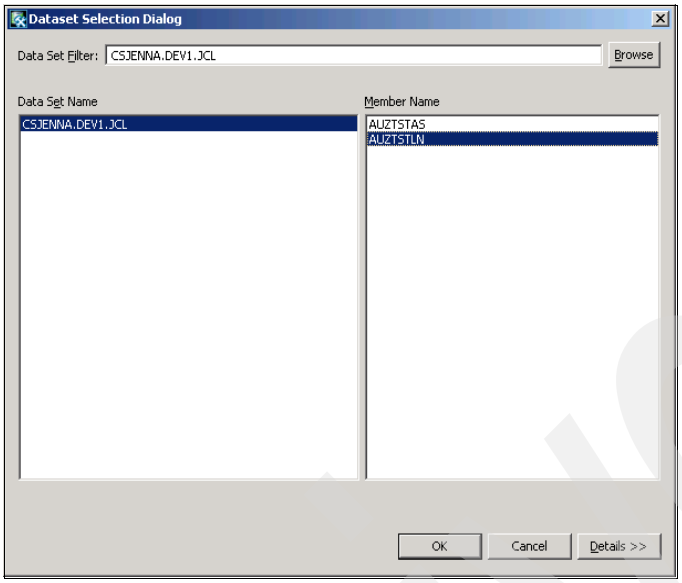


Figure 24-67 Select the member to create an ARCHDEF for

When the window, Create Architecture Definition, is re-displayed, the data set and member name will be listed in the field, Data Set Name. In the field, Source Types, use the Control key to highlight the data sets where the components can be found for the artifact you are creating an architecture definition for. Then click **Next** as shown in Figure 24-68.

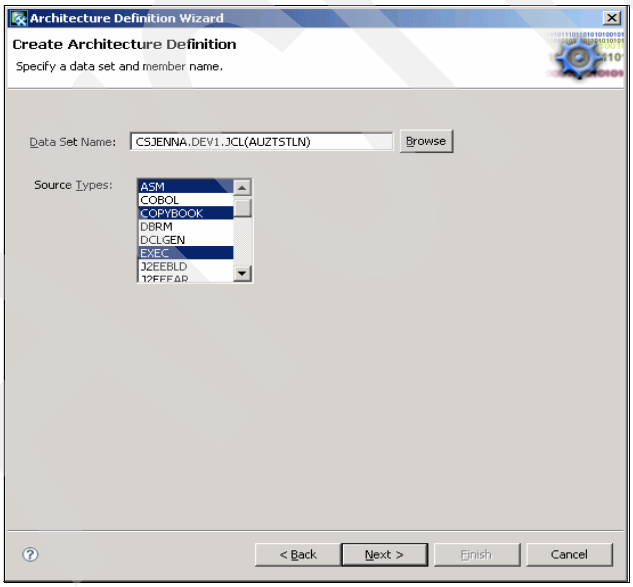


Figure 24-68 The Source Type field specifies where the artifacts associated assets can be found

When the generated architecture definition is returned, as shown in Figure 24-69, the comment “Generated by JCL2ARCHDEF” indicates that the architecture definition was created by parsing JCL. You can make any necessary edits here and your changes will be saved in the specified data set and member name displayed on the Architecture Definition Editor window.

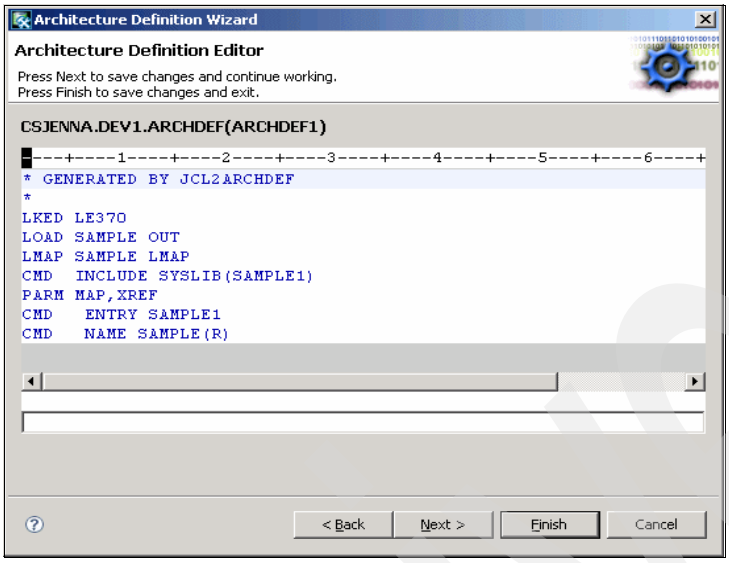


Figure 24-69 The created Architecture Definition is displayed

Click **Finish** to save your changes and exit from the Architecture Definition Wizard.

24.6.3 Creating an ARCHDEF from scratch

You can create an architecture definition manually by specifying the definition yourself. You will be placed into an editor where you can provide the parameters of an ARCHDEF to use for your SCLM-managed project. To use this method to create an ARCHDEF, follow the instructions below.

Open the project you want to create an architecture definition on. Once the project is open in the Project Editor frame, select SCLM from the menu bar above, then click **Architecture Definition Wizard**.

When the first Architecture Definition window is displayed, as shown in Figure 24-70, specify the Group to create the Architecture definition on, and the data set type that the architecture definition is to be created into. Then select the radio button to **Create a New Architecture Definition**, and click **Next**.

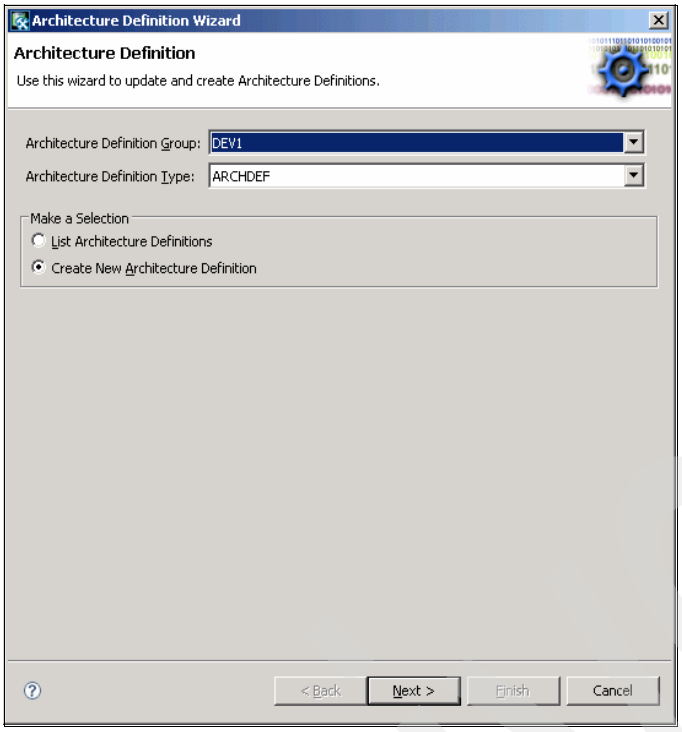


Figure 24-70 Specify the group and type of data set to create the architecture definition into

The window, Create Architecture Definition, appears. Provide the options to be used to create the architecture definition as shown in Figure 24-71. For a description of the parameters in this window, see the section “Creating an ARCHDEF from a load module” on page 632.

When you have completed specifying the options to use in creating the architecture definition, select the radio button, **Create within a text editor**, then click **Next**.

Archived

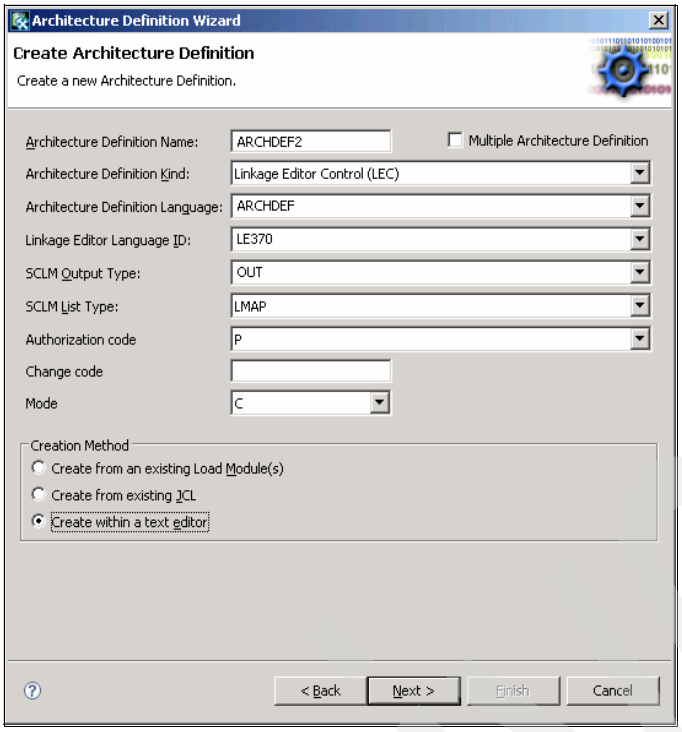


Figure 24-71 The Create Architecture Definition window specifies how to create the ARCHDEF

When the window, Architecture Definition Editor, is displayed in Figure 24-72, you can type in the architecture definition text as needed. Your changes will be saved in the specified data set and member name displayed on the Architecture Definition Editor window.

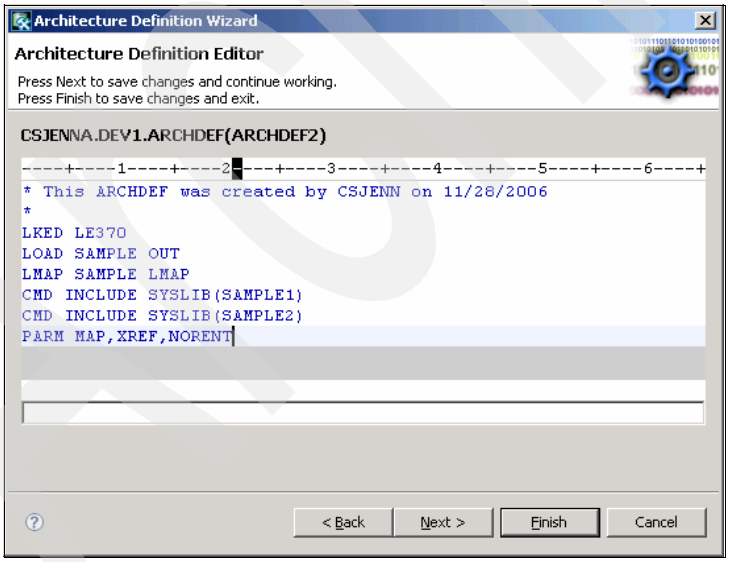


Figure 24-72 Enter the parameters for the architecture definition

Click **Finish** to save your changes and exit from the Architecture Definition Wizard.

24.7 The EAC Manager and RACF data set profile feature

The SCLM Administrator Toolkit workstation-based user interface contains a feature called the EAC Manager. EAC is the SCLM Advanced Edition product called Enhanced Access Control for SCLM for z/OS. Enhanced Access Control is a product that works in conjunction with RACF to provide security at a program level. Users can define profiles to provide greater security for SCLM and its resources so that authorized modifications to SCLM-managed source code can only be made by an SCLM user or program.

The Administrator Toolkit provides a GUI interface for EAC in which users can view access violations, create or edit EAC profiles and add new feature/function pairs. Even though the RACF Data Set Profile node appears under the EAC Manager, you do not have to have EAC installed to create and edit RACF Data Set Profiles.

In order to use EAC in the Administrator Toolkit GUI, a Rules File must be specified, and you must have RACF SPECIAL authority.

Note: EAC does not have to be installed in order to use the RACF Data Set Profile feature. RACF profiles may be accessed via the Admin Toolkit independent of the EAC Manager feature.

24.7.1 The EAC Manager node

The EAC Manager allows you to manage EAC profiles from within the Administrator Toolkit. To access the EAC Manager, open the workstation client, and click the z/OS host connection that you want to access the EAC Manager from. If you have not already logged into that z/OS host, the Admin Toolkit will prompt you to do so.

Once you are logged in to the z/OS host and the node is expanded, the EAC Manager node will be visible. Expand the EAC Manager node to see the EAC Resources, as shown in Figure 24-73.

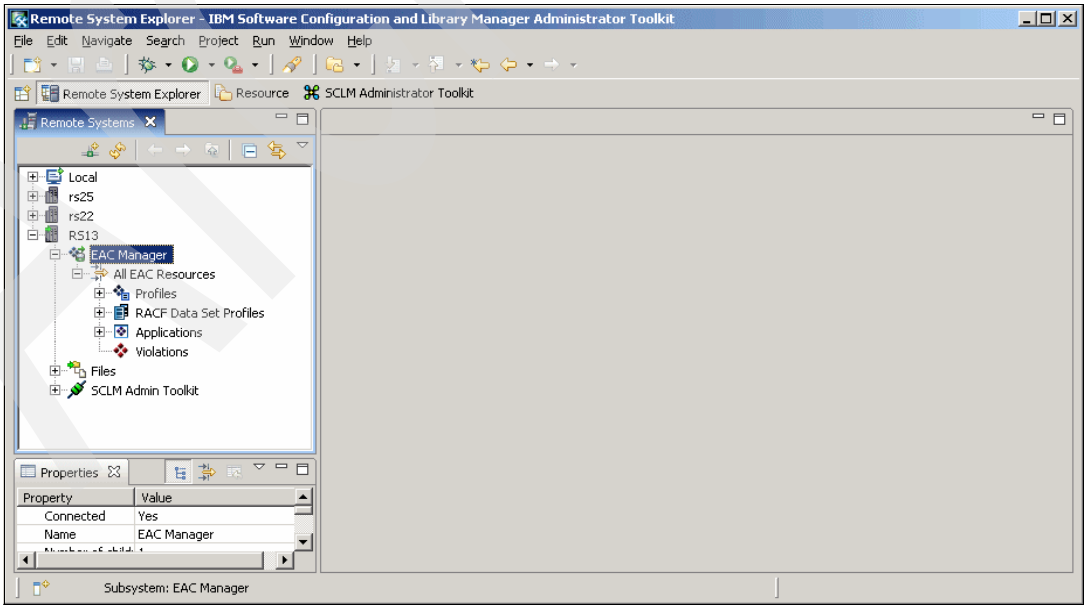


Figure 24-73 EAC Manager in the Remote Systems View

With the EAC Manager, you can:

- Click **Profiles** to view, edit, create or delete EAC profiles
- Click **Applications** to view, edit, create or delete EAC Application/Function pairs
- Click **Violations** to view EAC violations

View, edit, create or delete EAC profiles

A profile identifies a data set or RACF data set profile that is to be validated by EAC before a user can perform an action. Expand the node Profiles to see a list of existing EAC profiles that have already been defined to the EAC, as shown in Figure 24-74.

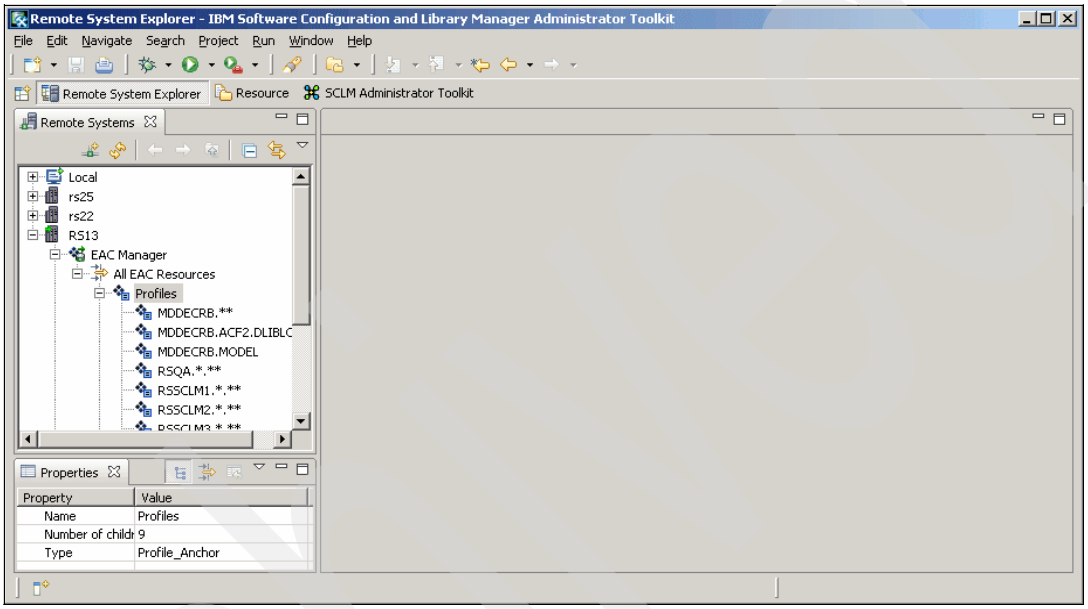


Figure 24-74 Expand the node to see the EAC Profiles

View or edit an existing EAC profile

To view an existing profile, double-click the profile name. The EAC Profile Dialog window opens displaying the options specified for the profile when the profile was created as shown in Figure 24-75.

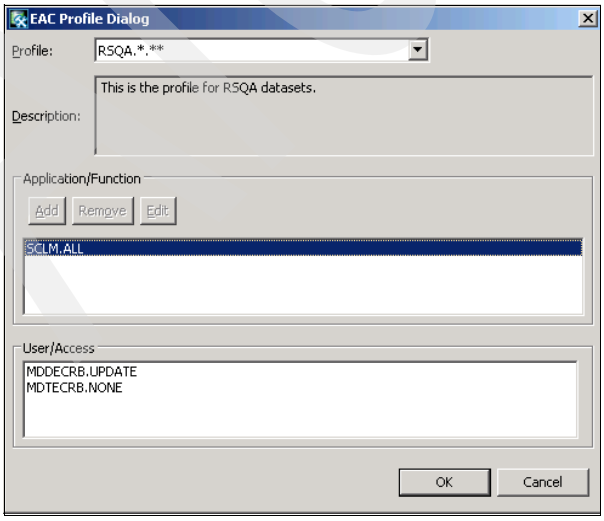


Figure 24-75 You can only Edit a profile if you have authority to do so

To edit an existing profile, highlight a profile and then right-click to open a pop-up menu. Select Edit. The EAC Profile Dialog window opens allowing you to edit the profile if you have authority to do so.

Create a new EAC profile

To create a new profile, highlight the node Profiles in the Remote Systems View, and right-click it to get an additional menu. When the menu appears, click **New**, then click **Profile** as shown in Figure 24-76.

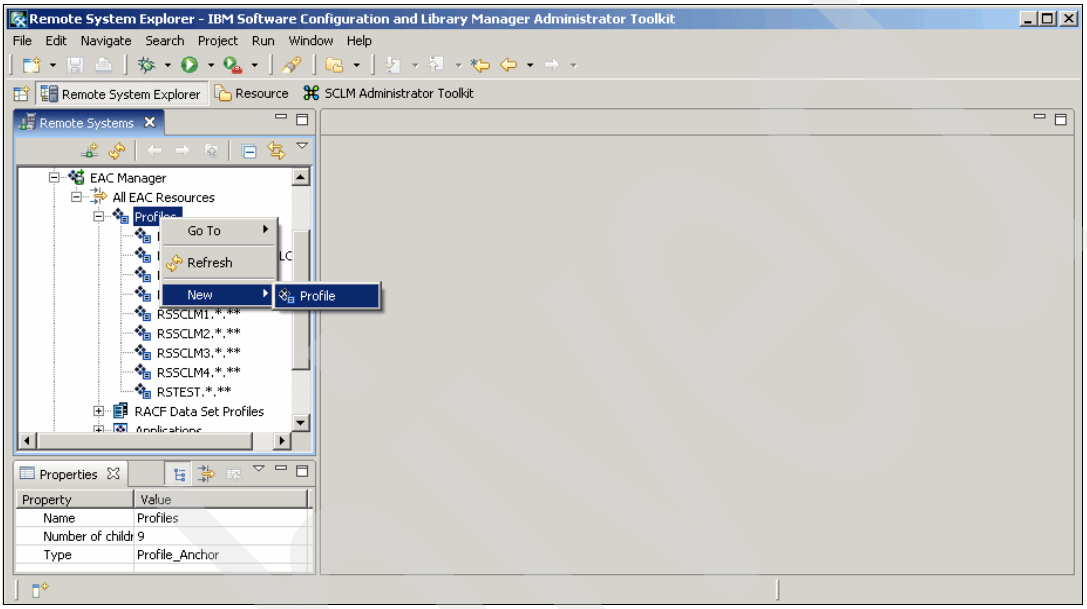


Figure 24-76 Right-click Profiles to create a new one

When the EAC Profile Dialog window opens as shown in Figure 24-77, specify the following information to create a new EAC Profile:

- The RACF data set **Profile** you want to create an EAC Profile for
- An optional **description** for the profile
- The **Application/Function** pair you want the profile to monitor
- The **user** IDs and corresponding **access** that the profile is to allow

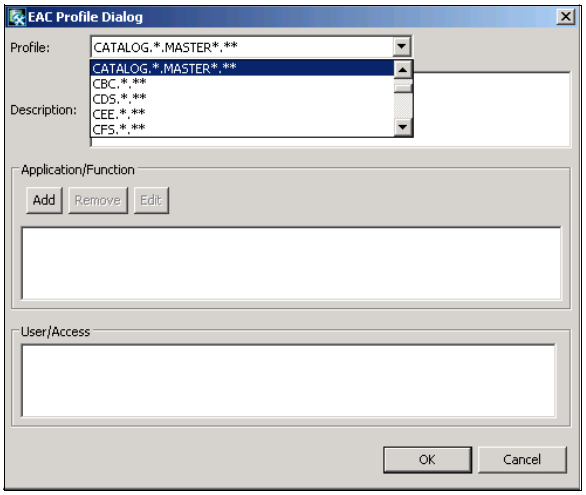


Figure 24-77 Provide the definitions for a new Profile

When you select the Application/Function pair button, **Add**, the Application Selection Dialog window opens allowing you to select the application/function pair to use for the EAC profile. An Application/Function pair is a combination of the application you are specifying access to, and the function during which EAC is to enforce the access granted in the profile.

Specify the following information on the Application Selection Dialog window as shown in Figure 24-78:

- The **Application/Function** that you want added to the EAC Profile
- The **user ID** and type of **access** to grant for the application and function

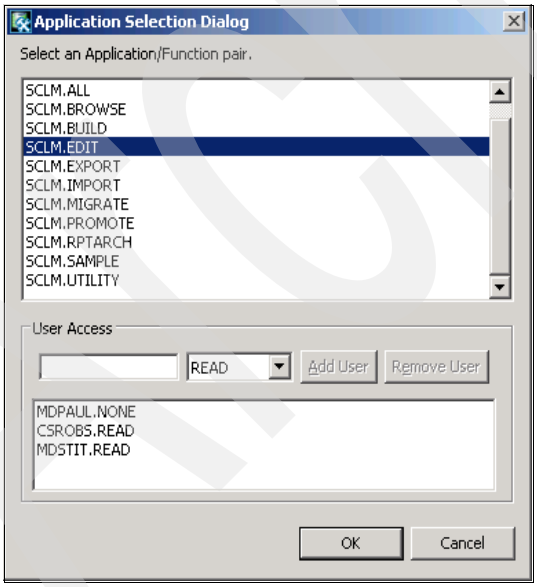


Figure 24-78 Select the Application/Function pair and the access to grant

Once the required fields are specified, click **OK**. The EAC Profile Dialog window is displayed with your options filled in. Once you are done specifying the options for your new EAC profile, click **OK**. The Profiles listed in the Remote Systems View will refresh, showing your newly created EAC profile in the list, as shown in Figure 24-79.

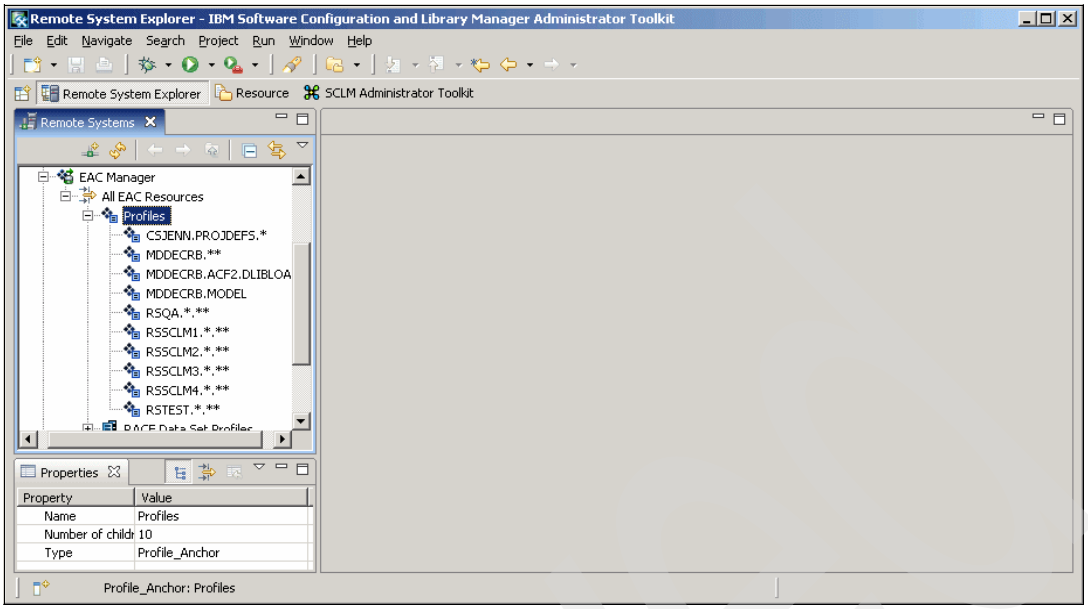


Figure 24-79 The new Profile appears in the Remote Systems View

Delete an existing profile

To delete an existing EAC Profile, highlight a profile in the list. Right-click the profile to get an additional pop-up menu. Select Delete to delete the highlighted profile, shown in Figure 24-80.

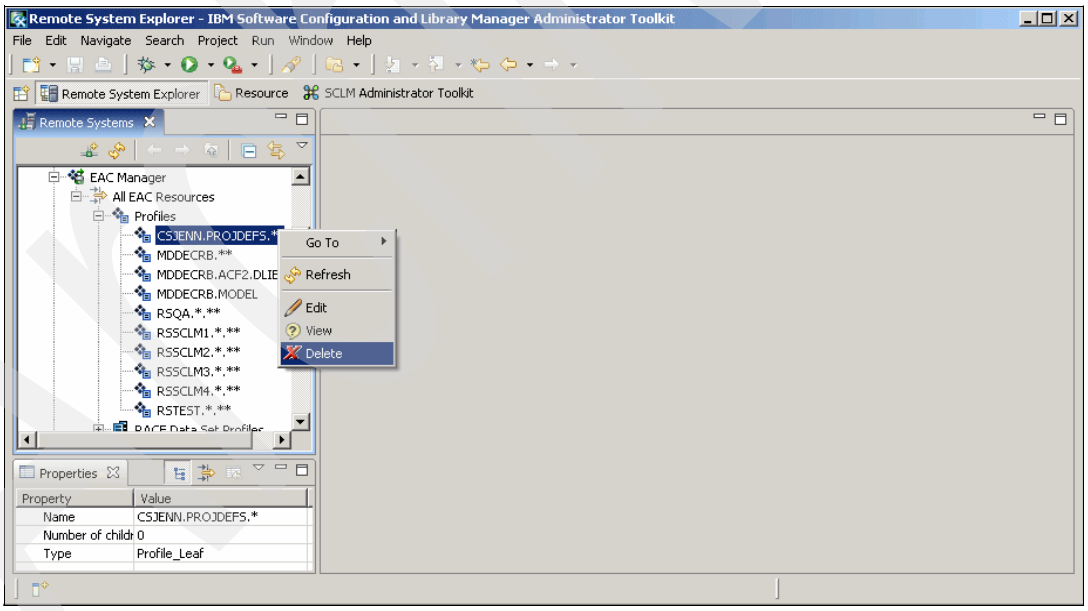


Figure 24-80 Right-click a profile to delete it from the list

View, edit, create or delete EAC applications

Click the node **Applications** in the Remote Systems View to get a list of all program applications defined to EAC, as shown in Figure 24-81.

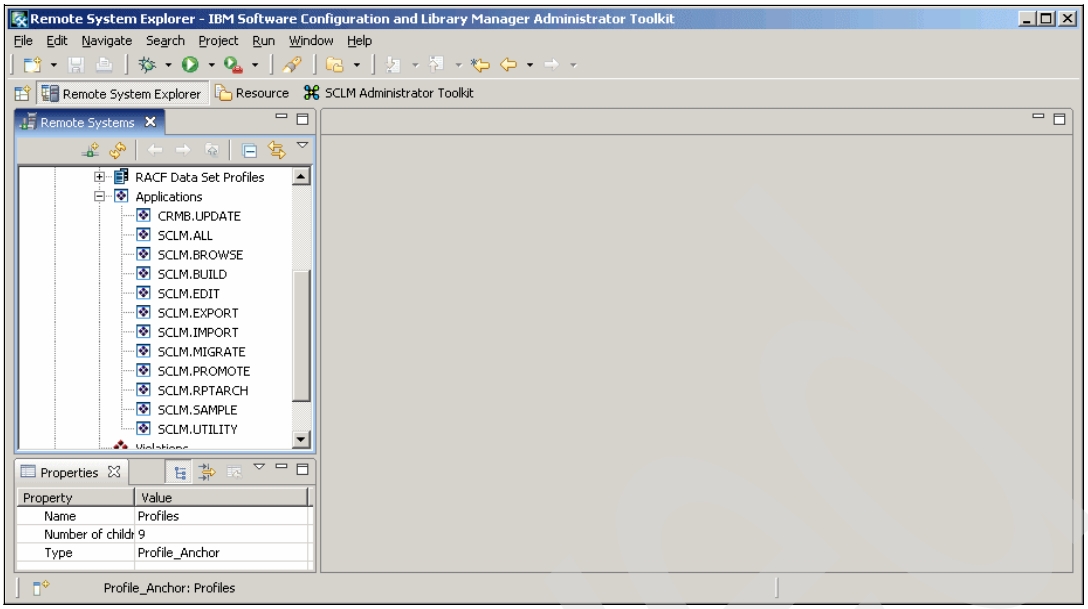


Figure 24-81 The list of Applications defined in EAC

View or edit an EAC application

In the Remote Systems View, highlight an application you want to view and right-click the application. A pop-up menu will appear on which you select View. The EAC Application Dialog window will display the options that were specified for the application when it was created as shown in Figure 24-82.

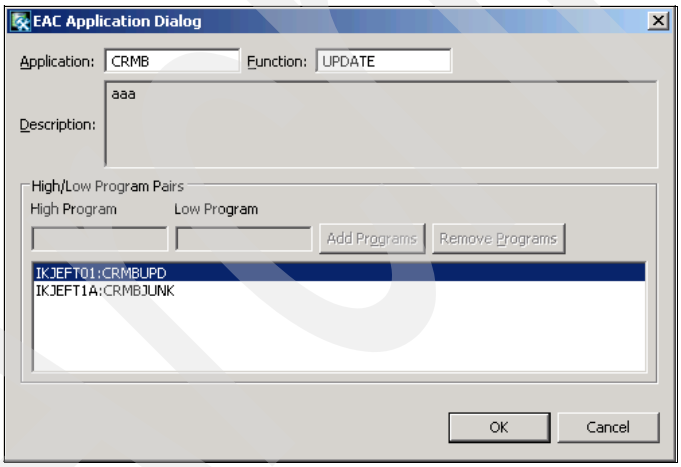


Figure 24-82 The Application Dialog in VIEW mode

To edit an EAC Application, highlight the application name in the Remote Systems View, then right-click the application to access the pop-up menu. Click **Edit**. When the EAC Application Dialog window opens, you can edit the application's options if you have authority to do so.

Create an EAC application

To create a new EAC Application, highlight Applications in the Remote Systems View, and right-click to receive a pop-up menu. Select New, then select Application. Likewise, you can highlight EAC Manager and right-click, then select **New**, then select EAC Application as shown in Figure 24-83.

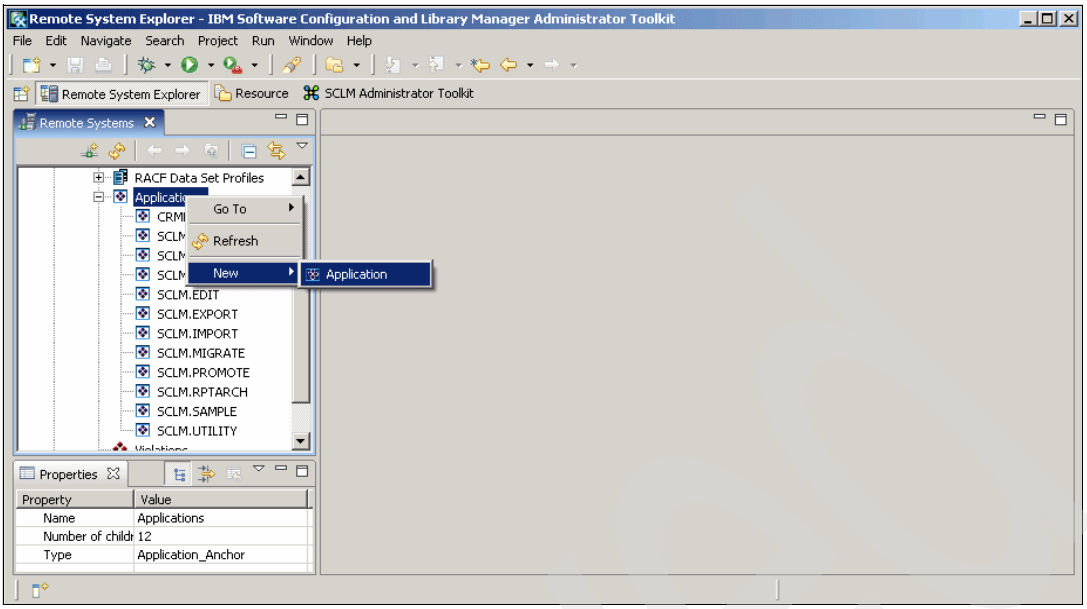


Figure 24-83 Create a new application

When the EAC Application Dialog window is displayed as shown in Figure 24-84 on page 651, you can specify the following options to create a new application:

- The name of the **application**
- The name of a **function** that will be performed within this application
- A **description** for the application/function
- The name of the **high and low programs** within a program chain that should be associated with this EAC application

Click **Add Programs** to add the high/low program pair to the application, as shown in Figure 24-84. To remove a high/low program pair, highlight the program pair, then click **Remove Programs** to have them removed from the defined application.

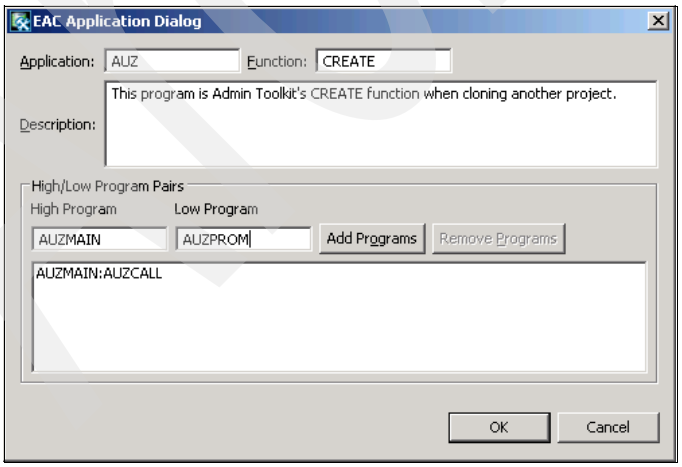


Figure 24-84 EAC Application Dialog window

When you have completed defining the parameters to the EAC application you are creating, click **OK**. The Remote Systems View will display your newly created Application in the tree as shown in Figure 24-85.

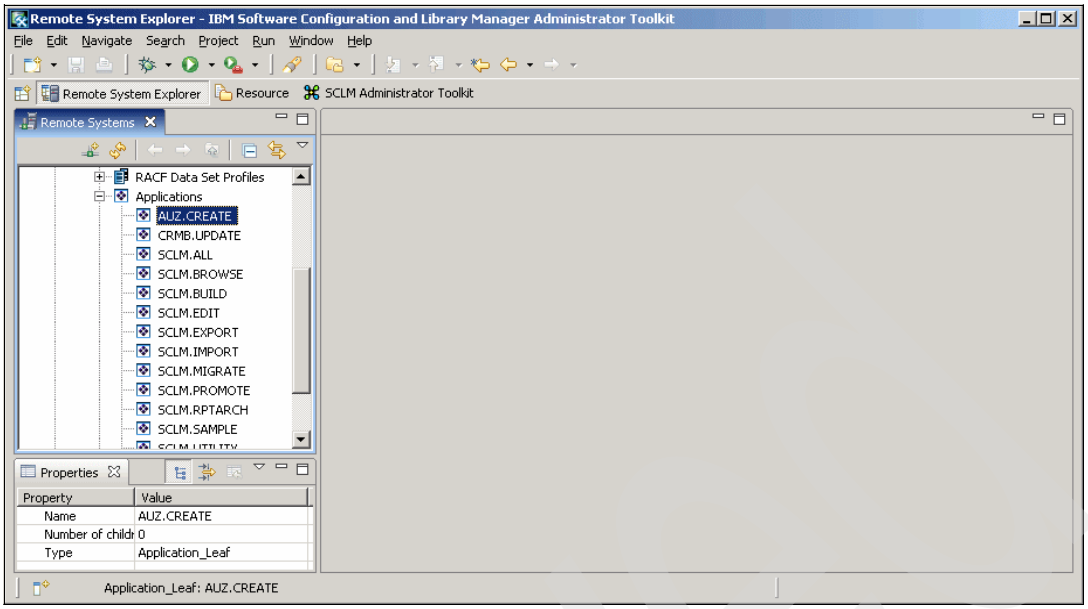


Figure 24-85 New application in the list

Delete an EAC application

To remove an EAC Application from the list, highlight an application in the Remote Systems View and right-click to view the pop-up menu. Click **Delete** to delete the application from EAC, as shown in Figure 24-86.

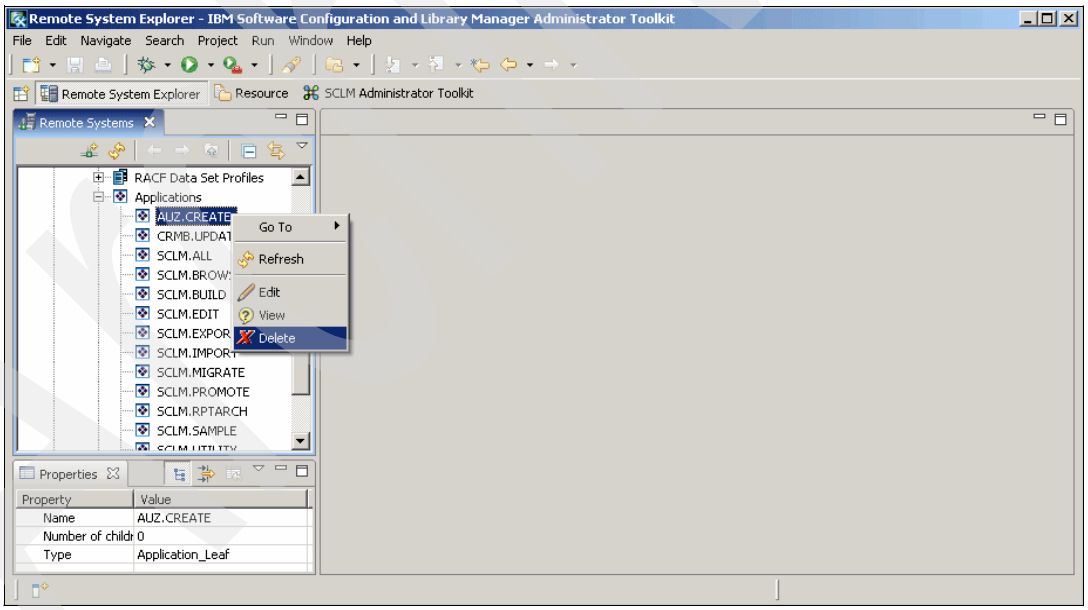


Figure 24-86 Delete EAC Application

View EAC violations

You can monitor the activity within EAC by tracking violations. To display a list of violations and the respective user ID and action attempted, either double-click the node **Violations** in the Remote Systems View, or highlight the node Violations, and right-click to get a pop-up menu that allows you to view the violations as shown in Figure 24-87.

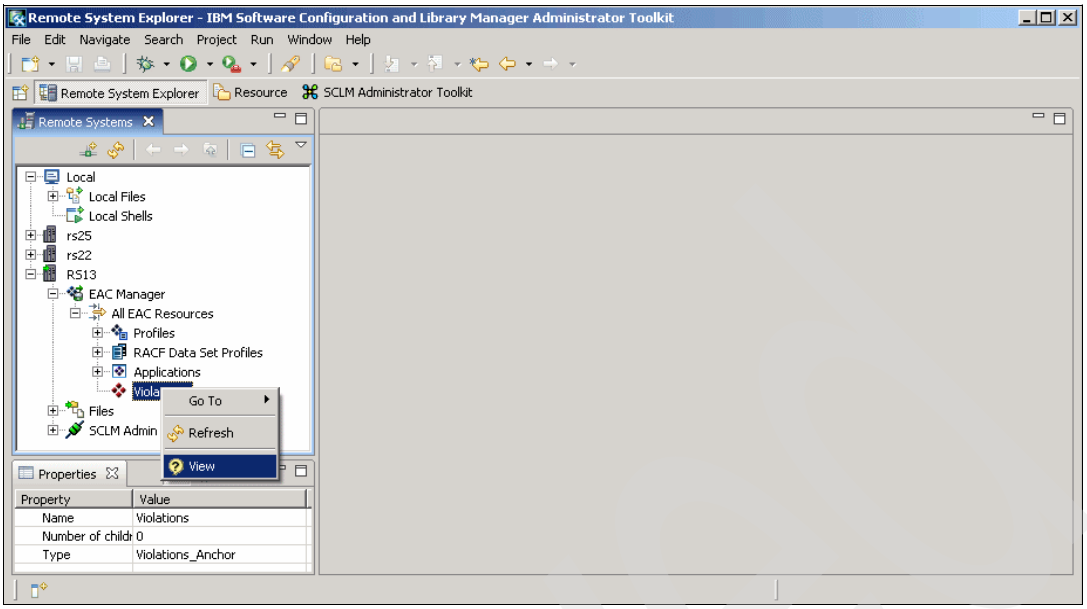


Figure 24-87 EAC14 View EAC Violations

Select the violation you want to view, and then click **Next** as shown in Figure 24-88.

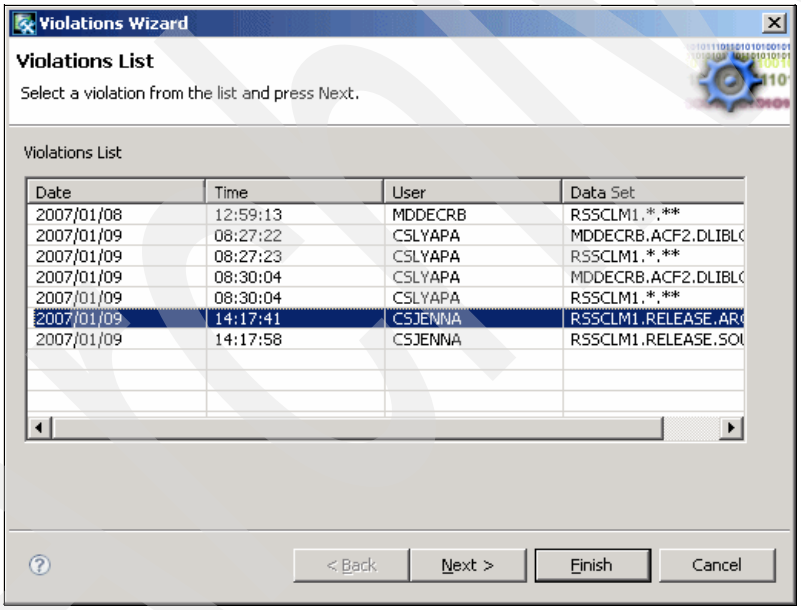


Figure 24-88 EAC violations listed

The Validation Detail Dialog window displays the following information about the violation you selected, shown in Figure 24-89:

- The fully qualified data set name that EAC used to validate a user's access on the resource.
- The application definition that EAC used to validate a user's access on the resource.
- The function that was defined to the application
- The user ID or RACF group ID that EAC used to validate a user's access on the resource.

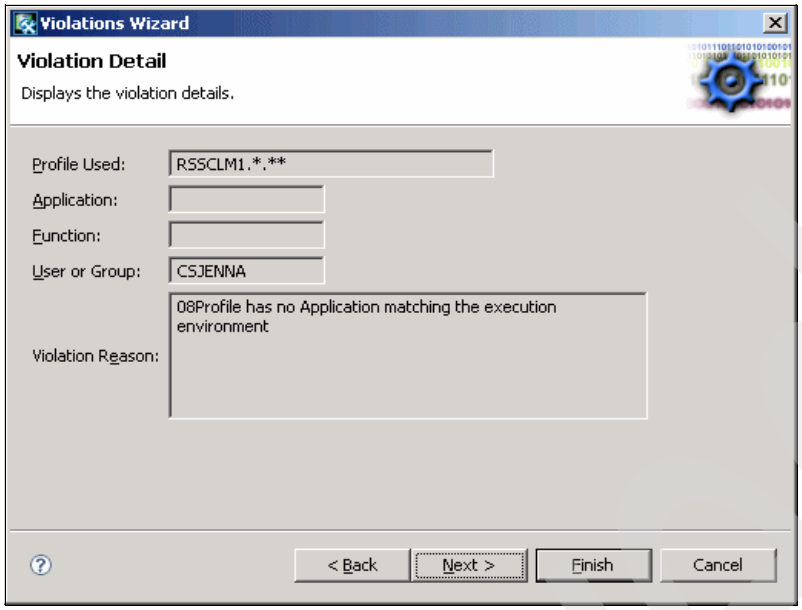


Figure 24-89 Violations Detail Dialog window 1

Click **Next** to view more information about the violation. The continuation of the Validation Detail Dialog panel displays the following information on the violation you are viewing, as shown in Figure 24-90:

- The reason for the violation
- The name of the object that access was attempted on, either the profile name or the data set name that was accessed
- The date and time the access was attempted
- The user ID that requested the access
- The name of the RACF current connect group associated with the access request
- The access that is required by RACF to allow a request to process
- The actual privilege returned to RACF by EAC
- The program in control when the data set access was attempted

Archive

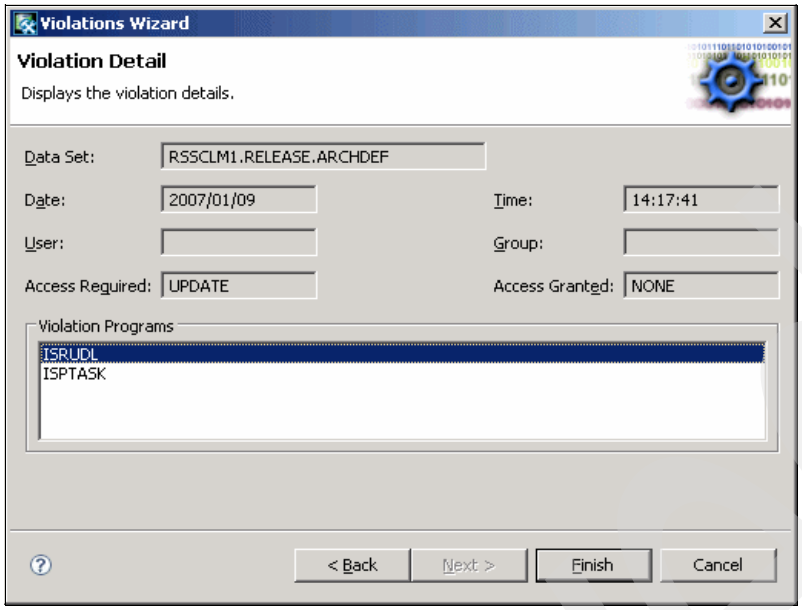


Figure 24-90 Violations Detail Dialog window 2

Click **OK** to close the Violations Detail Dialog window and return to the Remote Systems View to view another violation.

24.7.2 The RACF data set profile node

Even if you do not have EAC installed and in use on your z/OS host, you can still use the RACF Data Set Profile feature. Using the interface the Admin Toolkit provides, you can view, edit and create RACF data set Profiles.

To access the functions for the RACF Data Set Profile, expand the node in the Remote Systems View, as shown in Figure 24-91.

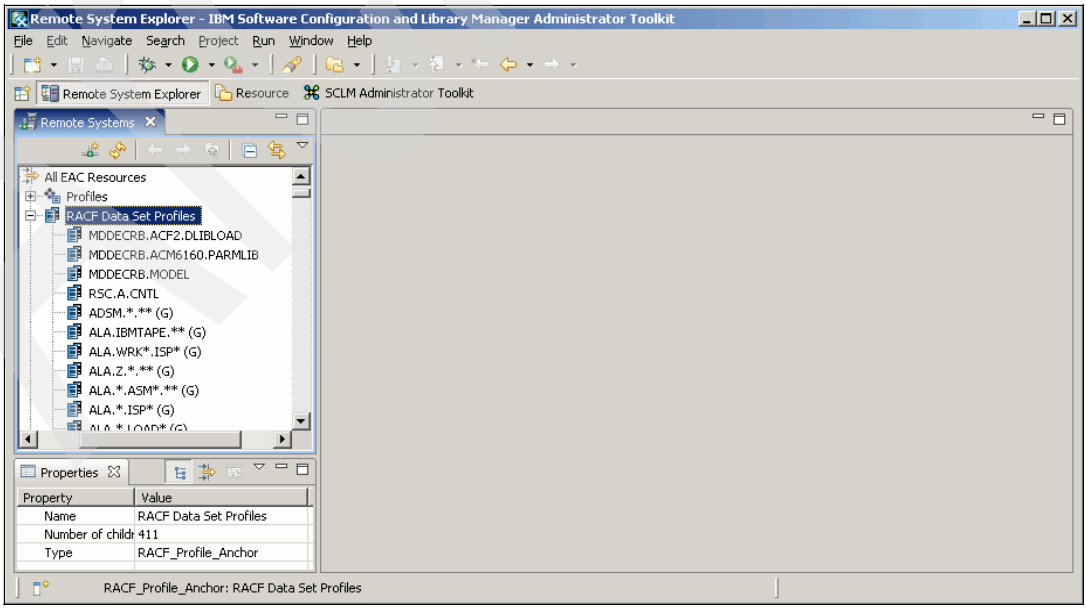


Figure 24-91 Expanded RACF Profile list

To view a RACF profile, highlight an existing profile, and right-click to get a pop-up menu. Then select View, as shown in Figure 24-92.

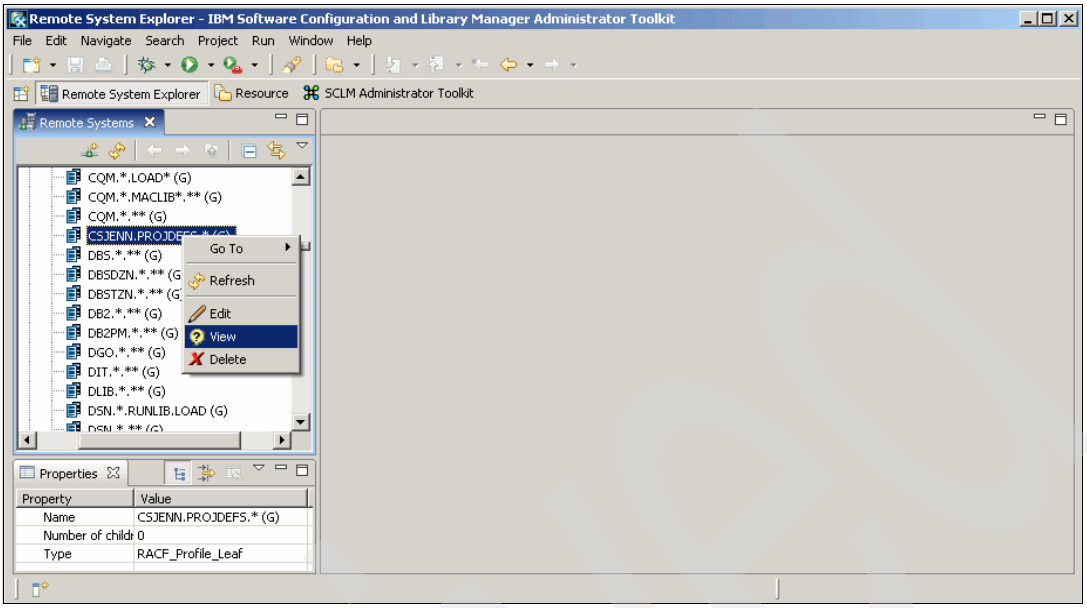


Figure 24-92 Select VIEW to see how a RACF profile was defined

To edit a RACF profile, either highlight an existing profile and right-click it to get a pop-up menu. Select Edit to edit the RACF Profile. Or, double-click a RACF profile to open the RACF Data Set Profile Wizard panels.

When the RACF Data Set Profile Wizard appears, it will display the options that were specified for the RACF Data Set Profile when it was created.

To create a new RACF Data Set Profile, right-click the node **RACF Data Set Profile** to receive a pop-up menu. Select option New, then select RACF Data Set Profile, as shown in Figure 24-93.

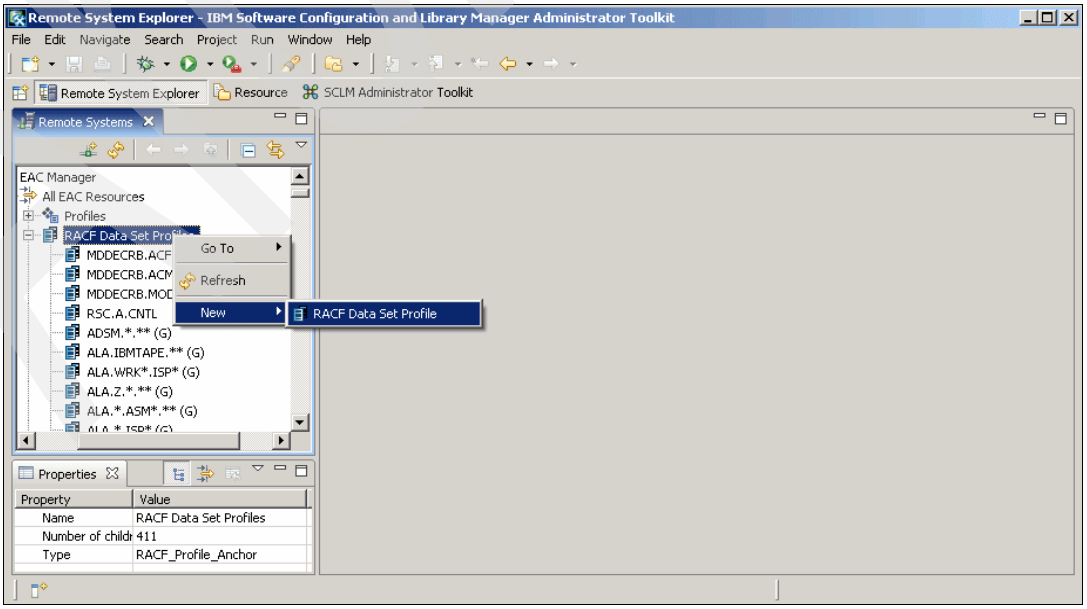


Figure 24-93 Create new RACF Profile

The RACF Data Set Profile Wizard opens with the first window. The following information can be specified to create a new RACF Profile:

- Specify a **RACF Profile** name up to 44 characters long
- Set **SETROPTS** if you want a refresh to update any changes to this profile
- Set the **Warning** option if you want to allow users to access the resource but receive a warning message of the violation
- Set the **Erase** option if z/OS data management is to physically erase the DASD data set extents if the data set defined to this profile is deleted.
- Specify the **Universal Access** for the resource defined in this profile
- Specify **Your Access** to this data set
- Specify the **Owner** of the profile, either a user ID or a group ID
- Specify the user ID or group ID to **Notify** if access to the data set is denied
- Specify the **Resource Owner** of the data set that this RACF Profile is defined on
- Specify the name of the group under which this profile was **Created**
- Specify the **Dataset** profile **Type**
- Specify the **Unit** type that the data set resides on
- If this is a VSAM data set, then provide the **Volume** that the data set resides on
- Specify the security **Retention Period**
- Specify the **Security Level**, which is the minimum level required to access a resource defined by this profile
- Specify the **Security Label** that identifies the security category

When you have completed entering in the values you want to create your RACF data set profile with, click **Next** to proceed to the next Wizard window as shown in Figure 24-94.

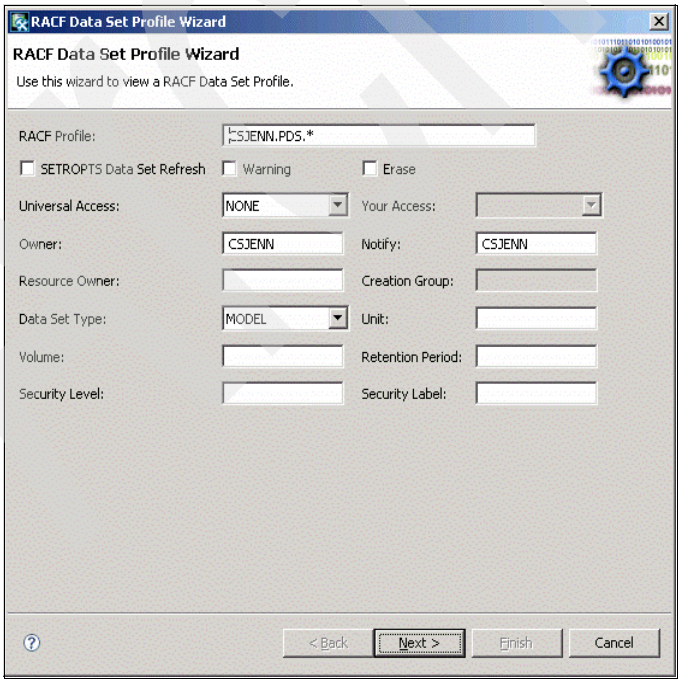


Figure 24-94 Specify parameters to create a new RACF Profile

The second window in the RACF Data Set Profile Wizard is displayed with options to define auditing functions, as shown in Figure 24-95. You can specify the following parameters:

- A security **Category** name that represents the security level needed to access the RACF resource. Your installation has its own security categories defined. Check with your RACF Administrator for valid security categories.
- The type of **Auditing** that should be monitored for all requests on the resource defined to the profile.
- The type of **Access** requests that should be monitored.

Once you have entered a Category name, click **Add** to add it to the profile. Also define in your profile what actions should be audited. For example, you can choose to audit only those failed attempts to UPDATE a resource protected by the profile. In this example, select FAILURE from the left-hand drop-down menu under the Auditing section of the window, and then select UPDATE from the right-hand drop-down menu. Then click **Add** to include the audit in your profile.

For both sections of the Wizard window, you can remove a security Category or an Audit action by highlighting the item to remove, then clicking **Remove**.

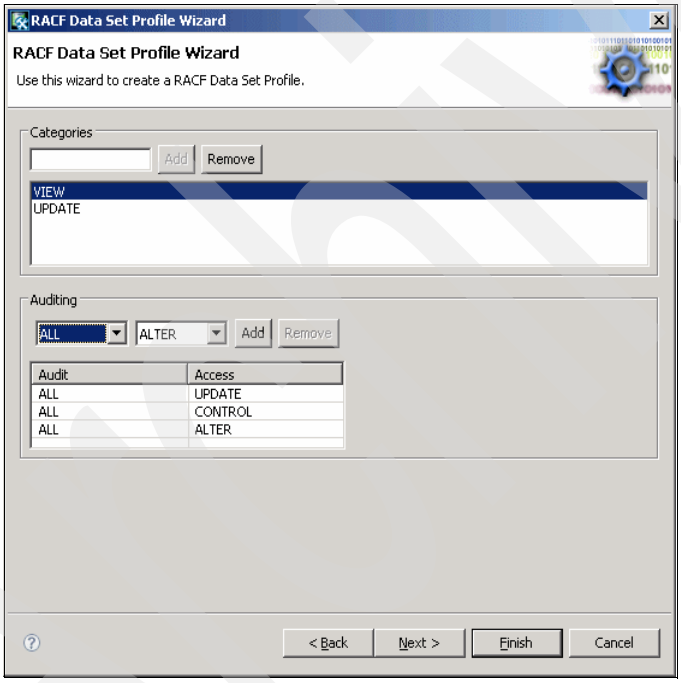


Figure 24-95 Specify the auditing parameters for the RACF Profile

When you have set all options you need, click **Next** to proceed to the last panel in the RACF Data Set Profile Wizard, as shown in Figure 24-96. The last Wizard panel lets you define an access list. On this window, you can provide the following information.

Under the area, Standard Access List:

- Specify the specific **user ID** and **access** that the user can have for the resources protected by the profile.

Under the area, Conditional Access List:

- Select the box next to **Conditional Access**.
- Specify the **User ID** or group ID and the type of **access** to grant.
- Specify the **Key** and **Value** combination to allow a more granular access to the protected resource..

The Conditional Access List is an additional set of criteria that must be met in order for the specified user ID or group ID to access the resource protected by the RACF profile. In the example shown in Figure 24-96 on page 659, the user MDSTIT has READ access to the specified resource in the profile but only if program AUZCPYCB is executing. User MDDECRB has UPDATE access to the resource with no additional conditions.

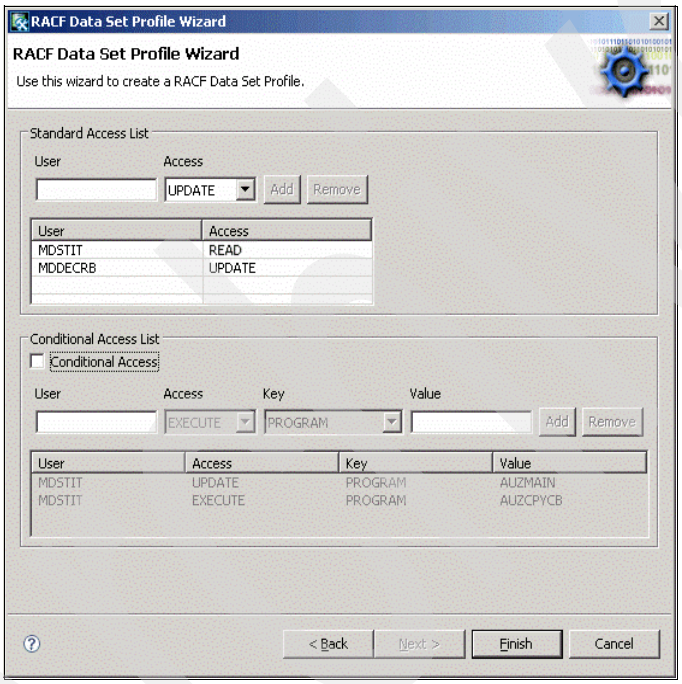


Figure 24-96 Specify the access granted to the RACF profile

When you have completed setting all options for the new RACF Data Set Profile, click the **Finish** button. You should now see your new RACF profile in the list in the Remote Systems View as shown in Figure 24-97.

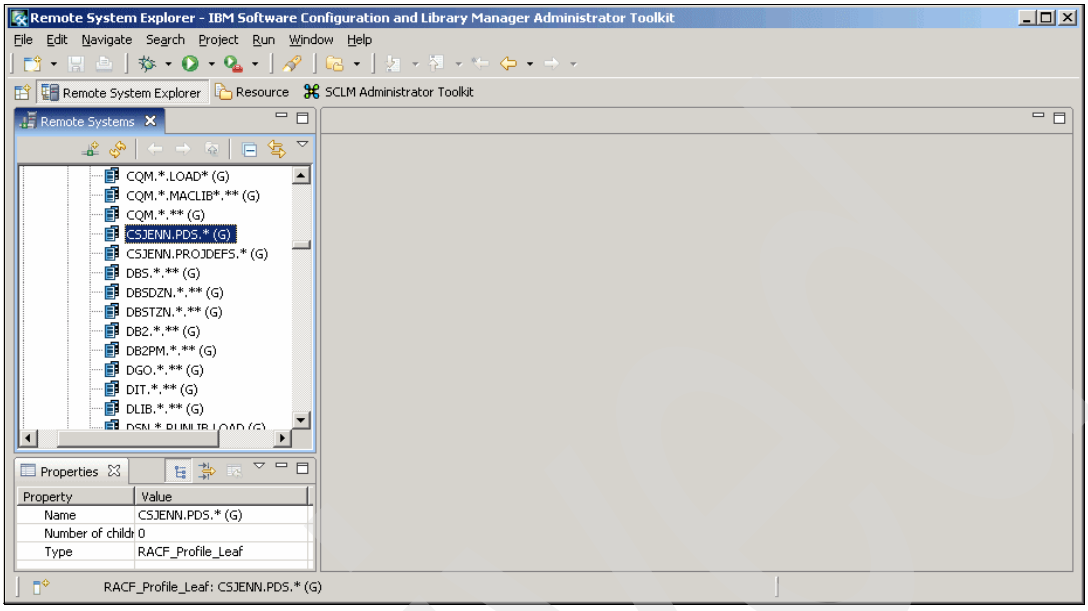


Figure 24-97 The new RACF profile appears in the list

Archive

25

Beyond the basics: A case study

On occasion, a business requirement can arise that necessitates creating a new SCLM project to mirror an existing project. However, the new project might have to differ slightly to accommodate changes that will be required in the future. For example, a team of developers might be shifting development responsibilities in anticipation of additional developers joining the team. Or perhaps another company's application source code has been acquired through a corporate merger, and the two sets of application code must now be maintained in the same SCLM project.

In this chapter we step through a scenario in which an existing SCLM project is cloned to create a new, separate SCLM project. The new project is then be tailored to suit the needs of the changing application source code by editing the project hierarchy and the project's application source code data sets. New artifacts are then migrated into the project and architecture definitions are created for the new artifacts.

The Admin Toolkit facilitates creating a new development group and the corresponding Group/Type data sets, as well as copying the application source code. When artifacts are migrated into the new project, the Administrator Toolkit automatically updates the accounting and auditing files with information about the new artifacts.

We cover the following tasks:

- Cloning an existing project
- Editing the new project
- Migrating artifacts into the new project
- Creating architecture definitions for the new artifacts

© Copyright IBM Corp. 2007. All rights reserved.

25.1 Cloning an existing project

In most circumstances, an existing SCLM-managed project can either be copied (copying just the project definition) or cloned (copying the project definition and the application source code) to create a new project without having to create one from scratch.

The workstation-based client will be used to illustrate the tasks necessary to complete the case study.

Begin by opening the project you want to clone in the Remote Systems View in the client. Either double-click the project name, or right-click the project to get an additional pop-up menu. Then select **Open Project**, as shown in Figure 25-1.

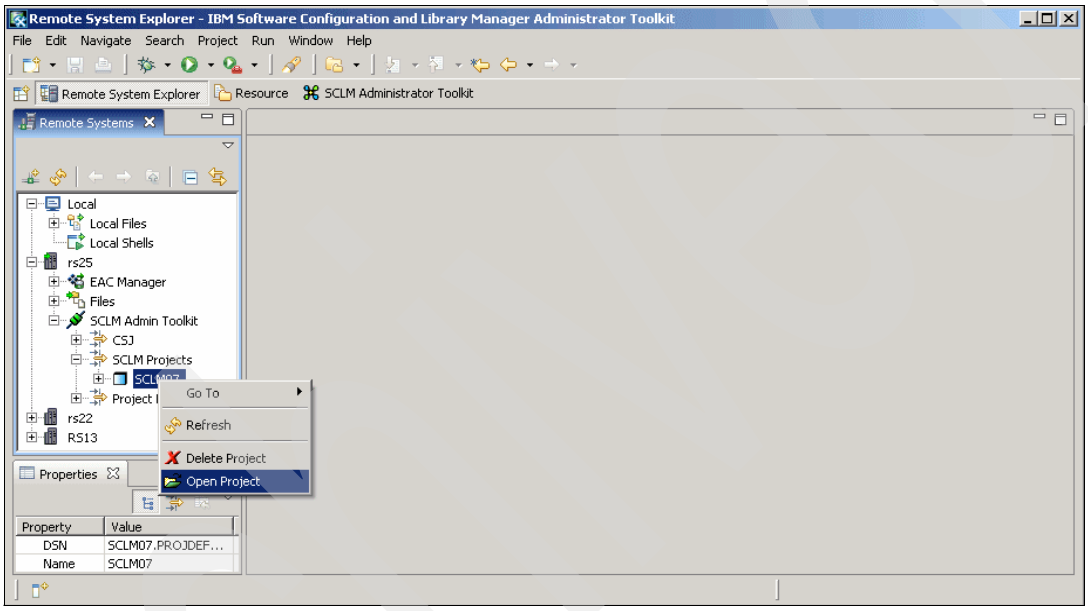


Figure 25-1 Open the project you want to clone

Select **SCLM** from the menu bar above, then click **Clone Project**, as shown in Figure 25-2.

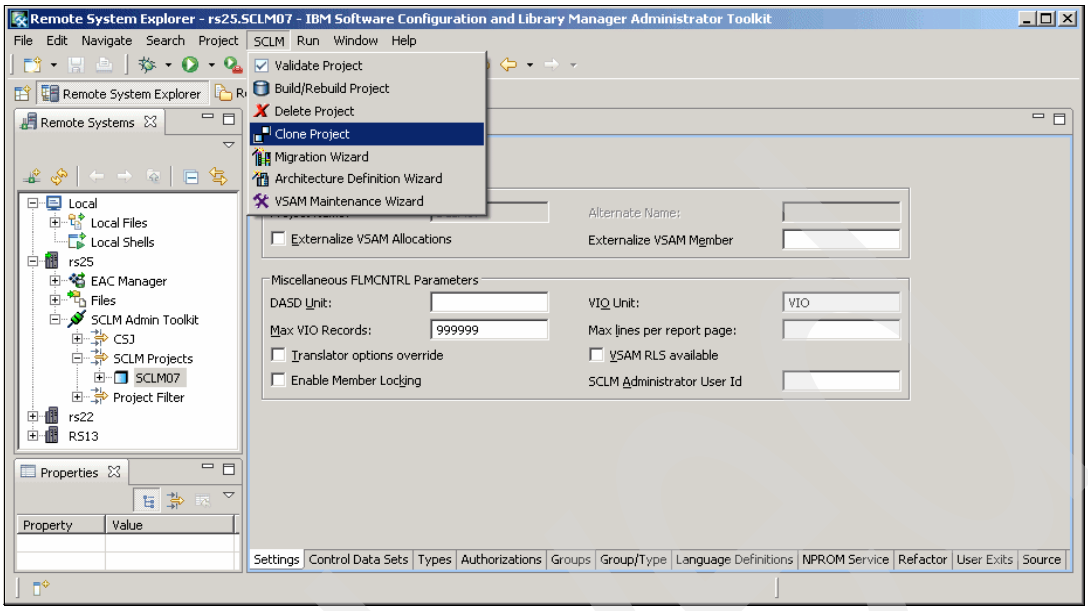


Figure 25-2 Select Clone Project

When the Clone Project Utility is selected from the SCLM pull-down menu, a pop-up window, Clone SCLM Project, as shown in Figure 25-3, allows you to specify the following options:

- The field, **Cloning Project Name**, is the name of the project you are making a copy of. In our example, we copy project SCLM07.
- The field, **New Project Name**, is the name of the project to be created. In our example, we create project CSJENNA and copy everything from SCLM07 into CSJENNA.
- The field, **Is the new project an alternate?** is asking if the new project, CSJENNA, is an alternate of the base project, SCLM07.
- The field, **Include project data?** is asking if you want to copy the application source data for project SCLM07 into project CSJENNA.
- The field, **New Alternate Name**, is only applicable if the option, Is The New Project an Alternate? is selected. Otherwise, it is ignored.

By selecting the Include Project Data option, you are permitting the Admin Toolkit to copy all Group/Type data sets and their source code from SCLM07 to CSJENNA so that an exact copy of the application source code being developed in project SCLM07 will exist in project CSJENNA. This allows you to begin development effort on a new application using the source code that is already available. If the option is not selected, then only the project definition data sets (<project>.PROJDEFS.*) from project SCLM07 are copied.

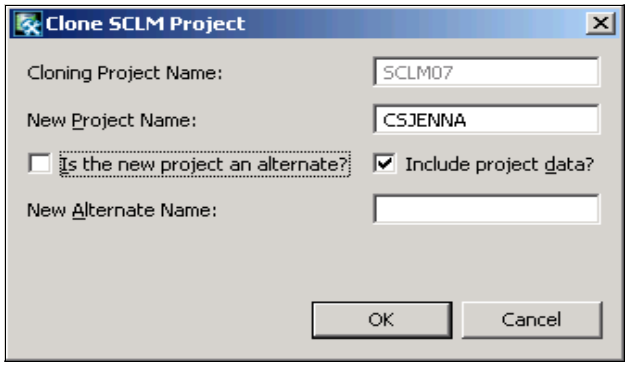


Figure 25-3 Specify a new project name, and copy the project data

Once you have clicked **OK**, the SCLM Admin Toolkit processes your request by copying all SCLM07 project data sets and application source code data sets, and creates data sets beginning with CSJENNA to complete the new project.

Figure 25-4 shows the new project CSJENNA in the Remote Systems View frame and opens the project for you to review in the Project Editor frame. Notice that project SCLM07 is still open. Now that we are done using project SCLM07, close that project in the Project Editor frame by clicking the **X** in the upper right-hand corner of the Project window.

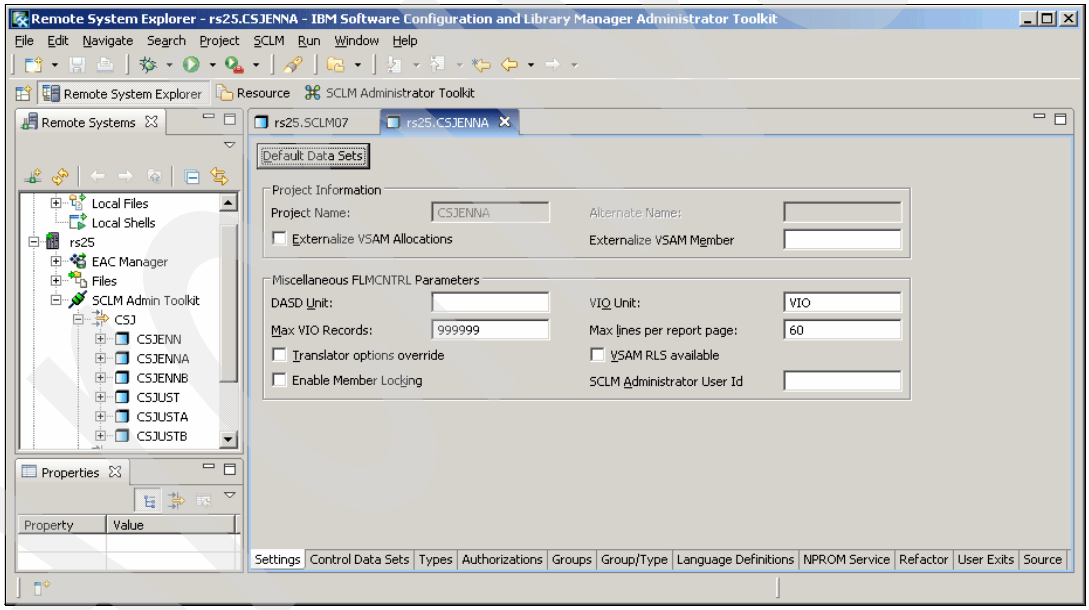


Figure 25-4 The new project is displayed in the Project Editor frame

25.2 Editing the new project

Now that project CSJENNA has been created by cloning project SCLM07, you can begin editing the project to suit your needs. SCLM provides a controlled environment to maintain and track all software components.

25.2.1 Changing project definition and control information

The project definition data sets define the environment of the project and are specified on the Settings tab in the Project Editor. The data set names are defaulted to those of the project that was cloned. Should you need to add an additional PROJDEFS data set, it can be specified on the Settings tab and the Admin Toolkit will create it when the project is built.

Control data sets are used to track and control application source code within the project hierarchy. SCLM stores accounting and audit information into data sets that are defined in the project control information. The Control Data Sets tab provides you with the ability to edit the project's control information.

Figure 25-5 shows the Control Data Sets tab which allows you to provide the primary control information that is to be used for the project, and optionally, alternate control information. Alternate control information can be specified for individual groups to override certain options specified in the primary control information.

To view the primary control information that was copied as a result of cloning another project, select the radio button next to the option Primary Control Information, and then click the **Edit** button.

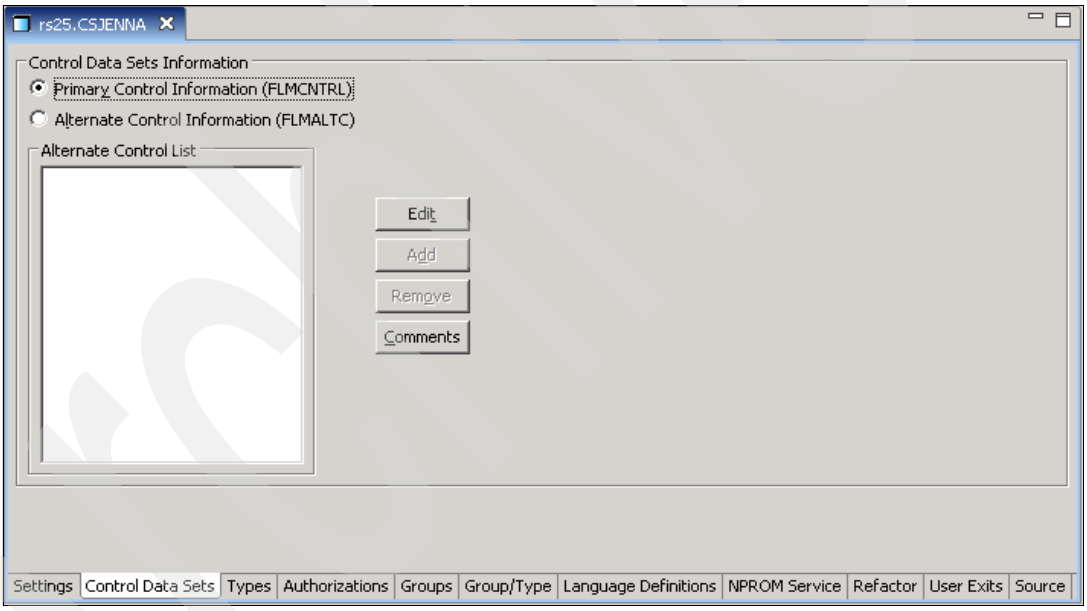


Figure 25-5 The Control Data Sets tab allows you to edit Accounting and Auditing files

The Accounting Dialog window in Figure 25-6 displays the primary control options already established for the project. In this window, you can specify the following information:

- The **Name** field defaults to Primary if this is the primary control information.
- The **Data Set Name field** is the pattern to use for the application source code data sets when versioning is used.
- The **Primary Accounting** data set is the name of the accounting file that will track the activity for the application source code. It contains information such as statistics, dependency information and build maps.
- The **Secondary Accounting** field is the backup accounting data set that may be used should something happen to the primary file.

- The **Export Accounting** field is the name of the data set that contains information that has been exported from the accounting data set.
- The **Control Data Set** contains information about the project's PDS members should they need to be backed up.
- The **Primary Auditing** data set is needed when SCLM's auditing capability will be used to track actions on project members
- The **Secondary Auditing** file is a backup data set that may be used should something happen to the primary auditing file.
- The **Version Data set** field specifies a pattern to use for the PDS data sets that are to have changes tracked.
- The **Version to Retain** field specifies the number of PDS data sets to retain.

When all options have been specified, click **OK**.

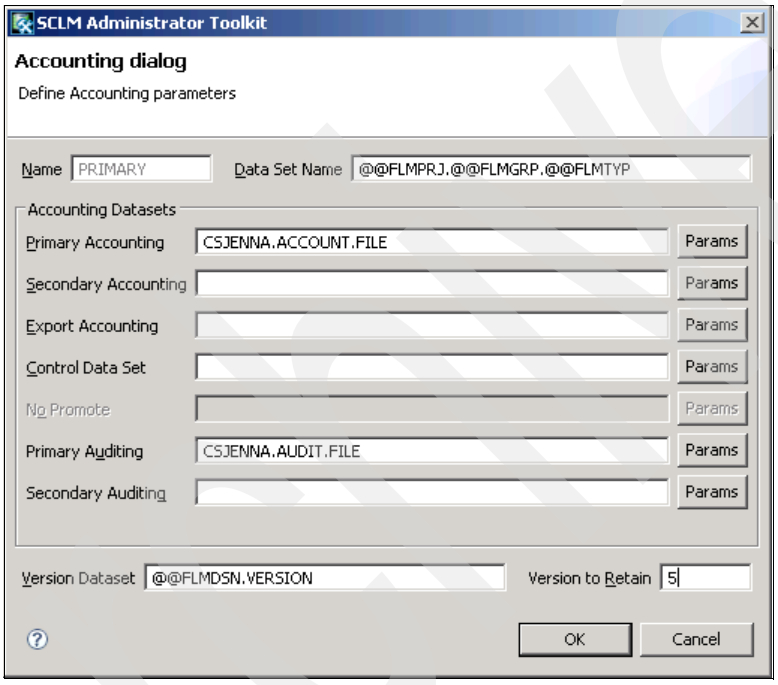


Figure 25-6 Accounting and auditing data sets track project activity

To add alternate control information, select the radio button next to the option Alternate Control Information on the Control Data Sets tab, and then click the **Add** button. The Accounting Dialog window in Figure 25-6 displays a blank window for you to provide the alternate control options.

25.2.2 Adding a new project group

The cloned project's group hierarchy can be changed to reflect the needs of the new project. A project hierarchy is a structural network that maps the relationship between the groups defined in the hierarchy and defines how data navigates up and down the hierarchy.

A group within a hierarchy contains permissions for accessing specific application source code and authorizations that define what actions can occur on source code as it is promoted to the next higher group in the hierarchy.

Adding a new group to a hierarchy can affect several other groups within the project hierarchy, because the structural promotion path is affected. Each group affected by the newly-added group must be edited to re-establish the promotion path, and permissions must be altered to accommodate the addition of a new group to the project.

The Admin Toolkit can simply add a new group to an existing project hierarchy because it automatically updates the affected groups when a new group is added. To add a new group to a project, select the Groups tab in the Project Editor.

The project hierarchy appears as a series of linked boxes with the arrows indicating the direction of the hierarchy. To add a new group, enter the new group name in the field Group Name at the top of the frame, then click the **Add** button. In this example, the group DEVTEST is being added as shown in Figure 25-7.

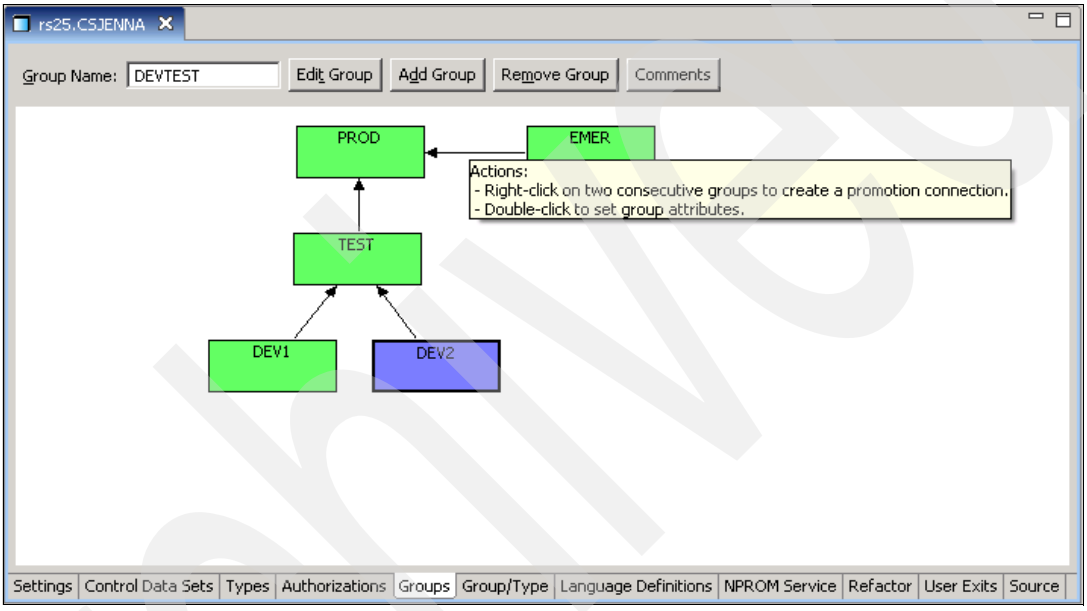


Figure 25-7 The existing project hierarchy was cloned from another project and may be edited

Once the **Add** button is clicked, the new group will appear in the tab highlighted in blue. To insert the new group into the hierarchy, right-click the new group box, then right-click the group box you want to be the parent, and a directional arrow will appear between the new group and the parent group you selected.

In our example, the group DEVTEST is going to be an integration test group for the development groups and will promote to the group TEST, as shown in Figure 25-8.

To establish the hierarchy for the development groups to promote to the new group DEVTEST, click a lower-level development group so that the group box turns blue. Right-click the development group, then right-click **DEVTEST**. The directional arrow will appear between the development group and the group DEVTEST.

The new project hierarchy is displayed in Figure 25-8.

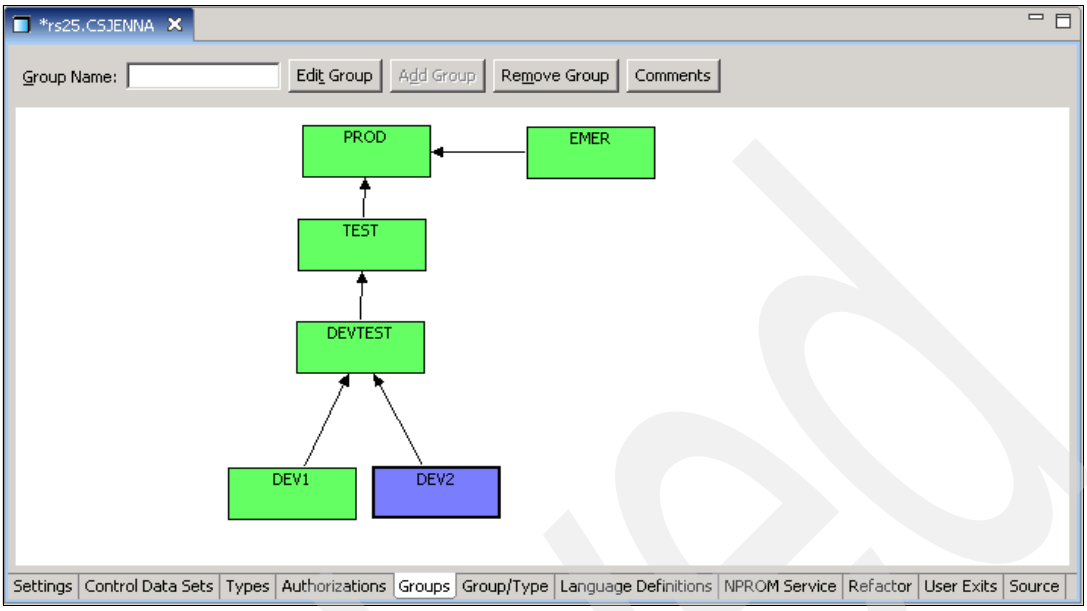


Figure 25-8 The new project hierarchy is displayed

To edit the parameters defined for a group, either double-click the group box, or click a group box so that it is highlighted in blue, then click the **Edit Group** button. The Group Dialog box is displayed. In the Group Dialog box, you can specify the following parameters:

- The **Group** name cannot be changed because you are editing this group, not creating it.
- The **Promotes to Group** option tells SCLM where the project is in the hierarchy. It specifies which group will receive the code from the selected group when promotes or copies are done.
- The **Alternate Control Options** are those project control options the group is to use in place of the Primary Control Options.
- The **Backout Groups** name specifies the package backout group defined in the project.
- The **Key** check box indicates that data moved to this group is erased from its previous group when it is promoted. If this box is not checked, then data moved to this group is copied upon promotion.
- The **Member Restore** check box indicates SCLM will restore individual members rather than a package
- The **Authorizations** section of the Group Dialog box allows you to Add Auth Codes to the group. The Available Auth Groups are listed in the left-hand frame, and the Selected Auth Codes are listed in the right-hand frame and will be used for this group in the project.

When you have completed editing parameters for this group, click **OK**, as shown in Figure 25-9.

Archived

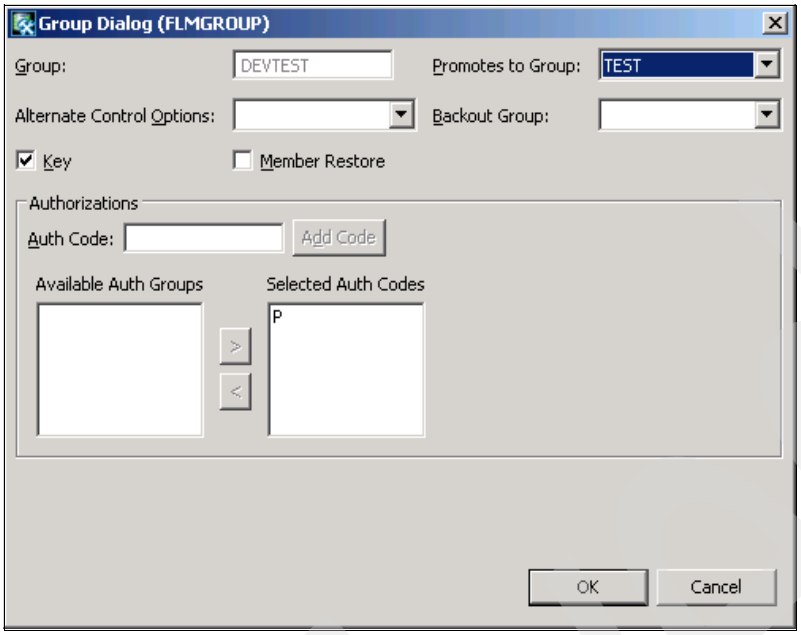


Figure 25-9 These are the defaulted options for the new group in the hierarchy

25.2.3 Adding new Group/Type data sets

Select the **Group/Type** tab in the Project Editor to select the group/type data sets to include in the project for the new group that was just added.

In the left-hand frame of the Group/Type tab are the available Group/Type Combinations to choose from in alphabetical order. Notice that the available data sets are those that are already defined to the project. The group that was added to the project was called DEVTEST. Scroll down past the DEV1 and DEV2 combination data sets and you will see DEVTEST combination data sets available to choose from, as shown in Figure 25-10.

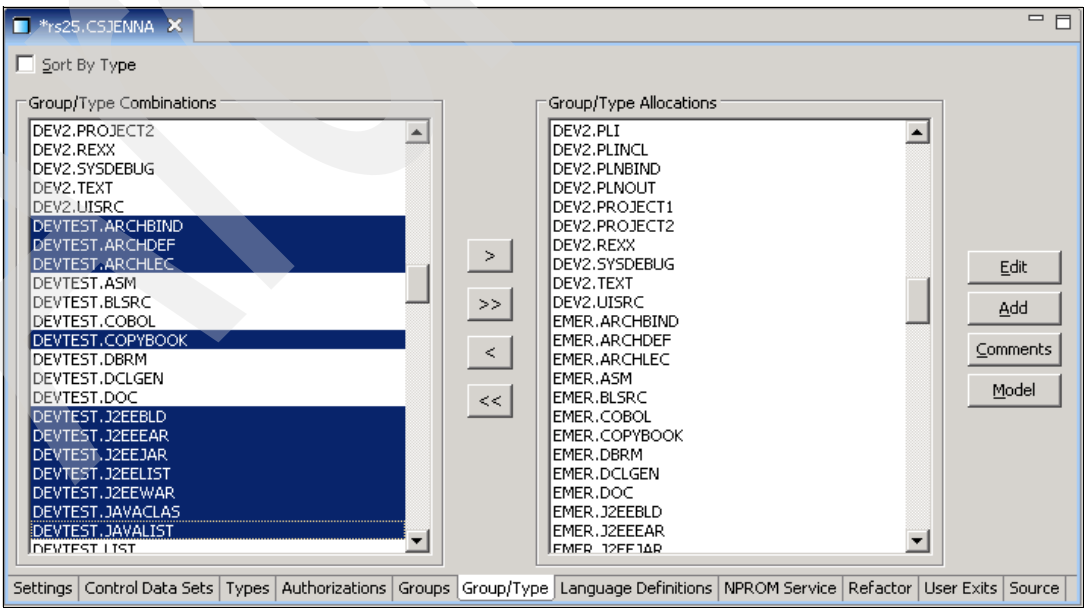


Figure 25-10 Select the data set group/type combinations to include for the new group

Select the data set group/type combinations from the left-hand frame that you want to include in the project for the new group. You can select multiple data sets by holding down the Control key and clicking the data sets you want.

When you have selected all the data sets you want to create, click the single right-hand arrow (**>**) to move the selected data sets to the right-hand frame, **Group/Type Allocations**. You will see the selected data sets in the right-hand frame. The Admin Toolkit will automatically create the new group/type data sets when the project source definition is updated.

25.2.4 Changing the language definitions

The existing language definitions are those that were cloned from project SCLM07. The language definitions that are in the SCLM macro library ISP.SISPMACS are default languages supplied by SCLM and should not need to be edited. However, the languages that were copied into the <project>.PROJDEFS.SOURCE data set were customized for project SCLM07 and should be edited to ensure any data set are changed.

To edit existing language definitions, open the Language Definitions tab and highlight a language you want to edit. Then click the **Edit** button to the right-hand side of the project Editor frame, as shown in Figure 25-11.

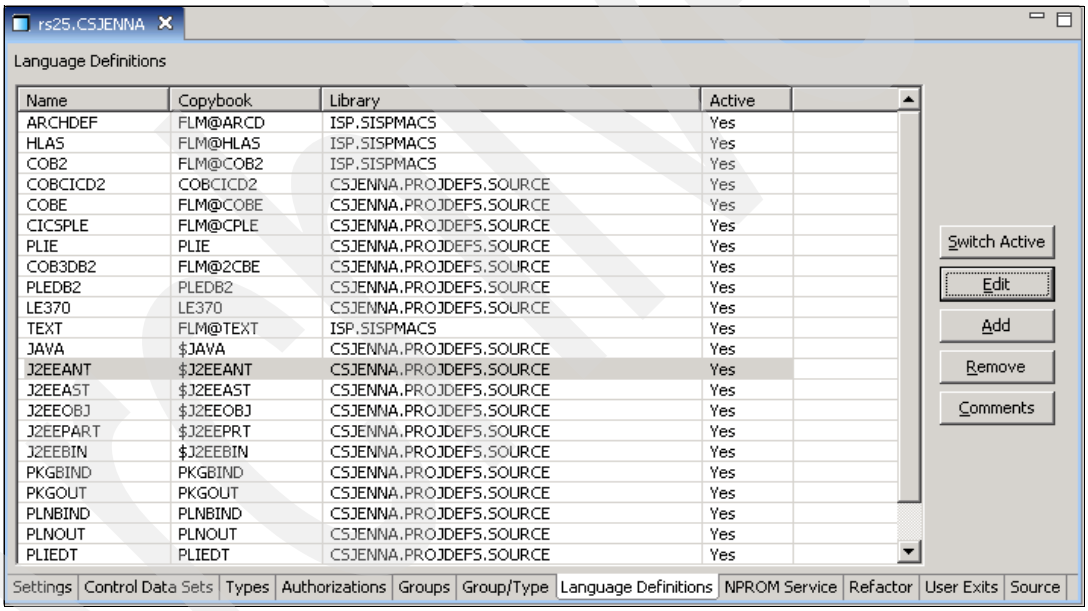


Figure 25-11 Select a language definition to edit

The Language Definition Wizard window appears allowing you to make any necessary edits. In this example, we must change the data set specifications in the Translator Call Name $J2EEANT. Click **Next** until the Translator Summary is displayed. On the Translator Summary panel, shown in Figure 25-12, highlight the Translator Call Name $J2EEANT, and click the **Edit** button.

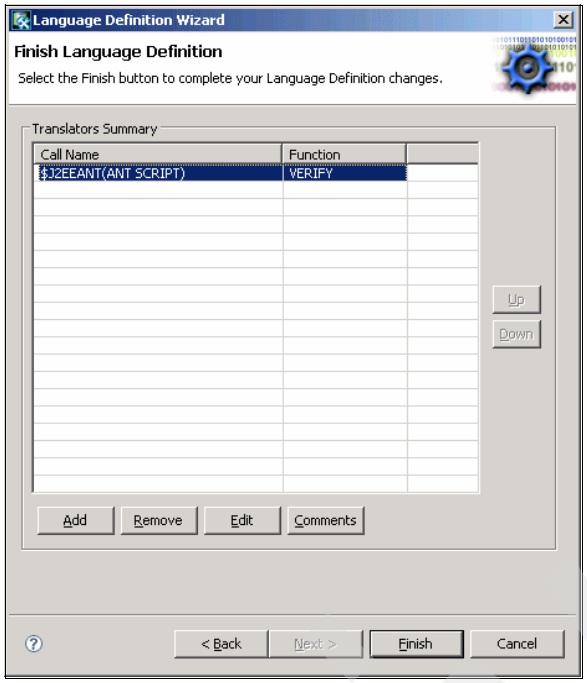


Figure 25-12 Highlight the Translator to edit

On the Translator Parameter window, the field Data Set Name must have its value changed from the value 'SCLM07.TEST.LOAD' to 'CSJENNA.DEVTEST.LOAD' as shown in Figure 25-13 and Figure 25-14.

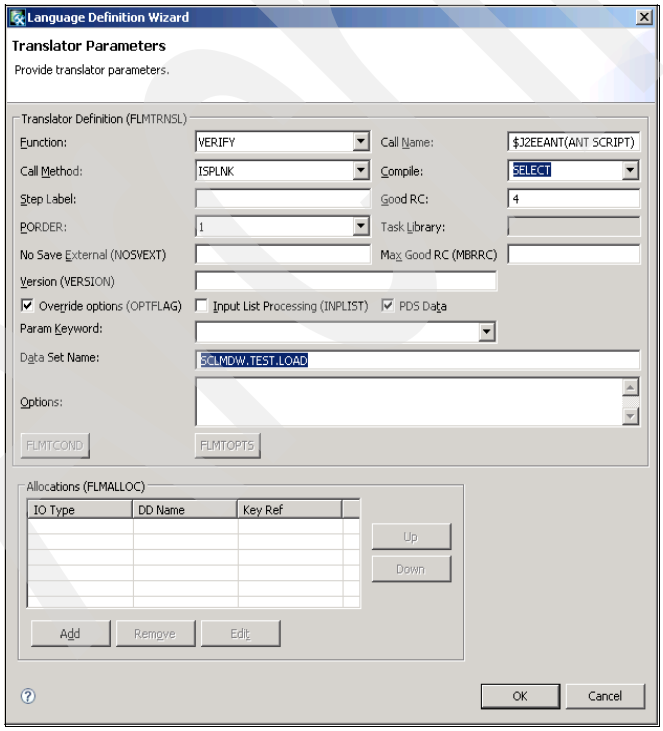


Figure 25-13 Change the field Data Set Name to reflect the new correct

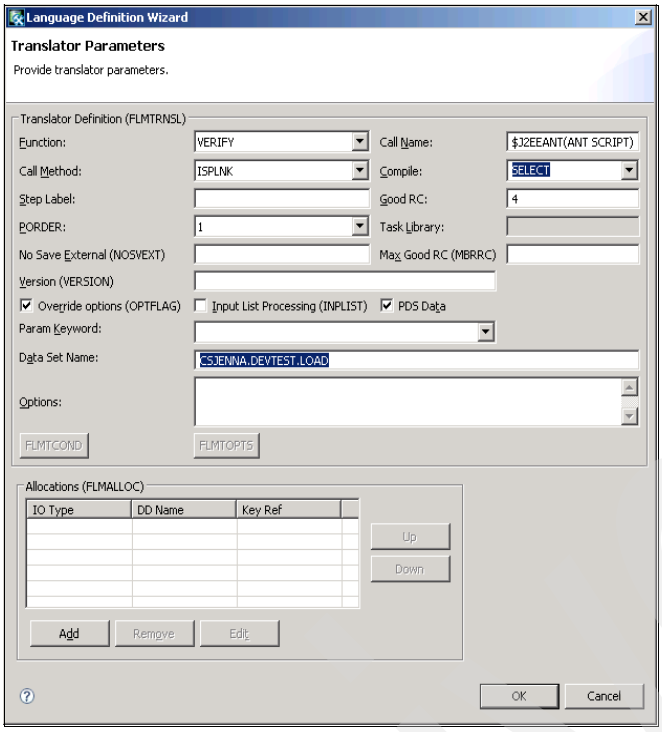


Figure 25-14 Change the Data Set Name to the new source code data set

When the Language Definition window, Finish Language Definition, is displayed, click the **Finish** button exit from the Language Definition Wizard. Each of the Language Definitions that reside in <project>.PROJDEFS.SOURCE data set must be checked to ensure that the customizations made for project SCLM07 are changed for new project CSJENNA.

25.2.5 Changing the user exit information

Once all Language Definitions have been customized for the new project CSJENNA, the User Exits and their corresponding parameters must also be changed. Just like Language Definitions, User Exits are tailored to a project's specific needs and must be edited when they are cloned from another project.

For this example, select the **User Exit** tab and highlight the first User Exit in the right-hand frame of the Project Editor, titled Selected User Exits. Then click the **Edit** button, as shown in Figure 25-15.

Archived

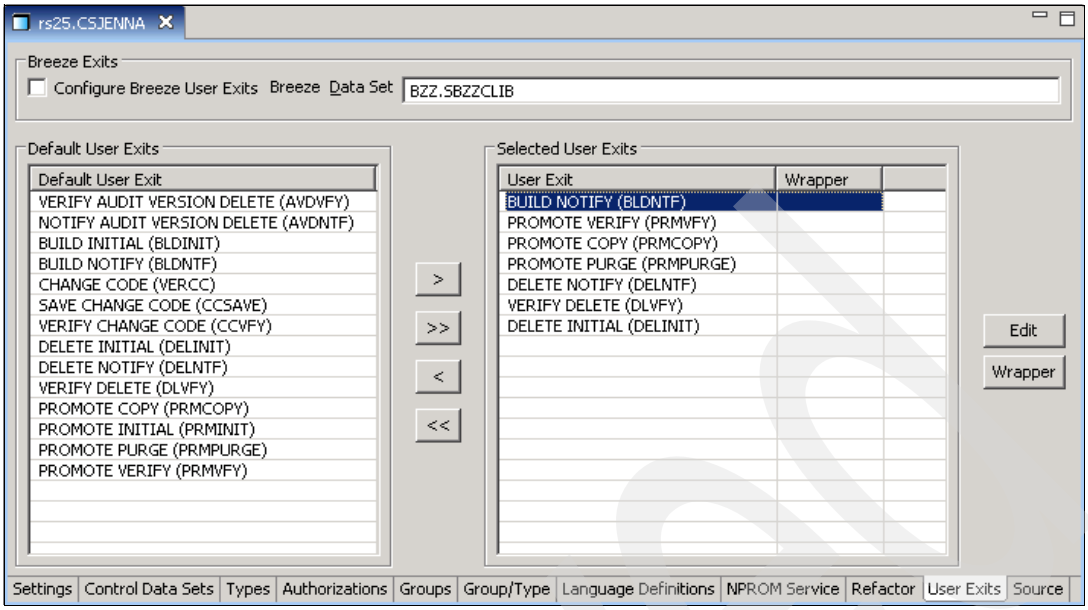


Figure 25-15 Select the User Exit to edit

A window with the title of the User Exit will appear, as shown in Figure 25-16, allowing you to change the parameters defined in the User Exit. In this example, the Load Data Set field contains a value that refers to the project SCLM07 that was cloned.

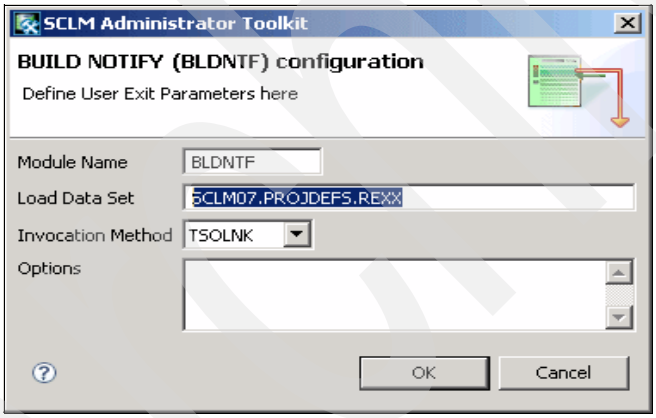


Figure 25-16 Change any values that do not apply to the new project

Change the data set name to reflect the new project's REXX data set, the click **OK**, as shown in Figure 25-17.

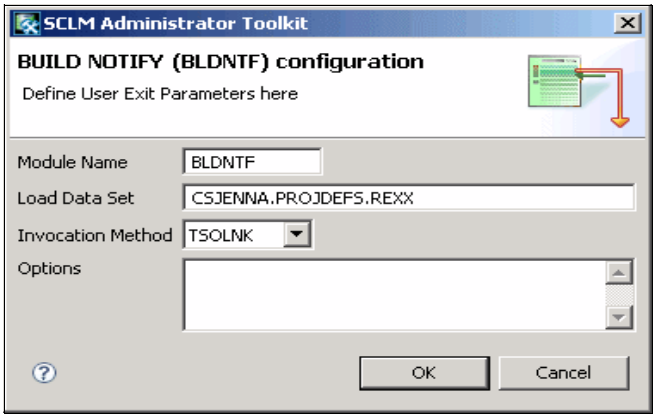


Figure 25-17 The user exit tailoring is complete

Continue editing the remaining user exits to tailor the parameters for the new project.

25.2.6 Building the project

When all edits have been completed, the project must then be built. Building a project means that all of the options that were changed within the project and saved in the PDML, or Project Definition Markup Language, are written to the <project>.PROJDEFS.SOURCE(<project>) data set. When a new SOURCE member is written, the edits that were made take effect and base SCLM uses the project with the changes that were made.

To save the changes to the new project and to build the project, select **File** from the menu bar above, then select **Save**. Or, you can click the disk icon on the button bar. Once the project has been saved, it must be built.

Select **SCLM** from the menu bar above, then select **Build/Rebuild Project**, as shown in Figure 25-18.

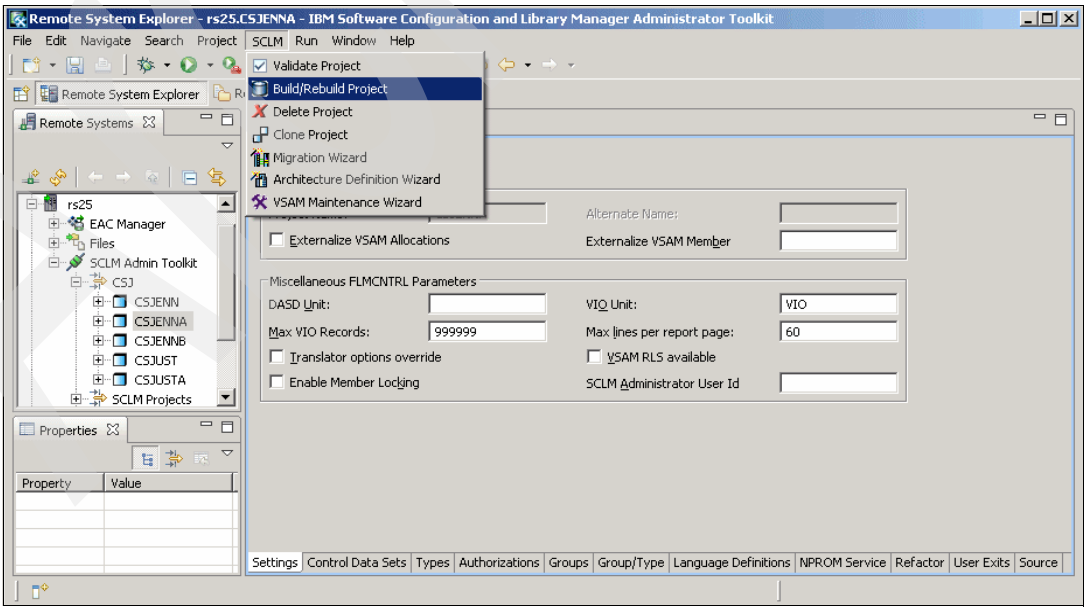


Figure 25-18 Select Build/Rebuild Project to create new members in <project>.PROJDEFS.SOURCE

Behind the build

When a project is built in the Admin Toolkit, all options that were edited or changed via the Admin Toolkit user interface are written to the members in data set <project>.PROJDEFS.SOURCE.

Figure 25-19 helps to illustrate what happens during the build process. The options and parameters that were modified in the project are updated in the project's PDML found in the <project>.PROJDEFS.SETTINGS data set. When you are ready for those changes to be updated in the project source, the build process reads the PDML and writes a new SOURCE member to the data set <project>.PROJDEFS.SOURCE.

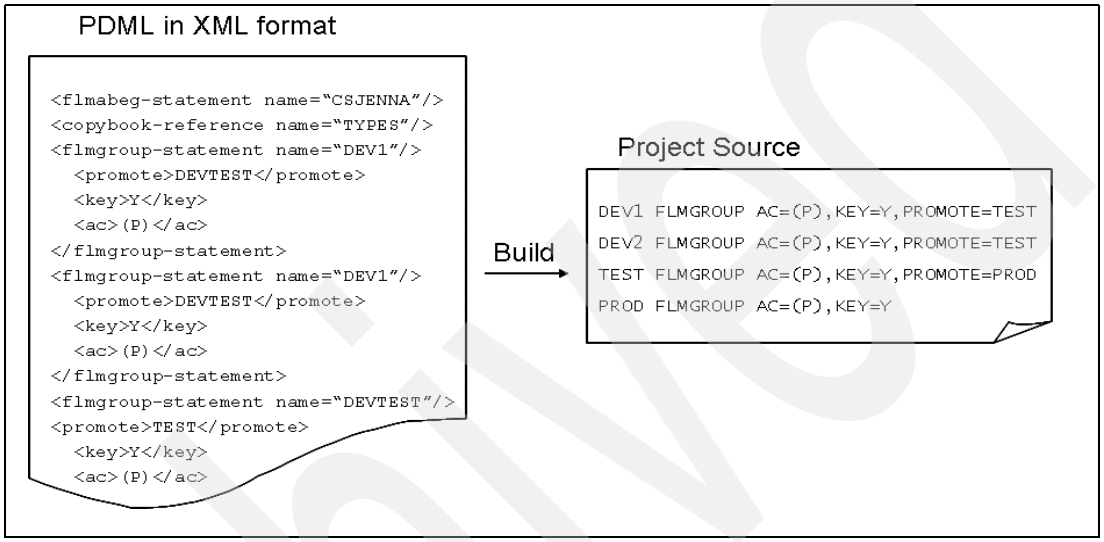


Figure 25-19 The Project Definition Markup Language updates the project source

For example, in the project data set CSJENNA.PROJDEFS.SOURCE(CSJENNA), there is a section that defines the GROUPS in the project. The PDML reflects the change we made to accommodate the new group DEVTEST that was added after cloning project SCLM07. After the build is complete, the project's SOURCE data set contains an updated CSJENNA member with the new group DEVTEST, as shown in Figure 25-20.

Archived

```
CSJENNA  FLMABEG
*
*        ************************************************************
*        *  DEFINE THE TYPES                                        *
*        ************************************************************
*
         COPY  TYPES
*
*        ************************************************************
*        *  DEFINE THE GROUPS                                       *
*        ************************************************************
*
DEV1     FLMGROUP                                                   X
               PROMOTE=DEVTEST,                                     X
               KEY=Y,                                               X
               AC=(P)
DEV2     FLMGROUP                                                   X
               PROMOTE=DEVTEST,                                     X
               KEY=Y,                                               X
               AC=(P)
DEVTEST  FLMGROUP                                                   X
               PROMOTE=TEST,                                        X
               KEY=Y,                                               X
               AC=(P)
TEST     FLMGROUP                                                   X
               PROMOTE=PROD,                                        X
               KEY=Y,                                               X
               AC=(P)
PROD     FLMGROUP                                                   X
               KEY=Y,                                               X
               AC=(P)
EMER     FLMGROUP                                                   X
               PROMOTE=PROD,                                        X
               KEY=Y,                                               X
               AC=(EMER)
....
```

Figure 25-20 The updated CSJENNA source member now contains the new group DEVTEST

Archived

25.3 Migrating artifacts into the new project

Once the project has been built, you can then migrate artifacts into the project from either remote sources or from sources on the host. Both the Migration Wizard and the Remote Migration Wizard will be demonstrated to illustrate how to migrate artifacts from a z/OS host and from a remote system.

Migrating an artifact from a z/OS host into an SCLM-managed project on a z/OS host

To begin migrating artifacts from a z/OS host, select **SCLM** from the menu bar, then select **Migration Wizard**, as shown in Figure 25-21.

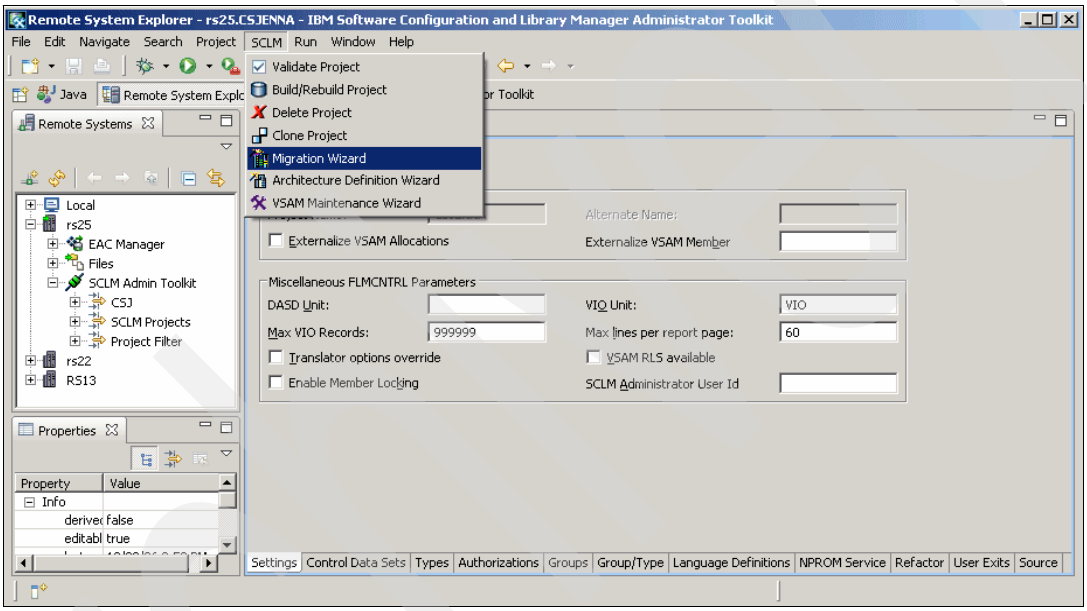


Figure 25-21 Select Migration Wizard to begin migrating an artifact into the open project

The initial Migration Wizard window is displayed, allowing you to add artifacts. Click the **Add** button to the right-hand side of the frame to begin adding artifacts into the Migration Wizard, as shown in Figure 25-22.

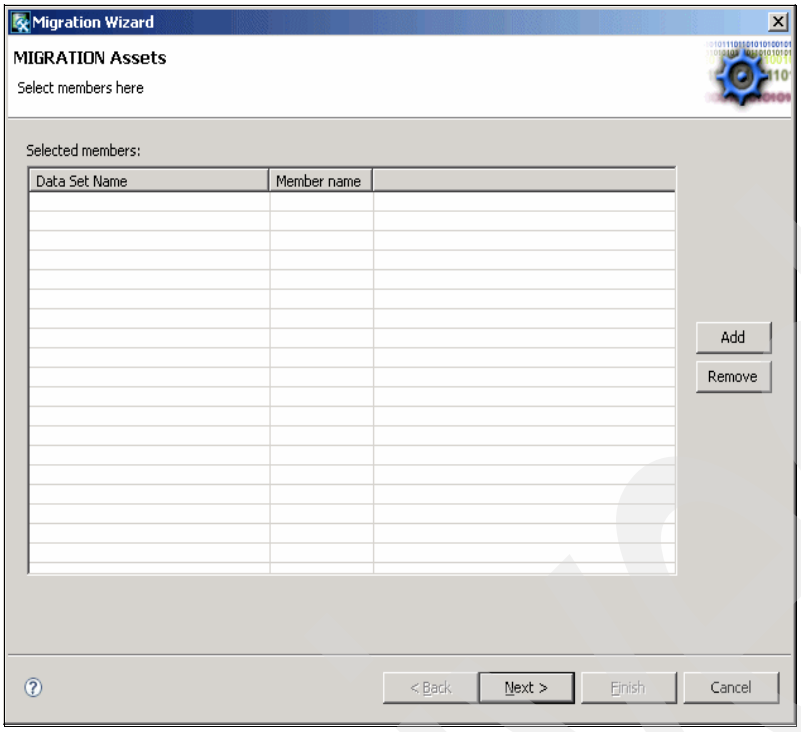


Figure 25-22 Select Add to specify artifacts to migrate into the open project

The Data set Selection Dialog window appears allowing you to specify the data set or data sets from which you want to migrate artifacts. The **Browse** button allows you to search for a partially qualified data set name. The Member Name frame contains the corresponding members from the selected data set. Highlight the members you want to migrate. You can select multiple members by holding down the Control key while clicking the member names.

Once you have selected the member or members you want to migrate, click **OK** to continue as shown in Figure 25-23.

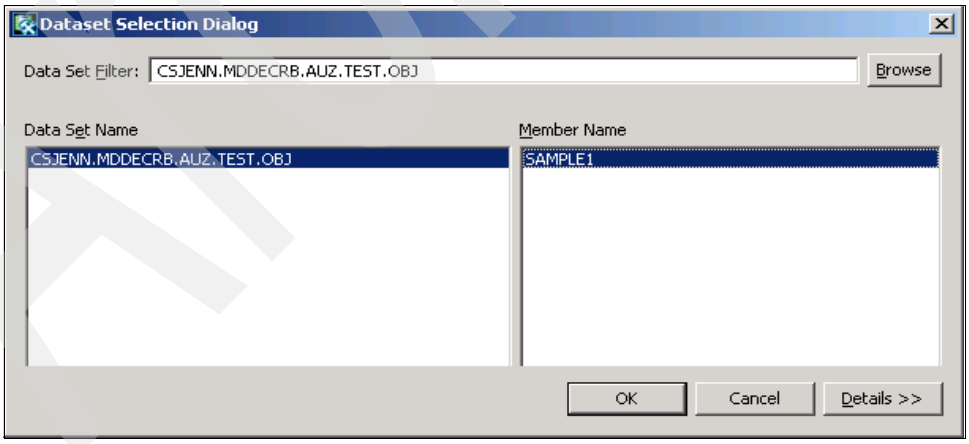


Figure 25-23 Select the member or members to migrate into the open project

When you have clicked **OK**, you are returned to the Migration Assets window with the specified data set and member name filled in. Click **Next** to continue, as shown in Figure 25-24.

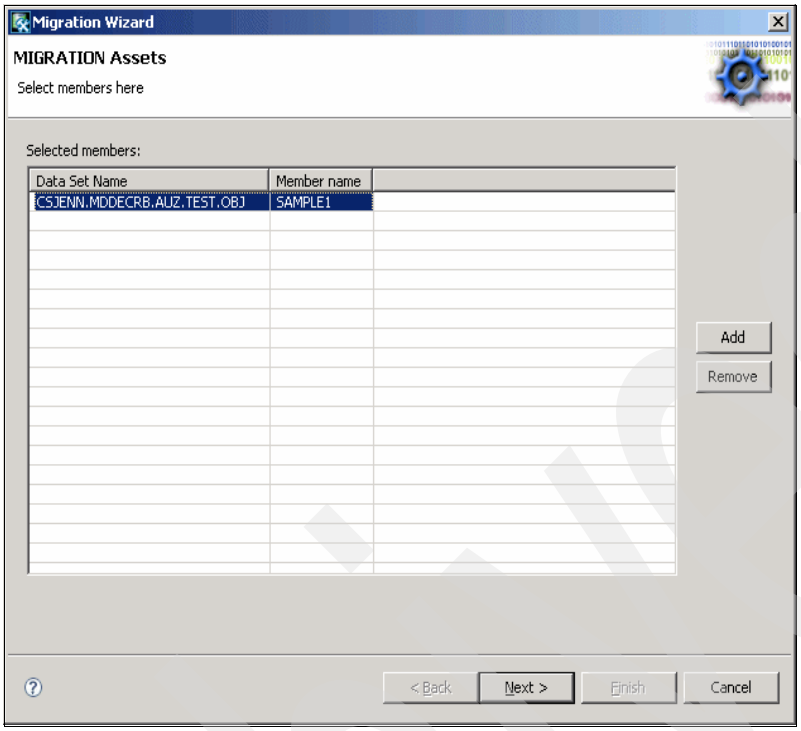


Figure 25-24 The selected artifacts are filled in the Migration Assets window

On the Migrate Asset Option window, specify the following information as shown in Figure 25-25:

- The **Group** to migrate artifacts into
- The data set **type** the members should be migrated into
- The **language** that SCLM is to use for the new artifacts being migrated into the project
- The **authorization code** that is to be used for the artifacts
- Optionally, the **change code** to use with the artifacts
- The **Mode** that is to be used to migrate the artifacts
- The **date and time** that is to be used to timestamp the artifacts

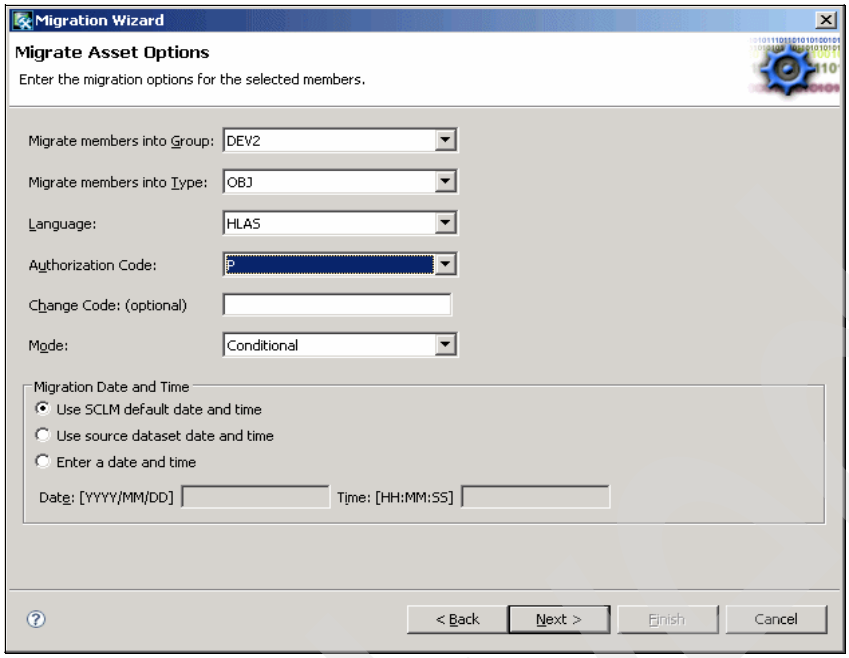


Figure 25-25 Specify the Migration parameters that are to be used for all artifacts

When all options have been set for the selected artifacts, click **Next** to begin the migration. When the migration has completed, you will receive a migration report as shown in Figure 25-26.

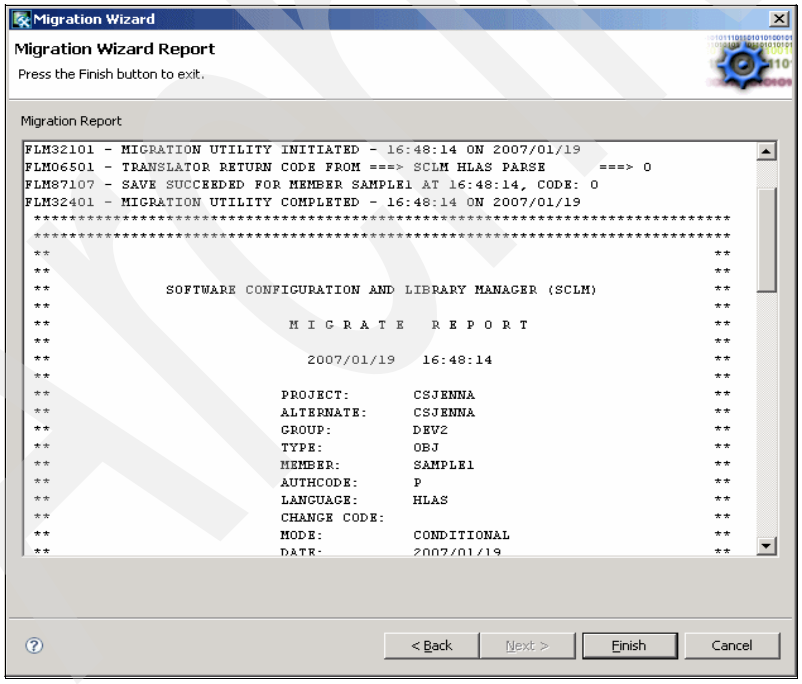


Figure 25-26 The migration report is returned for your review

The Migration Report window may be expanded to view more of the migration report. The scroll bars may also be used to view the complete report.

Migrating an artifact from a remote system into an SCLM-managed project on a z/OS host

To begin migrating artifacts from a z/OS host, select **Window** from the menu bar, then select **Open Perspective**, as shown in Figure 25-27.

Note: The Remote Migration Wizard is only available in the workstation-based user interface.

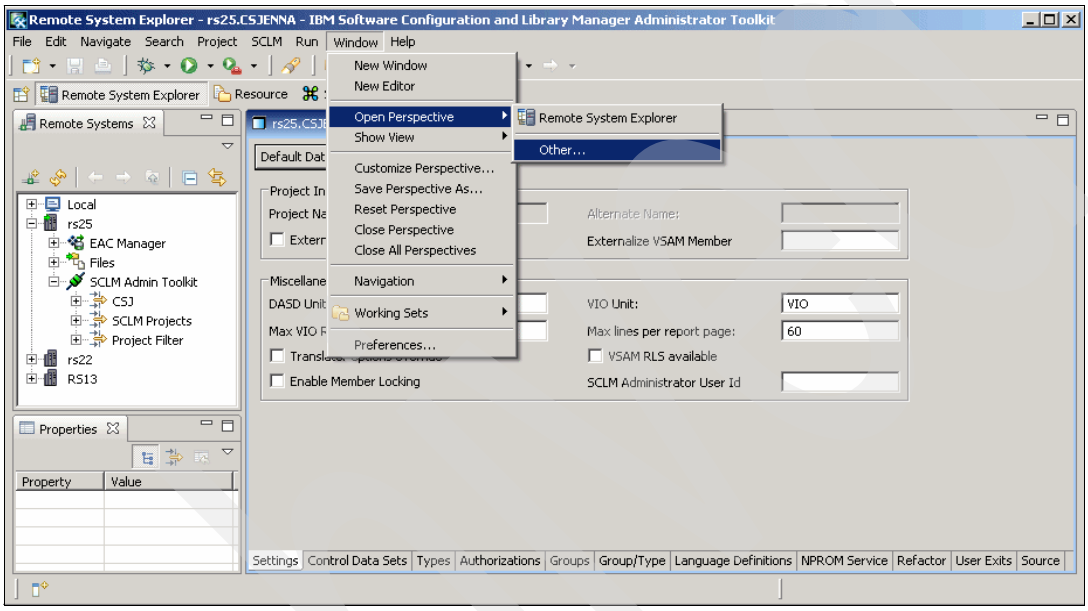


Figure 25-27 Navigate to the perspective that will allow you to access your remote assets

In this example, artifacts are being migrated from another Java project. Select the **Java** perspective and click **OK** as shown in Figure 25-28.

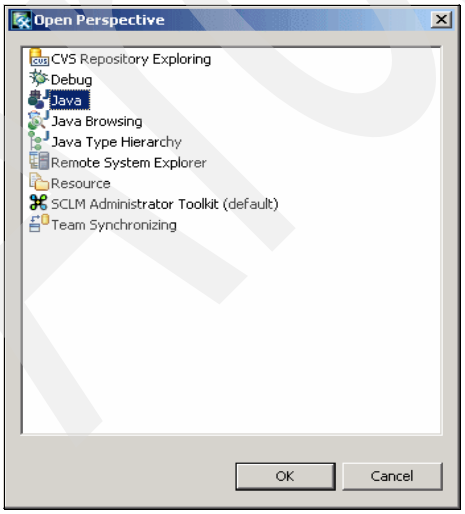


Figure 25-28 Select the perspective that will allow you to access your remote assets

The Resource perspective will open in place of the Remote Systems View perspective. Select the Java project you want to migrate artifacts from to expand it. Find the artifacts you want to migrate by holding the Control key and clicking the artifacts. Right-click the mouse button to see the additional menu appear. Then select **Team**. When Team is selected, the option **Migrate to SCLM** will be visible to select as shown in Figure 25-29.

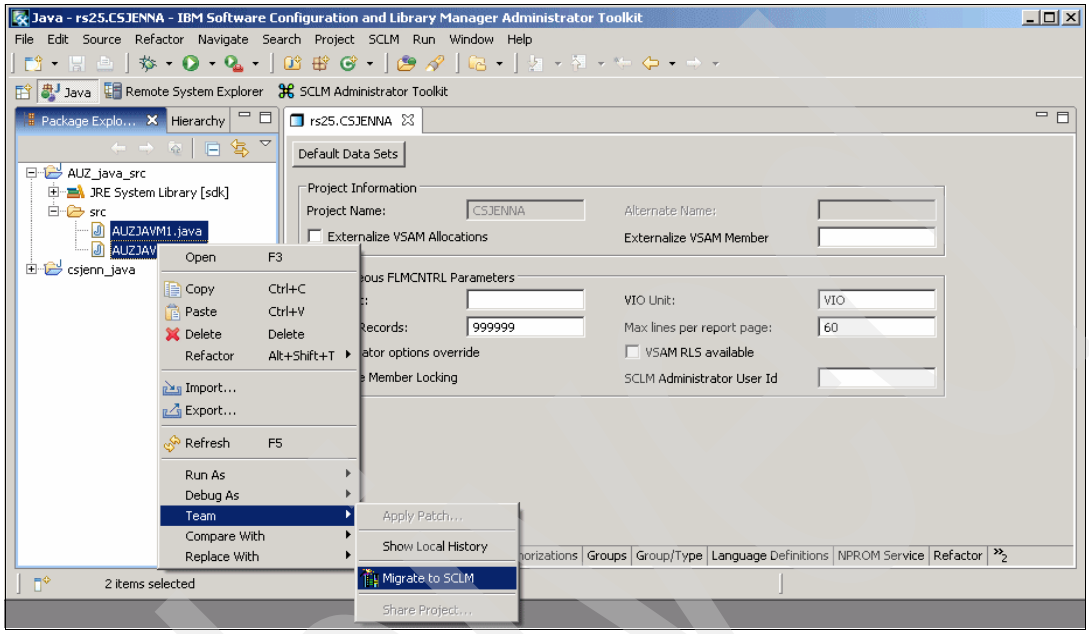


Figure 25-29 Highlight the artifacts to migrate, and right-click to get to the Migrate function

The SCLM Remote Migration Wizard window opens allowing you to select as many artifacts as you want to migrate. When you have finished, click **Next**.

When the Resource Selection window appears, highlight the artifacts you want to migrate. Click **Next** as shown in Figure 25-30.

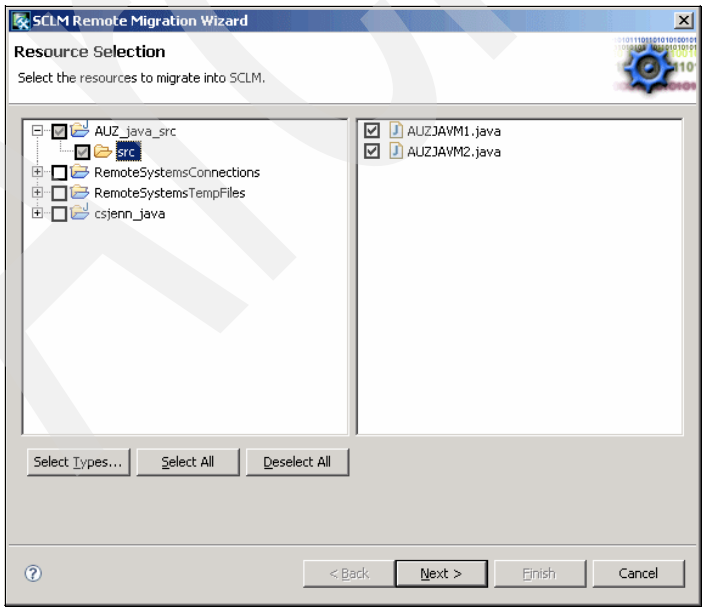


Figure 25-30 Select the remote artifacts to migrate

The window Migration Project opens asking you to select the host system you want to migrate your artifacts to. In addition, select the project name to migrate the artifacts into. Click **Next**. If you are not already logged into the host system you want to migrate artifacts into, the Administrator Toolkit will ask you to log in as shown in Figure 25-31.

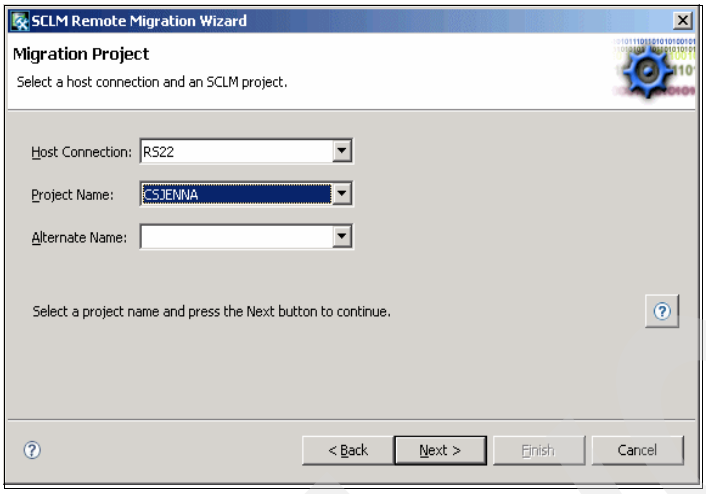


Figure 25-31 Log into the host system you want to migrate the artifacts into

The Migration Options window opens, and appears similar to the Migration Options window for migrating non-remote artifacts as seen in Figure 25-25 on page 680. Provide the following information as shown in Figure 25-32:

- The **Development Group** to migrate artifacts into
- The **authorization code** that is to be used for the artifacts
- Optionally, the **change code** to use with the artifacts
- The **Mode** that is to be used to migrate the artifacts
- The **date and time** that is to be used to timestamp the artifacts

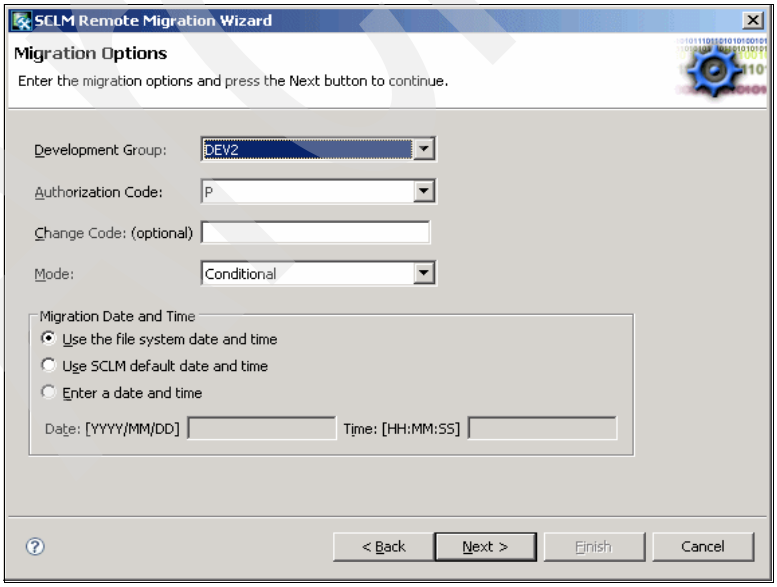


Figure 25-32 Specify the options to migrate the artifacts with

Click **Next** when you have provided all necessary parameters. The window Migration File Extension Mapping opens and will display all types of file extensions of those artifacts being migrated in. For example, if you chose to migrate a *.class file type and a *.html file type, they would be listed underneath the file type Java under the column File Extension.

Specify the SCLM Type of data set that the Java files should be migrated into. In addition, tell SCLM what language it should use for the artifact once it is in the project.

Note: If there is not a suitable language for the artifacts you are migrating in, you must first add a language to the project, then migrate the remote artifacts into the project.

The option Perform File Truncation Checking checks the length of the file name to ensure it adheres to the host system's requirement of keeping data set member names to eight characters or less. If an artifact is found to have a name longer than eight characters, the name is truncated and written to a translation table where its original file name is mapped to its newly truncated name. In this way, artifacts whose file names do not adhere to the 8-character limit can still be migrated into an SCLM-managed project with their unique file name as shown in Figure 25-33.

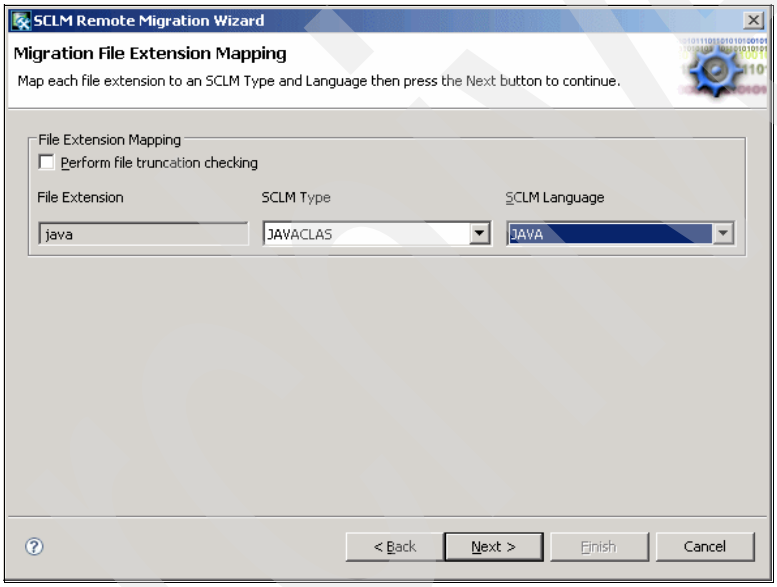


Figure 25-33 Specify the data set type and SCLM language to use for the artifacts

Once the options have been specified in the Migration File Extension Mapping window, click **Next**.

The artifact names are written to a translation table if they are longer than 8 characters. They are then written to a host-based file and migrated into the project from the file they were written to. A migration report will be returned for your review once the migration has been completed, as shown in Figure 25-34.

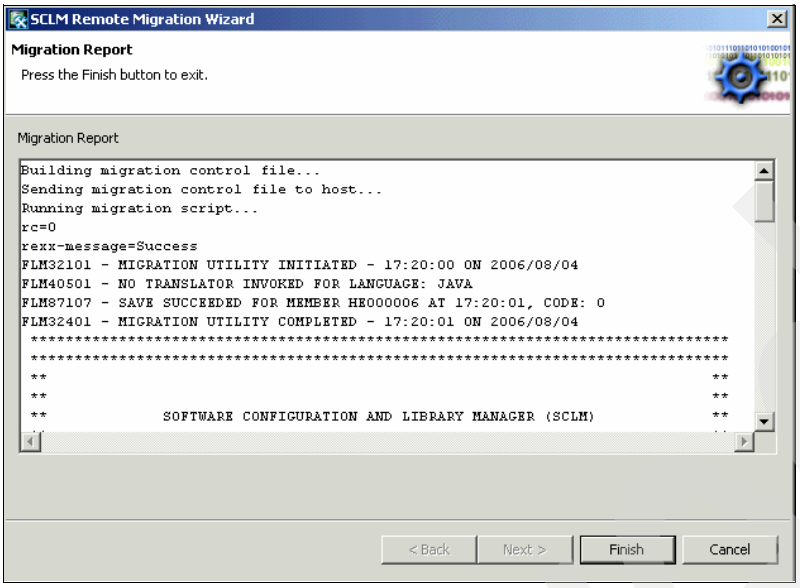


Figure 25-34 The Remote Migration Report

To view the migrated artifacts that now reside in your SCLM-managed project data sets, you can go to an ISPF Data Set List Utility to view the members.

25.4 Creating architecture definitions for the new artifacts

Once new artifacts have been migrated into the project, architecture definitions should be created for them to tell base SCLM how it is to treat the new artifacts when processing build and promote requests.

You can automatically generate SCLM architecture definitions from an existing load module or parse existing JCL into an ARCHDEF. You can also manually create ARCHDEFs from scratch. The two artifacts SAMPLE1 and SAMPLE2 that we migrated in are part of an executable load module. Therefore, we will create our architecture definition by hand, specifying the options that apply to the load module we want to create.

With the project CSJENNA open in the Project Editor frame, select **SCLM** from the menu bar above, then click **Architecture Definition Wizard** as shown in Figure 25-35.

Archive

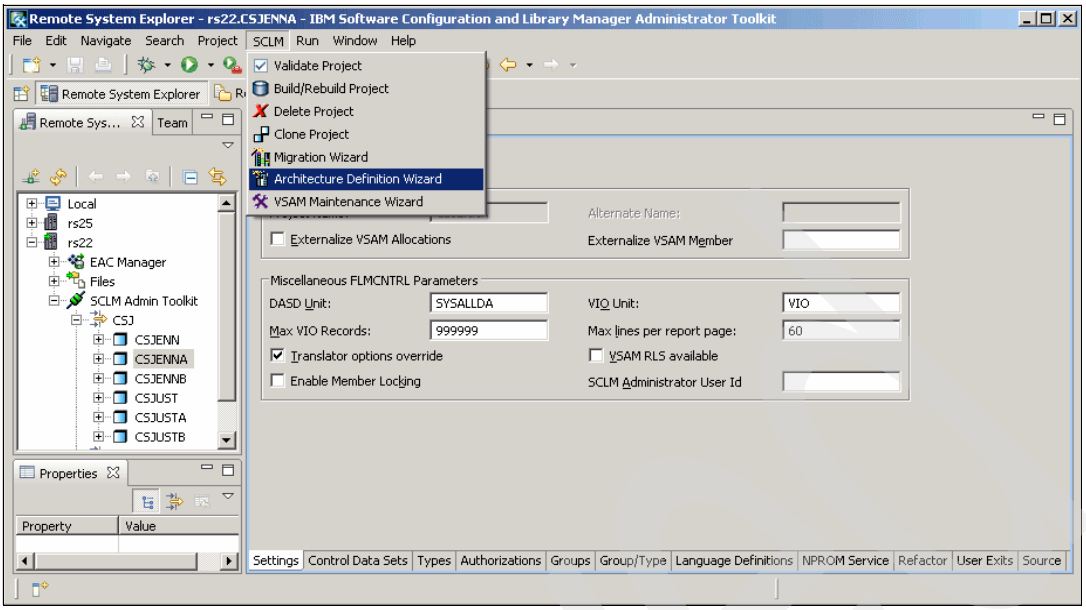


Figure 25-35 Select Architecture Definition Wizard from the pull-down menu

When the Architecture Definition Wizard opens, as shown in Figure 25-36, specify the project group you want to create the architecture definition for, and specify the type of data set the Admin Toolkit is to store the architecture definition into. Select the radio button, **Create New Architecture Definition**, and then click **Next**.

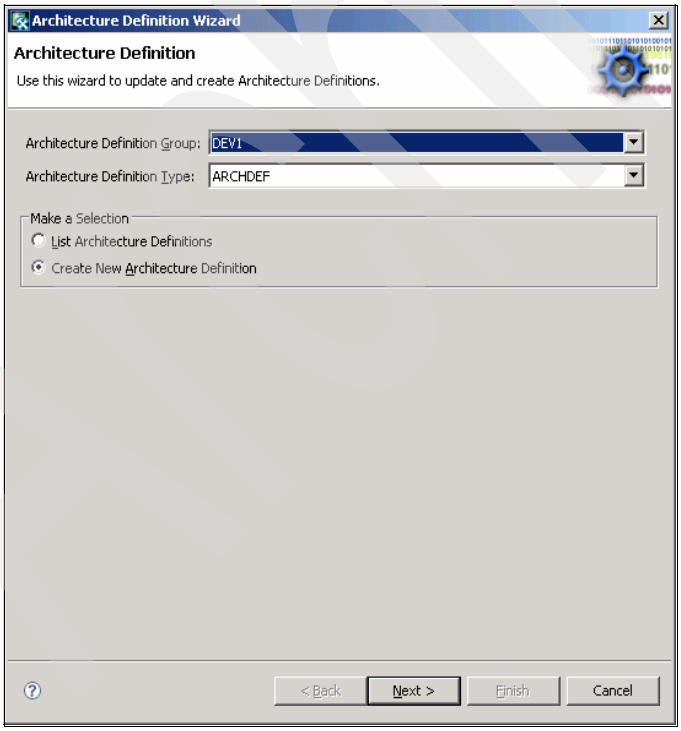


Figure 25-36 Architecture Definition Wizard window

When the window, Create Architecture Definition, is displayed as shown in Figure 25-37, provide the following information to create an architecture definition:

1. The **Architecture Definition Name** should be the PDS member name that gets generated into the data set name provided on the previous panel. In this case, we will create an architecture definition called ARCHDEF2.
2. Specify the **Kind** of architecture definition to create. In our example, we will create a Link-Edit Control architecture definition.
3. Specify the **Language** in the project that the Admin Toolkit is to use to create the architecture definition. Our example will use language FLM@ARCD, which is a standard language provided by base SCLM. We renamed this language in our project to ARCHDEF.
4. Specify the language to use as the **Linkage Editor Language.** Our example will use language FLM@L370, which is a standard language provided by base SCLM. We renamed this language in our project to LE370.
5. Tell the Admin Toolkit where SCLM is to put the **output** from building the architecture definition. Our architecture definition is going to create a LOAD module, so use LOAD as the type of output data set.
6. Tell the Admin Toolkit where SCLM is to place the **listing** from building the LOAD module. Our example will use LMAP as the type of output data set.
7. Select the **Authorization Code** to use with the artifacts defined to this architecture definition.
8. Specify an optional **Change Code** to associate with this architecture definition.
9. Specify the **Mode** to use when creating this architecture definition.

Select the radio button, **Create within a text editor**, and click **Next**.

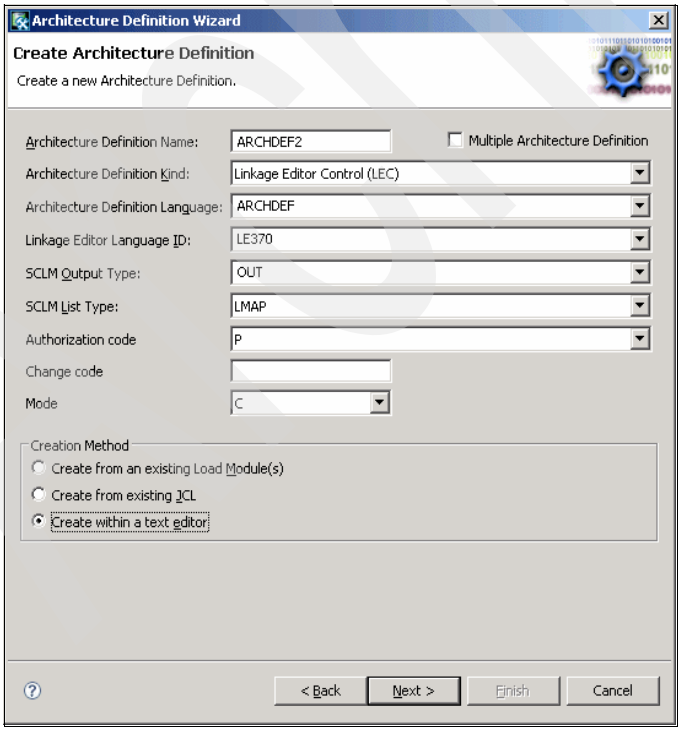


Figure 25-37 The Create Architecture Definition window specifies how to create the ARCHDEF

When the window, Architecture Definition Editor, is displayed, we type in the architecture definition text that we need to build a load module. Our changes will be saved in the specified data set and member'CSJENNA.DEV1.ARCHDEF(ARCHDEF2)' displayed on the Architecture Definition Editor window as shown in Figure 25-38.

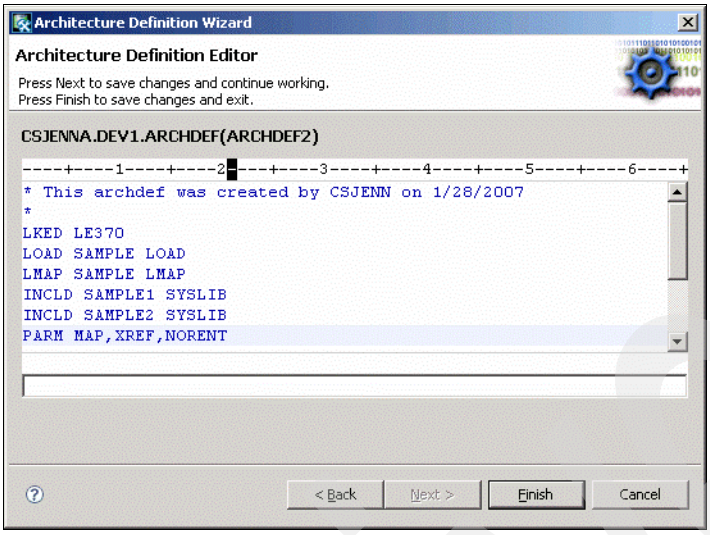


Figure 25-38 Enter the parameters for the architecture definition

This architecture definition tells base SCLM how it is to process the artifacts SAMPLE1 and SAMPLE2 during a build. This architecture definition is a Link-Edit Control architecture definition and contains instructions on how to produce a complete load module.

The keyword **LKED** identifies the language to be used to process the contents of the architecture definition. In our example, the language that will be used is LE370.

The keyword **LOAD** indentifies the load module name to be created and the type of data set to place the completed load module.

The keyword **LMAP** is optional, but tells base SCLM what group/Type data set to put the linkage editor listings. The value for this example is LMAP.

The keyword **INCLD** identifies other source members that this architecture definition references. The referenced members will be processed before this architecture definition, and their output will be used to create the load module.

The keyword **PARM** passes options to the linkage editor.

Click **Finish** to save your changes and exit from the Architecture Definition Wizard.

Part 7

Appendixes

Archived

© Copyright IBM Corp. 2007. All rights reserved.

Archived

A

Chapter 1 listings

Example: A-1 SCLM01.PROJDEFS.SOURCE(ALLOCDS)

```
//SCLMADM$ JOB (ACCT#),'ALLOC-DSN',CLASS=A,REGION=4096K,
//    MSGCLASS=X,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//*--------------------------------------------------------------------*
//*  SCLM01.PROJDEFS.SOURCE(ALLOCDS)                                   *
//*
//*  DELETE AND REALLOCATE THE DEMO BASE AND VERSION FILES.            *
//*--------------------------------------------------------------------*
//*
//*--------------------------------------------------------------------*
//*  PROC TO ALLOCATE FIXED OUTPUT LIBRARIES                           *
//*  LMAP                                                              *
//*--------------------------------------------------------------------*
//FB121   PROC FILE=
//STEP1   EXEC PGM=IEFBR14
//DD01 DD  DSN=&FILE,
//     DISP=(NEW,CATLG,DELETE),
//     UNIT=SYSDA,
//     SPACE=(CYL,(2,10,10)),
//     DCB=(RECFM=FBA,LRECL=121,BLKSIZE=0)
//   PEND
//*--------------------------------------------------------------------*
//*  PROC TO ALLOCATE VARIABLE BLOCK OUTPUT LIBRARIES                  *
//*  SOURCLST                                                          *
//*--------------------------------------------------------------------*
//VB137   PROC FILE=
//STEP1   EXEC PGM=IEFBR14
//DD01 DD  DSN=&FILE,
//     DISP=(NEW,CATLG,DELETE),
//     UNIT=SYSDA,
//     SPACE=(CYL,(2,10,10)),
//     DCB=(RECFM=VBA,LRECL=137,BLKSIZE=0)
//   PEND
//*--------------------------------------------------------------------*
```

© Copyright IBM Corp. 2007. All rights reserved.

```
//*  PROC TO ALLOCATE FIXED OUTPUT LIBRARIES                        *
//*  OBJLIB, SOURCE, COPYLIB, ETC.                                  *
//*-----------------------------------------------------------------*
//FB80    PROC FILE=
//STEP1   EXEC PGM=IEFBR14
//DD01 DD  DSN=&FILE,
//     DISP=(NEW,CATLG,DELETE),
//     UNIT=SYSDA,
//     SPACE=(CYL,(1,5,10)),
//     DCB=(RECFM=FB,LRECL=80,BLKSIZE=0)
//   PEND
//*-----------------------------------------------------------------*
//*  PROC TO ALLOCATE FIXED OUTPUT LIBRARIES                        *
//*  LOADLIB                                                        *
//*-----------------------------------------------------------------*
//LOAD    PROC FILE=
//STEP1   EXEC PGM=IEFBR14
//DD01 DD  DSN=&FILE,
//     DISP=(NEW,CATLG,DELETE),
//     UNIT=SYSDA,
//     SPACE=(CYL,(2,10,10)),
//     DCB=(RECFM=U,LRECL=0,BLKSIZE=6144)
//   PEND
//*-----------------------------------------------------------------*
//*  NOW DO THE ALLOCATES                                           *
//*-----------------------------------------------------------------*
//FILE01  EXEC   FB80,FILE=SCLM01.DEV1.ARCHDEF
//FILE02  EXEC   FB80,FILE=SCLM01.DEV1.OBJ
//FILE03  EXEC   FB80,FILE=SCLM01.DEV1.SOURCE
//FILE04  EXEC   FB80,FILE=SCLM01.DEV1.PLINCL
//FILE05  EXEC   FB80,FILE=SCLM01.DEV1.COPYLIB
//FILE06  EXEC   FB80,FILE=SCLM01.DEV1.MACROS
//FILE07  EXEC   LOAD,FILE=SCLM01.DEV1.LOAD
//FILE08  EXEC  VB137,FILE=SCLM01.DEV1.LIST
//FILE09  EXEC  FB121,FILE=SCLM01.DEV1.LMAP
//*
//FILE11  EXEC   FB80,FILE=SCLM01.DEV2.ARCHDEF
//FILE12  EXEC   FB80,FILE=SCLM01.DEV2.OBJ
//FILE13  EXEC   FB80,FILE=SCLM01.DEV2.SOURCE
//FILE14  EXEC   FB80,FILE=SCLM01.DEV2.PLINCL
//FILE15  EXEC   FB80,FILE=SCLM01.DEV2.COPYLIB
//FILE16  EXEC   FB80,FILE=SCLM01.DEV2.MACROS
//FILE17  EXEC   LOAD,FILE=SCLM01.DEV2.LOAD
//FILE18  EXEC  VB137,FILE=SCLM01.DEV2.LIST
//FILE19  EXEC  FB121,FILE=SCLM01.DEV2.LMAP
//*
//FILE21  EXEC   FB80,FILE=SCLM01.TEST.ARCHDEF
//FILE22  EXEC   FB80,FILE=SCLM01.TEST.OBJ
//FILE23  EXEC   FB80,FILE=SCLM01.TEST.SOURCE
//FILE24  EXEC   FB80,FILE=SCLM01.TEST.PLINCL
//FILE25  EXEC   FB80,FILE=SCLM01.TEST.COPYLIB
//FILE26  EXEC   FB80,FILE=SCLM01.TEST.MACROS
//FILE27  EXEC   LOAD,FILE=SCLM01.TEST.LOAD
//FILE28  EXEC  VB137,FILE=SCLM01.TEST.LIST
//FILE29  EXEC  FB121,FILE=SCLM01.TEST.LMAP
```

Archived

```
//*
//FILE31  EXEC   FB80,FILE=SCLM01.PROD.ARCHDEF
//FILE32  EXEC   FB80,FILE=SCLM01.PROD.OBJ
//FILE33  EXEC   FB80,FILE=SCLM01.PROD.SOURCE
//FILE34  EXEC   FB80,FILE=SCLM01.PROD.PLINCL
//FILE35  EXEC   FB80,FILE=SCLM01.PROD.COPYLIB
//FILE36  EXEC   FB80,FILE=SCLM01.PROD.MACROS
//FILE37  EXEC   LOAD,FILE=SCLM01.PROD.LOAD
//FILE38  EXEC  VB137,FILE=SCLM01.PROD.LIST
//FILE39  EXEC  FB121,FILE=SCLM01.PROD.LMAP
//*
```

Example: A-2 SCLM01.PROJDEFS.SOURCE(ALLOCVER)

```
//SCLMADM$ JOB (ACCT#),'ALLOC-DSN',CLASS=A,REGION=4096K,
//    MSGCLASS=X,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//*--------------------------------------------------------------------*
//*  SCLM01.PROJDEFS.SOURCE(ALLOCVER)                                  *
//*
//*  ALLOCATE THE SAMPLE VERSION FILES.                                *
//*--------------------------------------------------------------------*
//*  PROC TO ALLOCATE VARIABLE BLOCK VERSION LIBRARIES                 *
//*  VERSION.*                                                         *
//*--------------------------------------------------------------------*
//VB259   PROC FILE=
//STEP1   EXEC PGM=IEFBR14
//DD01 DD  DSN=&FILE,
//     DISP=(NEW,CATLG,DELETE),
//     UNIT=SYSDA,
//     SPACE=(CYL,(4,20,10)),
//     DCB=(RECFM=VB,LRECL=259,BLKSIZE=0)
//   PEND
//*--------------------------------------------------------------------*
//*  NOW DO THE ALLOCATES                                              *
//*--------------------------------------------------------------------*
//FILE21  EXEC   VB259,FILE=SCLM01.TEST.ARCHDEF.VERSION
//FILE23  EXEC   VB259,FILE=SCLM01.TEST.SOURCE.VERSION
//FILE24  EXEC   VB259,FILE=SCLM01.TEST.PLINCL.VERSION
//FILE25  EXEC   VB259,FILE=SCLM01.TEST.COPYLIB.VERSION
//FILE26  EXEC   VB259,FILE=SCLM01.TEST.MACROS.VERSION
//*
//FILE31  EXEC   VB259,FILE=SCLM01.PROD.ARCHDEF.VERSION
//FILE33  EXEC   VB259,FILE=SCLM01.PROD.SOURCE.VERSION
//FILE34  EXEC   VB259,FILE=SCLM01.PROD.PLINCL.VERSION
//FILE35  EXEC   VB259,FILE=SCLM01.PROD.COPYLIB.VERSION
//FILE36  EXEC   VB259,FILE=SCLM01.PROD.MACROS.VERSION
//*
```

Archived

Example: A-3 SCLM01.PROJDEFS.SOURCE(FLM@ARCD)

```
************************************************************************
* FLM@ARCD - ARCHITECTURE DEFINITIONS                                  *
*                                                                      *
*********************  GENERAL NOTES  ******************************
*                                                                      *
*  THIS LANGUAGE DEFINITION IS AN EXAMPLE THAT CAN SERVE AS A          *
*  REFERENCE IN THE CONSTRUCTION AND CUSTOMIZATION OF LANGUAGE         *
*  DEFINITIONS FOR A PARTICULAR APPLICATION AND ENVIRONMENT.           *
*                                                                      *
******************** CUSTOMIZATION NOTES ****************************
*                                                                      *
* THIS IS A PARSER TRANSLATOR AND ONLY USED TO GATHER STATISTICS       *
* FOR THE ARCHITECTURE DEFINITIONS.                                    *
* ITS USE IS OPTIONAL.                                                 *
*                                                                      *
* BE SURE TO ADD DSNAME PARAMETER IF TRANSLATOR LIES IN A PRIVATE      *
* LIBRARY.                                                             *
************************************************************************
* CHANGE ACTIVITY:                                                     *
* @01 2005/05/16  - ADDED LANGUAGE DESCRIPTION                        *
*                                                                      *
************************************************************************
         FLMLANGL    LANG=ARCHDEF,ARCH=Y,VERSION=1,                   C
               LANGDESC='ARCHITECTURE DEFINITION'
*
          FLMTRNSL   CALLNAM='ARCHDEF PARSER',                        C
               FUNCTN=PARSE,                                          C
               COMPILE=FLMLSS,                                        C
               PORDER=1,                                              C
               OPTIONS=(PTABLEDD=,                                    C
               SOURCEDD=SOURCE,                                       C
               TBLNAME=FLMPALST,                                      C
               STATINFO=@@FLMSTP,                                     C
               LISTINFO=@@FLMLIS,                                     C
               LISTSIZE=@@FLMSIZ,                                     C
               CONTIN=0,                                              C
               EOLCOL=72)
*           (* SOURCE     *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
* 5694-A01 (C) COPYRIGHT IBM CORP 1980, 2006
```

Archived

Example: A-4 SCLM01.PROJDEFS.SOURCE(FLM@COBE)

```
*********************************************************************
*          ENTERPRISE COBOL LANGUAGE DEFINITION FOR SCLM            *
*                                                                   *
*                                                                   *
*********************  GENERAL NOTES  ***************************** *
*                                                                   *
*  THIS LANGUAGE DEFINITION IS AN EXAMPLE THAT CAN SERVE AS A       *
*  REFERENCE IN THE CONSTRUCTION AND CUSTOMIZATION OF LANGUAGE      *
*  DEFINITIONS FOR A PARTICULAR APPLICATION AND ENVIRONMENT.        *
*                                                                   *
******************** CUSTOMIZATION NOTES ***************************
*                                                                   *
* LIST EACH "STATIC" COPY DATASET UNDER THE FLMSYSLB MACRO.         *
* USE THE LANGUAGE NAME (VALUE OF THE LANG KEYWORD) FOR THE LABEL   *
* ON THE FIRST FLMSYSLB MACRO.                                      *
*   EXAMPLE:                                                        *
*    COBE    FLMSYSLB   XXXXX.XXXXX.XXXXX                          *
*            FLMSYSLB   YYYYY.YYYYY.YYYYY                          *
*                                                                   *
* CUSTOMIZE THE 'OPTIONS' AND 'GOODRC' FIELDS TO YOUR STANDARDS.    *
*                                                                   *
* ADD THE 'DSNAME' FIELD IF THE TRANSLATOR IS IN A PRIVATE LIBRARY. *
*                                                                   *
* CHANGING THE VERSION FIELD ON FLMLANGL WILL CAUSE ALL MEMBERS TO BE *
* OUT OF DATE.  EACH MEMBER WILL BE RE-BUILT THE NEXT TIME BUILD IS *
* INVOKED.                                                          *
*                                                                   *
* IF YOU DON'T WANT A COMPILER LISTING, CHANGE THE DDNAME=SYSPRINT  *
* TO AN IOTYPE=W AND REMOVE THE UNNECESSARY PARAMETERS.             *
*                                                                   *
* THIS LANGUAGE DEFINITION IS COMPATIBLE WITH THE AD/CYCLE COBOL    *
* COMPILER.                                                         *
*                                                                   *
*********************************************************************
*-------------------------------------------------------------------*
* CHANGE ACTIVITY:                                                  *
* FLAG=REASON DATE      ID  DESCRIPTION                             *
* ------------------------------------------------------------------*
* M42P2691    1994/04/26 - : CHANGED DFLTTYP=COMPLIST TO LIST TO    *
*             MATCH TYPES DEFINED IN EXAMPLE PROJECTS.              *
* $01 OW03966           - : V4.1 DEVELOPMENT APAR                   *
* $02 OA17586 2006/09/29 RX: REMOVED COB2 COMMENTS, TSLINL, TSSEQP. *
*                                                                   *
*********************************************************************
*
       FLMLANGL    LANG=COBE,VERSION=D071115,ALCSYSLB=Y,            C
             LANGDESC='ENTERPRISE COBOL'
*
*********************************************************************
*       --INCLUDE SET SPECIFICATION FOR SAMPLE PROJECT SCLM01--      *
*********************************************************************
*
         FLMINCLS   TYPES=(COPYLIB)
*
```

```
*
*********************************************************************
*         --PARSER TRANSLATOR--                                     *
*********************************************************************
*
          FLMTRNSL   CALLNAM='SCLM COBOL PARSE',                      C
               FUNCTN=PARSE,                                          C
               COMPILE=FLMLPCBL,                                      C
               PORDER=1,                                              C
               CALLMETH=LINK,                                         C
               OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,)
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
*********************************************************************
*         --ENTERPRISE COBOL INTERFACE--                            *
*********************************************************************
*
          FLMTRNSL   CALLNAM='ENTERPRISE COBOL COMPILER',             C
               FUNCTN=BUILD,                                          C
               COMPILE=IGYCRCTL,                                      C
               DSNAME=ECOBOL.V340.SIGYCOMP,                           C
               VERSION=3.4.0,                                         C
               GOODRC=4,                                              C
               PORDER=1,                                              C
               OPTIONS=(XREF,LIB,APOST,NODYNAM,LIST,NONUMBER,NOSEQ)
*
*********************************************************************
*         --DDNAME ALLOCATION--                                     *
*********************************************************************
*
           FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,               C
               RECNUM=5000,DFLTTYP=OBJ
*
           FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
*
           FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT2,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT3,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT4,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT5,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT6,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT7,RECNUM=5000
*
           FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
            FLMCPYLB NULLFILE
```

Archived

```
*
          FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
           FLMCPYLB NULLFILE
*
          FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,KEYREF=LIST,             C
              RECFM=FBA,LRECL=133,                                    C
              RECNUM=50000,PRINT=Y,DFLTTYP=LIST
*                                                                     *
*                                                                     *
* 5694-A01     COPYRIGHT IBM CORP 1990, 2007                          *
*                                                                     *
```

Example: A-5 SCLM01.PROJDEFS.SOURCE(FLM@HLAS)

```
*********************************************************************
*          HIGH LEVEL ASSEMBLER LANGUAGE DEFINITION FOR SCLM        *
*                                                                   *
*********************  GENERAL NOTES  ***************************** *
*                                                                   *
*  THIS LANGUAGE DEFINITION IS AN EXAMPLE THAT CAN SERVE AS A       *
*  REFERENCE IN THE CONSTRUCTION AND CUSTOMIZATION OF LANGUAGE      *
*  DEFINITIONS FOR A PARTICULAR APPLICATION AND ENVIRONMENT.        *
*                                                                   *
********************* CUSTOMIZATION NOTES ***************************
*                                                                   *
* LIST EACH "STATIC" COPY DATASET UNDER THE FLMSYSLB MACRO.         *
* USE THE LANGUAGE NAME (VALUE OF THE LANG KEYWORD) FOR THE LABEL   *
* ON THE FIRST FLMSYSLB MACRO.                                      *
*   EXAMPLE:                                                        *
*    HLAS    FLMSYSLB   XXXXX.XXXXX.XXXXX                          *
*            FLMSYSLB   YYYYY.YYYYY.YYYYY                          *
*                                                                   *
* CUSTOMIZE THE 'OPTIONS' AND 'GOODRC' FIELDS TO YOUR STANDARDS.    *
*                                                                   *
* ADD THE 'DSNAME' FIELD IF THE TRANSLATOR IS IN A PRIVATE LIBRARY. *
*                                                                   *
* CHANGING THE VERSION FIELD ON FLMLANGL WILL CAUSE ALL MEMBERS TO BE *
* OUT OF DATE.  EACH MEMBER WILL BE RE-BUILT THE NEXT TIME BUILD IS *
* INVOKED.                                                          *
*                                                                   *
* IF YOU DON'T WANT A COMPILER LISTING, CHANGE THE DDNAME=SYSPRINT  *
* TO AN IOTYPE=W AND REMOVE THE UNNECESSARY PARAMETERS.             *
*                                                                   *
*********************************************************************
*-------------------------------------------------------------------*
* CHANGE ACTIVITY:                                                  *
* FLAG=REASON DATE       ID  DESCRIPTION                            *
* ------------------------------------------------------------------*
* $01         2006/10/20 - : ADD LANGUAGE DESCRIPTION.              *
* $02=OA17586 2006/11/17 RX: SPECIFIED SYSPRINT AS FBA 121. REMOVED *
*             OS/VS COMMENTS. UPDATED HIGH LEVEL ASSEMBLER VERSION  *
*             FROM 1 TO V1R5.                                       *
*                                                                   *
```

Archived

```
*********************************************************************
HLAS     FLMSYSLB SYS1.MACLIB
*
         FLMLANGL    LANG=HLAS,VERSION=D071123,ALCSYSLB=Y,            C
               LANGDESC='HIGH LEVEL ASSEMBLER'
*
*********************************************************************
*         --PARSER TRANSLATOR--                                     *
*********************************************************************
          FLMTRNSL   CALLNAM='SCLM HLAS PARSE',                      C
               FUNCTN=PARSE,                                         C
               COMPILE=FLMLPGEN,                                     C
               PORDER=1,                                             C
               OPTIONS=(SOURCEDD=SOURCE,                             C
               STATINFO=@@FLMSTP,                                    C
               LISTINFO=@@FLMLIS,                                    C
               LISTSIZE=@@FLMSIZ,                                    C
               LANG=A)              *** THIS IS ASSEMBLER ONLY ***
*          (* SOURCE      *)
          FLMALLOC  IOTYPE=A,DDNAME=SOURCE
           FLMCPYLB @@FLMDSN(@@FLMMBR)
*
*********************************************************************
*         --BUILD TRANSLATOR(S)-- --HIGH LEVEL ASSEMBLER INTERFACE--  *
*********************************************************************
*
          FLMTRNSL   CALLNAM='HL ASM',                               C
               FUNCTN=BUILD,                                         C
               COMPILE=ASMA90,                                       C
               VERSION=1.5,                                          C
               GOODRC=0,                                             C
               PORDER=1,                                             C
               OPTIONS=(XREF(SHORT),LINECOUNT(75),OBJECT,RENT)
*
*********************************************************************
*         --DDNAME ALLOCATION--                                     *
*********************************************************************
*
           FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=9000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,RECNUM=15000
*
           FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,               C
               RECNUM=9000,DFLTTYP=OBJ
*
           FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
            FLMCPYLB NULLFILE
*
           FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
            FLMCPYLB NULLFILE
*
           FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
*
           FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,KEYREF=LIST,PRINT=Y,    C
               RECFM=FBA,LRECL=121,                                  C
```

Archived

```
              DFLTTYP=LIST,RECNUM=20000
*                                                                     *
*                                                                     *
* 5694-A01     COPYRIGHT IBM CORP 1980, 2007                          *
*                                                                     *
```

Example: A-6 SCLM01.PROJDEFS.SOURCE(FLM@L370)

```
*********************************************************************
*        370/LINKAGE EDITOR LANGUAGE DEFINITION FOR SCLM            *
*                                                                   *
*********************  GENERAL NOTES  ********************************
*                                                                   *
*  THIS LANGUAGE DEFINITION IS AN EXAMPLE THAT CAN SERVE AS A       *
*  REFERENCE IN THE CONSTRUCTION AND CUSTOMIZATION OF LANGUAGE      *
*  DEFINITIONS FOR A PARTICULAR APPLICATION AND ENVIRONMENT.        *
*                                                                   *
******************** CUSTOMIZATION NOTES ****************************
*                                                                   *
* ADD FLMCPYLB MACROS FOR EACH 'STATIC' INPUT DATASET FOR LINKEDIT  *
* PROCESSING, TO THE 'SYSLIB' FLMALLOC MACRO.                       *
* MAKE SURE PORDER=3. THE LINK EDIT USES DD NAME LIST SUBSTITUTION. *
* CUSTOMIZE THE 'OPTIONS' AND 'GOODRC' FIELDS TO YOUR STANDARDS.    *
*                                                                   *
*                                                                   *
*********************************************************************
*-------------------------------------------------------------------*
* CHANGE ACTIVITY:                                                  *
* FLAG=REASON DATE       ID  DESCRIPTION                            *
* ------------------------------------------------------------------*
* OY34273     1990/08/10 - : CHANGED LRECL=6144 TO LRECL=0 FOR      *
*             SYSLMOD AND SYSUT1.  GT4045 - AAB.                    *
* PTR2783     1993/03/08 - : REMOVED COMMENTED LINES THAT DID NOT   *
*             ADD VALUE.   :                                        *
* M42SLD      1994/04/26 - : ADDED ADDITIONAL LIBRARIES TO SYSLIB   *
*             CONCATENATION TO SUPPORT CICS ENVIRONMENT.            *
* OW03966     XXXX/XX/XX - : V4.1 DEVELOPMENT APAR                  *
* Z/OS 1.5    XXXX/XX/XX - : R15DEV DEVELOPMENT CLEAN-UP            *
* $01=Z/OS 1.8 2005/05/16 -: ADD LANGUAGE DESCRIPTION.              *
* $02=OA17586 2006/10/23 RX: ADDED CEE.SCEELKED TO SYSLIB           *
*             CONCATENATION TO SUPPORT ENTERPRISE COBOL AND PL/I.   *
*                                                                   *
*********************************************************************
*
        FLMLANGL    LANG=LE370,CANEDIT=N,VERSION=D071108,            C
              LANGDESC='370/LINKAGE EDITOR'
*
         FLMTRNSL   CALLNAM='LKED/370',                             C
              FUNCTN=BUILD,                                         C
              COMPILE=IEWL,                                         C
              VERSION=F64,                                          C
              GOODRC=0,                                             C
              PORDER=3,                                             C
              OPTIONS=(DCBS,MAP)
```

```
*
*  1       (* SYSLIN *)
           FLMALLOC  IOTYPE=S,KEYREF=INCL,RECFM=FB,LRECL=80,          C
               RECNUM=20000,DDNAME=SYSLIN
*
*  2       (* LOAD MODULE NAME *)
           FLMALLOC  IOTYPE=L,KEYREF=LOAD
*
*  3       (* SYSLMOD *)
           FLMALLOC  IOTYPE=P,KEYREF=LOAD,RECFM=U,LRECL=0,            C
               RECNUM=500,DIRBLKS=20,DDNAME=SYSLMOD
*
*  4       (* SYSLIB  *)
           FLMALLOC  IOTYPE=A,DDNAME=SYSLIB
            FLMCPYLB CEE.SCEELKED
            FLMCPYLB CICS.TS31.CICS.SDFHLOAD
            FLMCPYLB DB2.V810.SDSNLOAD
            FLMCPYLB SYS1.LINKLIB
*
*  5       (* N/A *)
           FLMALLOC  IOTYPE=N
*
*  6       (* SYSPRINT *)
           FLMALLOC  IOTYPE=O,KEYREF=LMAP,RECFM=FBA,LRECL=121,        C
               RECNUM=2500,PRINT=Y,DDNAME=SYSPRINT
*
*  7       (* N/A *)
           FLMALLOC  IOTYPE=N
*
*  8       (* N/A *)
           FLMALLOC  IOTYPE=N
*
*  9       (* N/A *)
           FLMALLOC  IOTYPE=N
*
* 10       (* N/A *)
           FLMALLOC  IOTYPE=N
*
* 11       (* N/A *)
           FLMALLOC  IOTYPE=N
*
* 12       (* SYSTERM *)
           FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
            FLMCPYLB NULLFILE
*
*
* 5694-A01     COPYRIGHT IBM CORP 1980, 2007
*
```

Archived

Example: A-7 Figure 1-7. SCLM01.PROJDEFS.SOURCE(FLM@PLIE)

```
************************************************************************
*     ENTERPRISE PL/I COMPILER LANGUAGE DEFINITION FOR SCLM            *
*                                                                      *
*********************  GENERAL NOTES  ******************************
*                                                                      *
*  THIS LANGUAGE DEFINITION IS AN EXAMPLE THAT CAN SERVE AS A          *
*  REFERENCE IN THE CONSTRUCTION AND CUSTOMIZATION OF LANGUAGE         *
*  DEFINITIONS FOR A PARTICULAR APPLICATION AND ENVIRONMENT.           *
*                                                                      *
******************** CUSTOMIZATION NOTES ****************************
*                                                                      *
* LIST EACH "STATIC" COPY DATASET UNDER THE FLMSYSLB MACRO.            *
* USE THE LANGUAGE NAME (VALUE OF THE LANG KEYWORD) FOR THE LABEL      *
* ON THE FIRST FLMSYSLB MACRO.                                         *
*   EXAMPLE:                                                           *
*    PLIE    FLMSYSLB   XXXXX.XXXXX.XXXXX                            *
*            FLMSYSLB   YYYYY.YYYYY.YYYYY                            *
*                                                                      *
* CUSTOMIZE THE 'OPTIONS' AND 'GOODRC' FIELDS TO YOUR STANDARDS.       *
*                                                                      *
* ADD THE 'DSNAME' FIELD IF THE TRANSLATOR IS IN A PRIVATE LIBRARY.    *
*                                                                      *
* CHANGING THE VERSION FIELD ON FLMLANGL WILL CAUSE ALL MEMBERS TO BE  *
* OUT OF DATE.  EACH MEMBER WILL BE RE-BUILT THE NEXT TIME BUILD IS    *
* INVOKED.                                                             *
*                                                                      *
* IF YOU DON'T WANT A COMPILER LISTING, CHANGE THE DDNAME=SYSPRINT     *
* TO AN IOTYPE=W AND REMOVE THE UNNECESSARY PARAMETERS.                *
*                                                                      *
* THE PL/I COMPILER GIVE OUT A NUMBER OF WARNINGS OFTEN RESULTING      *
* IN A RETURN CODE OF 4.  YOU MIGHT WANT TO ADJUST YOUR GOODRC=0 TO    *
* GOODRC=4 TO ACCOUNT FOR THIS.                                        *
*                                                                      *
************************************************************************
*----------------------------------------------------------------------*
* CHANGE ACTIVITY:                                                     *
* FLAG=REASON DATE       ID  DESCRIPTION                               *
* ---------------------------------------------------------------------*
* @01        2005/05/16     - ADDED LANGUAGE DESCRIPTION              *
* $02=OA17586 2006/08/24 RX: REMOVED PLIPRPX DD. REPALCED             *
*            DDNAME=SYSCIN WITH DDNAME=SYSIN. CHANGED BUILD           *
*            TRANSLATOR PORDER TO 1. SPECIFIED SYSPRINT AS VBA 137.   *
*                                                                      *
************************************************************************
       FLMLANGL    LANG=PLIE,VERSION=PLIEV1.0,                        C
             LANGDESC='ENTERPRISE PL/I'
*
************************************************************************
*       --INCLUDE SET SPECIFICATION FOR EXAMPLE PROJECT SCLM01--       *
************************************************************************
*
         FLMINCLS   TYPES=(PLINCL)
*
*
```

Archived

```
*********************************************************************
*         --PARSER TRANSLATOR--                                     *
*********************************************************************
*
          FLMTRNSL   CALLNAM='SCLM PL/I PARSE',                      C
               FUNCTN=PARSE,                                         C
               COMPILE=FLMLPGEN,                                     C
               PORDER=1,                                             C
               OPTIONS=(SOURCEDD=SOURCE,                             C
               STATINFO=@@FLMSTP,                                    C
               LISTINFO=@@FLMLIS,                                    C
               LISTSIZE=@@FLMSIZ,                                    C
               LANG=I)
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
*********************************************************************
*         --PL/I ENTERPRISE INTERFACE--                             *
*********************************************************************
*
          FLMTRNSL   CALLNAM='ENTERPRISE PL/I COMPILER',             C
               FUNCTN=BUILD,                                         C
               COMPILE=IBMZPLI,                                      C
               DSNAME=EPLI.V350.SIBMZCMP,                            C
               VERSION=3.3.0,                                        C
               GOODRC=0,                                             C
               PORDER=1,                                             C
               OPTIONS=(MACRO,OBJECT,SOURCE,XREF)
*
*********************************************************************
*         --DDNAME ALLOCATION--                                     *
*********************************************************************
*
           FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,DFLTTYP=OBJ,    C
               RECNUM=5000
*
           FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,RECNUM=5000
           FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000 @02C
*                                                                  2@02D
           FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
            FLMCPYLB NULLFILE
*
           FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,                       C
               KEYREF=LIST,DFLTTYP=LIST,                             C
               RECFM=VBA,LRECL=137,                                  C
               PRINT=Y,RECNUM=5000
*                                                                    *
*                                                                    *
* 5694-A01     COPYRIGHT IBM CORP 1980, 2007                         *
*
```

Archived

Example: A-8 SCLM01.PROJDEFS.SOURCE(FLM@TEXT)

```
*********************************************************************
*       UNTRANSLATED TEXT LANGUAGE DEFINITION FOR SCLM              *
*                                                                   *
*********************  GENERAL NOTES  ********************************
*                                                                   *
*  THIS LANGUAGE DEFINITION IS AN EXAMPLE THAT CAN SERVE AS A       *
*  REFERENCE IN THE CONSTRUCTION AND CUSTOMIZATION OF LANGUAGE      *
*  DEFINITIONS FOR A PARTICULAR APPLICATION AND ENVIRONMENT.        *
*                                                                   *
*********************************************************************
*                                                                   *
* CUSTOMIZATION IS NOT REQUIRED.                                    *
*********************************************************************
* CHANGE ACTIVITY:                                                  *
*                                                                   *
* @01 2005/05/16  - ADDED LANGUAGE DESCRIPTION                      *
*                                                                   *
*********************************************************************
         FLMLANGL    LANG=TEXT,VERSION=TEXTV1.0,                     C
               LANGDESC='UNTRANSLATED TEXT'
*
* PARSER TRANSLATOR
*
          FLMTRNSL   CALLNAM='SCLM TEXT PARSE',                      C
               FUNCTN=PARSE,                                         C
               COMPILE=FLMLPGEN,                                     C
               PORDER=1,                                             C
               OPTIONS=(SOURCEDD=SOURCE,                             C
               STATINFO=@@FLMSTP,                                    C
               LISTINFO=@@FLMLIS,                                    C
               LISTSIZE=@@FLMSIZ,                                    C
               LANG=T)
*          (* SOURCE      *)
          FLMALLOC  IOTYPE=A,DDNAME=SOURCE
           FLMCPYLB @@FLMDSN(@@FLMMBR)
*
* BUILD TRANSLATOR(S)
*
* 5694-A01 (C) COPYRIGHT IBM CORP 1980, 2006
```

Example: A-9 SCLM01.PROJDEFS.SOURCE(SCLMACCT)

```
//SCLMACCT JOB (76T12B0,629,671,T12,,N),'SCLMADM',CLASS=A,
// NOTIFY=&SYSUID,MSGCLASS=X
//*
//* SCLM01.PROJDEFS.SOURCE(SCLMACCT)
//*
//* DELETE, DEFINE AND INITIALIZE SCLM ACCOUNT FILE.
//*
//ACCOUNT EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
```

Archived

```
    DELETE 'SCLM01.ACCOUNT.FILE' PURGE CLUSTER
    DEFINE CLUSTER (NAME('SCLM01.ACCOUNT.FILE') -
                    CYL(50 10) -
                    KEYS(26 0) -
                    RECORDSIZE(264 32000) -
                    SHAREOPTIONS(4,3) -
                    IMBED SPEED UNIQUE SPANNED) -
            INDEX  (NAME('SCLM01.ACCOUNT.FILE.INDEX')) -
            DATA   (NAME('SCLM01.ACCOUNT.FILE.DATA') -
                    CISZ(2048) -
                    FREESPACE(50 50))
/*
//INITACCT EXEC PGM=IDCAMS,COND=(4,LT)
//INPUT DD *
VSAM DATABASE INITIALIZATION RECORD
/*
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
   REPRO INFILE(INPUT) OUTDATASET('SCLM01.ACCOUNT.FILE')
/*
```

Example: A-10 SCLM01.PROJDEFS.SOURCE(SCLMVERS)

```
//SCLMVERS JOB (76T12BO,629,671,T12,,N),'SCLMADM',CLASS=A,
// NOTIFY=&SYSUID,MSGCLASS=T
//*
//* SCLM01.PROJDEFS.SOURCE(SCLMVERS)
//*
//* DELETE, DEFINE AND INITIALIZE SCLM VERSION FILE.
//*
//VERSION EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
    DELETE 'SCLM01.VERSION.FILE' PURGE CLUSTER
    DEFINE CLUSTER +
          (NAME('SCLM01.VERSION.FILE') +
           CYL(10 10) +
           KEYS(40 0) +
           IMBED +
           RECORDSIZE(264 32000) +
           SHAREOPTIONS(4,3) +
           SPEED +
           SPANNED +
           UNIQUE) +
           INDEX(NAME('SCLM01.VERSION.FILE.INDEX') -
           ) +
           DATA(NAME('SCLM01.VERSION.FILE.DATA') -
           CISZ(2048) +
           FREESPACE(50 50) +
           )
/*
//*********************************************************************
//*
```

Archived

```
//*  INITIALIZE THE VERSION CONTROL FILE
//*
//*********************************************************************
//INITVERS EXEC PGM=IDCAMS
//INPUT  DD *
                       SCLM VERSION CONTROL FILE INITIALIZATION RECORD
/*
//OUTPUT DD DSN=SCLM01.VERSION.FILE,DISP=SHR
//SYSPRINT  DD SYSOUT=H
//SYSIN     DD *
     REPRO INFILE(INPUT) OUTFILE(OUTPUT)
/*
//*
```

Example: A-11 SCLM01.PROJDEFS.SOURCE(SCLM01)

```
TITLE '*** PROJECT DEFINITION FOR SCLM01     ***'
SCLM01   FLMABEG
*
*        ************************************************************
*        *  DEFINE THE TYPES                                        *
*        ************************************************************
*
ARCHDEF  FLMTYPE
SOURCE   FLMTYPE
COPYLIB  FLMTYPE
PLINCL   FLMTYPE
MACROS   FLMTYPE
LIST     FLMTYPE
OBJ      FLMTYPE
LMAP     FLMTYPE
LOAD     FLMTYPE
*
*        ************************************************************
*        *  DEFINE THE GROUPS                                       *
*        ************************************************************
*
DEV1     FLMGROUP AC=(P,D),KEY=Y,PROMOTE=TEST
DEV2     FLMGROUP AC=(P,D),KEY=Y,PROMOTE=TEST
TEST     FLMGROUP AC=(P),KEY=Y,PROMOTE=PROD
PROD     FLMGROUP AC=(P),KEY=Y
*
*********************************************************************
*               PROJECT CONTROLS
*********************************************************************
*
         FLMCNTRL ACCT=SCLM01.ACCOUNT.FILE,                          C
               VERS=SCLM01.VERSION.FILE,                             C
               MAXVIO=999999,                                        C
               MEMLOCK=Y,                                            C
               CONTROL=SCLM01.CONTROL.FILE,                          C
               ADMINID=DOHERTL,                                      C
               VIOUNIT=VIO
```

Archived

```
*
*********************************************************************
*               VERSIONING AND AUDITABILITY                         *
*********************************************************************
*
        FLMATVER GROUP=TEST,                                         C
              TYPE=SOURCE,                                           C
              VERSION=YES
*
        FLMATVER GROUP=PROD,                                         C
              TYPE=SOURCE,                                           C
              VERSION=YES
*
*********************************************************************
*               LANGUAGE DEFINITION TABLES
*********************************************************************
*
*        USE THE FOLLOWING FORMAT TO COPY IN THE MACROS USED BY YOUR
*        OWN SYSTEM
*
         COPY  FLM@ARCD          -- ARCHITECTURE DEFINITION
         COPY  FLM@COBE          -- ENTERPRISE COBOL  LANGUAGE DEF.
         COPY  FLM@PLIE          -- ENTERPRISE PL/I  LANGUAGE DEF.
         COPY  FLM@HLAS          -- HIGH-LEVEL ASSEMBLER LANG. DEF.
         COPY  FLM@L370          -- LINK EDIT LANG. DEFINITION
         COPY  FLM@TEXT          -- TEXT   LANGUAGE DEFINITION
*
         FLMAEND
```

Example: A-12 SCLM01.PROJDEFS.SOURCE($ASMPROJ)

```
//SCLM01A JOB (ACCOUNT),'NAME',
// MSGCLASS=X,CLASS=A,NOTIFY=&SYSUID
//*
//*-- ASSEMBLE SCLM PROJECT SCLM01
//*
//* 5647-A01 (C) COPYRIGHT IBM CORP. 1987
//*
//ASMPROJ  PROC  PROJID=,PROJDEF=
//*-------------------------------------------------------------------*
//*   ASSEMBLE AND LINK A PROJECT DEFINITION                          *
//*                                                                   *
//*   PROC PARAMETERS:                                                *
//*                                                                   *
//*      PROJID  -  HIGH-LEVEL QUALIFIER FOR PROJECT                  *
//*      PROJDEF -  PROJECT DEFINITION MEMBER NAME                    *
//*                                                                   *
//*   NOTE: MODIFY SYSLIB DSNAMES TO GET THE SCLM RELEASE MACROS      *
//*   AND ANY LANGUAGE DEFINITIONS YOU NEED.                          *
//*-------------------------------------------------------------------*
//ASM      EXEC PGM=ASMA90,REGION=4096K,PARM=OBJECT
//SYSLIB   DD DSN=&PROJID..PROJDEFS.SOURCE,DISP=SHR
//         DD DSN=ISP.SISPMACS,DISP=SHR
//SYSPRINT DD SYSOUT=H
```

```
//SYSPUNCH DD DUMMY
//SYSIN    DD DSN=&PROJID..PROJDEFS.SOURCE(&PROJDEF),DISP=SHR
//SYSUT1   DD UNIT=SYSALLDA,SPACE=(CYL,(2,2))
//SYSLIN   DD DSN=&&INT,DISP=(,PASS),UNIT=SYSDA,SPACE=(CYL,(1,5)),
//            DCB=(BLKSIZE=0)
//*-------------------------------------------------------------------*
//LINK     EXEC PGM=IEWL,PARM='RENT,LIST,MAP',REGION=512K,
//          COND=(0,NE,ASM)
//SYSPRINT DD SYSOUT=H
//SYSLIN   DD DSN=&&INT,DISP=(OLD,DELETE)
//SYSLIB   DD DSN=&PROJID..PROJDEFS.LOAD,DISP=SHR
//SYSUT1   DD UNIT=SYSALLDA,SPACE=(CYL,(2,2))
//SYSLMOD  DD DISP=SHR,DSN=&PROJID..PROJDEFS.LOAD(&PROJDEF)
//         PEND
//*-------------------------------------------------------------------*
//ASMLINK  EXEC  PROC=ASMPROJ,PROJID=SCLM01,PROJDEF=SCLM01
//
```

Example: A-13 Flexible Naming of Project Partitioned Data Sets

```
SCLM01   FLMABEG
.
.
*        ************************************************************
*        *  DEFINE THE GROUPS                                       *
*        ************************************************************
*
DEV1     FLMGROUP AC=(P,D),KEY=Y,PROMOTE=TEST,ALTC=DEVCNTL
DEV2     FLMGROUP AC=(P,D),KEY=Y,PROMOTE=TEST,ALTC=DEVCNTL
TEST     FLMGROUP AC=(P),KEY=Y,PROMOTE=PROD,ALTC=TESTCNTL
PROD     FLMGROUP AC=(P),KEY=Y
*
*********************************************************************
*              PROJECT CONTROLS
*********************************************************************
*
         FLMCNTRL ACCT=SCLM01.ACCOUNT.FILE,                          C
               VERS=SCLM01.VERSION.FILE,                             C
               DSNAME=ISPFSCLM.SCLM01.@@FLMGRP.@@FLMTYP,             C
               VERPDS=@@FLMDSN.VERSION,                              C
               MAXVIO=999999,                                        C
               VIOUNIT=VIO
*
DEVCNTL  FLMALTC ACCT=SCLM01.DEVACCT.FILE,                           C
               VERS=SCLM01.DEVVERS.FILE,                             C
               DSNAME=ISPFSCLM.DEV.@@FLMPRJ.@@FLMGRP.@@FLMTYP
*
TESTCNTL FLMALTC ACCT=SCLM01.TESTACCT.FILE,                          C
               VERS=SCLM01.TESTVERS.FILE,                            C
               DSNAME=ISPFSCLM.TEST.@@FLMPRJ.@@FLMGRP.@@FLMTYP
.
.
         FLMAEND
```

Archived

Example: A-14 Language Definitions Supplied with z/OS SCLM

```
*********************************************************************
* SCLM Language Definitions
*********************************************************************
FLM@ARCD - Architecture definitions
FLM@ASM  - OS/VS assembler language definition for SCLM
FLM@ASMC - SCLM language definition for
           OS/VS assembler with CICS preprocessor
FLM@ASMH - OS/VS assembler 'h' language definition for SCLM
FLM@BDO  - Language definition for DB2 bind/free clist output
FLM@BD2  - Language definition for DB2 bind/free clist
FLM@BMS  - SCLM language definition for
FLM@BOOK - Script 3 processor with bookmaster
FLM@CC   - SCLM language definition for
           C/370 with CICS preprocessor
FLM@CCBE - SCLM language definition for
           Enterprise COBOL with CICS preprocessor
FLM@CCOB - SCLM language definition for
           OS COBOL with CICS preprocessor
FLM@CICS - SCLM language definition for
           COBOL II with CICS preprocessor
FLM@CLNK - SCLM language definition for C pre-link with link-edit
FLM@CLST - Untranslated clist language definition for SCLM
FLM@COBL - OS/VS COBOL language definition for SCLM
FLM@COBE - Enterprise COBOL language definition for SCLM
FLM@COB2 - COBOL II language definition for SCLM
FLM@COPY - SCLM language definition to migrate object into SCLM as an
           Output
FLM@CPLK - SCLM language definition for C/370 with C pre-link
FLM@CPLE - SCLM language definition for PL/I Enterprise compiler
           And CICS preprocessor
FLM@CPLO - SCLM language definition for OS PL/I optimizer compiler
           And CICS preprocessor 3.2.1
FLM@C370 - C/370 language definition for SCLM
FLM@DTLC - Language Definition Dialog Tag Language (DTL)
FLM@EASM - SCLM language definition for OS assembler with DB2 and
           CICS preprocessor
FLM@EC   - SCLM language definition for C/370 with DB2 and CICS
           Preprocessors
FLM@ECOB - SCLM language definition OS COBOL with DB2 and CICS
           Preprocessor
FLM@ECO2 - SCLM language definition for COBOL II with DB2 and CICS
           Preprocessors
FLM@EPLO - SCLM language definition for CICS, DB2, and OS PL/I
           Optimizer compiler
FLM@FASM - SCLM language definition for High Level Assembler
           with IDILANGX output for FAULT ANALYZER
FLM@HLAF - SCLM language definition for High Level Assembler
           with IDILANGX output for FAULT ANALYZER and ASM
           compile options override on CC architecture member.
FLM@FORT - FORTRAN IV H extended language definition for SCLM
FLM@FPAS - SCLM language definition for Pascal
           with IDILANGX output for FAULT ANALYZER
FLM@HLAS - OS/VS high level assembler language definition for SCLM
FLM@IASM - SCLM language definition for OS assembler with CICS
```

Arrived

```
           Preprocessor
FLM@IC   - SCLM language definition for C/370 with CICS
           Preprocessor
FLM@ICO2 - SCLM language definition for COBOL II with CICS
           Preprocessor
FLM@IPLO - SCLM language definition for OS PL/I optimizer compiler
           And CICS preprocessor
FLM@L370 - 370/linkage editor language definition for SCLM
FLM@MVTS - SCLM language definition for MFS with TEST option
FLM@MVUT - SCLM language definition for MFS with COMPRESS option
FLM@OBJ  - SCLM Dummy language definition to migrate object and
           Load
FLM@PLIC - OS PL/I checkout compiler language definition for SCLM
FLM@PLIE - PL/I Enterprise compiler language definition for SCLM
FLM@PLIO - OS PL/I optimizing compiler language definition for SCLM
FLM@PSCL - VS pascal language definition for SCLM
FLM@RASM - OS/VS assembler 'h' language definition for SCLM
           This language definition uses the rexx parser
FLM@RCBL - OS/VS COBOL II language definition for SCLM
           This language definition uses the rexx parser
FLM@RCIS - SCLM C/370 language definition
           Using flmlrci parser (C/C++ with include set support)
FLM@RC37 - OS/VS C370 language definition for SCLM
           This language definition uses the rexx parser
FLM@REXC - This language definition is for the rexx/370 compiler
FLM@REXX - Untranslated rexx language definition for SCLM
FLM@SCRP - Script 3 processor
FLM@TEXT - Untranslated text language definition for SCLM
FLM@WBCC - SCLM language definition for BORLAND(tm) C++ for windows
FLM@WBRC - SCLM language definition for BORLAND(tm) C++ for windows
FLM@WDUM - SCLM language definition for IBM CSET/2 or CSET++ for OS/2
           Compile dummy object to hold resource dlls
FLM@WEXE - SCLM language definition for workstation links to
           Generate .exe
FLM@WICC - SCLM language definition for IBM CSET/2 or CSET++ for OS/2
           Compile source to object
FLM@WIPF - SCLM language definition to build OS/2 help
FLM@WLNK - SCLM language definition for workstation link386 command
FLM@WRC  - SCLM language definition for workstation resource
           Compiler to generate .res
FLM@WTLK - SCLM language definition for BORLAND(tm) C++ for windows
FLM@WXLC - SCLM language definition for IBM CSET++ for AIX
           Compile source to object
FLM@2ASM - SCLM language definition for DB2 and OS/VS assembler
FLM@2C   - SCLM language definition for C/370 with DB2 preprocessor
FLM@2CBE - SCLM language definition for Enterprise COBOL with
           Integrated DB2 Preprocessor
FLM@2CBF - SCLM language definition for Enterprise COBOL compiler
           with integral DB2 preprocessing and Fault Analyzer side
           file generation
FLM@2CCE - SCLM language definition for Enterprise COBOL with
           Integrated DB2 and CICS Preprocessor.
FLM@2COB - SCLM language definition for OS COBOL with DB2
           Preprocessor
FLM@2CO2 - SCLM language definition for OS COBOL II with DB2
```

```
           Preprocessor
FLM@2FRT - DB2 and FORTRAN IV H extended language definition for SCLM
FLM@2PLE - SCLM language definition for Enterprise PL/I with
           Integrated DB2 Preprocessor
FLM@2PLF - SCLM language definition for DB2 and PL/I enterprise
           compiler and NCAL linkedit to a sub-module library with
           side file generation.
FLM@2PLO - SCLM language definition for SCLM DB2 and OS PL/I
           Optimizer compiler
```

Example: A-15 Sample project definition with Package Backout

```
SCLM01   FLMABEG
*
*        ************************************************************
*        *  DEFINE THE TYPES                                        *
*        ************************************************************
*
ARCHDEF  FLMTYPE
SOURCE   FLMTYPE
COPYLIB  FLMTYPE
PLINCL   FLMTYPE
MACROS   FLMTYPE
ARCHPACK FLMTYPE  ISAPACK=Y
LIST     FLMTYPE
OBJ      FLMTYPE  BACKUP=Y
LMAP     FLMTYPE
LOAD     FLMTYPE  BACKUP=Y
DBRM     FLMTYPE  BACKUP=Y
BACKUP   FLMTYPE  PACKFILE=Y,REUSEDAY=90
*
*        ************************************************************
*        *  DEFINE THE GROUPS                                       *
*        ************************************************************
*
DEV1     FLMGROUP AC=(P,A,LONGNAME),KEY=Y,PROMOTE=TEST
DEV2     FLMGROUP AC=(P,A),KEY=Y,PROMOTE=TEST
TEST     FLMGROUP AC=(P),KEY=Y,PROMOTE=PROD
BACKGRP  FLMGROUP AC=(P),KEY=Y,PROMOTE=PROD
PROD     FLMGROUP AC=(P),KEY=Y,BKGRP=BACKGRP,BKMBRLVL=Y
.
.
.
         FLMAEND
```

Archived

Example: A-16 Sample FLMLIBS Skeleton

```
)CM
)CM THIS DEFINES THE STEPLIB AND ISPF LIBRARIES
)CM TO BE USED DURING SCLM BATCH OPERATIONS
)CM
)CM BE SURE TO INCLUDE THE LOAD LIBRARIES CONTAINING ISPF.
//*
//*****************************************************************
//* STEPLIB LIBRARIES
//*****************************************************************
//*
//STEPLIB  DD  DSN=ISP.SISPLPA,DISP=SHR
//         DD  DSN=ISP.SISPLOAD,DISP=SHR
//         DD  DSN=ISP.SISPSASC,DISP=SHR
//*
//*****************************************************************
//* ISPF LIBRARIES
//*****************************************************************
//*
//ISPMLIB  DD  DSN=ISP.SISPMXXX,DISP=SHR ISPF MSGS
//*
//ISPSLIB  DD  DSN=ISP.SISPSXXX,DISP=SHR ISPF SKELS
//         DD  DSN=ISP.SISPSLIB,DISP=SHR ISPF SKELS
//*
//ISPPLIB  DD  DSN=ISP.SISPPXXX,DISP=SHR ISPF PANELS
//*
//ISPTLIB  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
//         DD  DSN=ISP.SISPTXXX,DISP=SHR ISPF TABLES
//*
//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 DATASET
//*                                                          OW01230
//SYSTERM  DD SYSOUT=*
//*
//*---------------------------------------------------------------
//* TEMPORARY CLIST CONTAINING COMMAND TO BE EXECUTED
//*---------------------------------------------------------------
//SYSPROC  DD  DSN=&&&&CLIST&STEP,DISP=(OLD,DELETE)
//         DD  DSN=ISP.SISPCLIB,DISP=SHR          CLIST LIBRARY OW01230
//*
)CM
```

```
)CM 5655-042 (C) COPYRIGHT IBM CORP 1989, 1995 */
```

Example: A-17 DD Statements to Preallocate Data Sets

```
//ISPCTL0  DD DISP=NEW,UNIT=VIO,SPACE=(CYL,(1,1)),
//            DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)
//ISPCTL1  DD DISP=NEW,UNIT=VIO,SPACE=(CYL,(1,1)),
//            DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)
.
.
.
//ISPCTLW  DD DISP=NEW,UNIT=VIO,SPACE=(CYL,(1,1)),
//            DCB=(LRECL=80,BLKSIZE=800,RECFM=FB)
//* In the above section of JCL, there is one DD for each screen
//* defined, based on the value of keyword MAXIMUM_NUMBER_OF_
//*  SPLIT_SCREENS in the configuration table.
//* The DD name is in the form ISPCTLx, where x can be
//* 1-9, A-W.  For example, if the keyword value = 8, only
//* ISPCTL1 to ISPCTL8 need to be coded.
//* ISPCTL0 is a special case, used only by Edit for the Submit
//* command.
//ISPWRK1  DD DISP=NEW,UNIT=VIO,SPACE=(CYL,(1,1)),
//            DCB=(LRECL=256,BLKSIZE=2560,RECFM=FB)
//ISPWRK2  DD DISP=NEW,UNIT=VIO,SPACE=(CYL,(1,1)),
//            DCB=(LRECL=256,BLKSIZE=2560,RECFM=FB)
.
.
.
//ISPWRKW  DD DISP=NEW,UNIT=VIO,SPACE=(CYL,(1,1)),
//            DCB=(LRECL=256,BLKSIZE=2560,RECFM=FB)
//* In the above section of JCL, there is one DD for each screen
//* defined, based on the value of keyword MAXIMUM_NUMBER_OF_
//* SPLIT_SCREENS in the configuration table.
//* The DD name is in the form ISPWRKx, where x can be
//* 1-9, A-W.  For example, if the value of the keyword = 8,
//* only ISPWRK1 to ISPWRK8 need to be coded.
//ISPLST1  DD DISP=NEW,UNIT=VIO,SPACE=(CYL,(1,1)),
//            DCB=(LRECL=121,BLKSIZE=1210,RECFM=FBA)
//ISPLST2  DD DISP=NEW,UNIT=VIO,SPACE=(CYL,(1,1)),
//            DCB=(LRECL=121,BLKSIZE=1210,RECFM=FBA)
.
.
.
//ISPLSTW  DD DISP=NEW,UNIT=VIO,SPACE=(CYL,(1,1)),
//            DCB=(LRECL=121,BLKSIZE=1210,RECFM=FBA)
//* In the above section of JCL, there is one DD for each screen
//* defined, based on the value of keyword MAXIMUM_NUMBER_OF_
//* SPLIT_SCREENS in the configuration table.
//* The DD name is in the form ISPLSTx, where x can be
//* 1-9, A-W.  For example, if the value of the keyword = 8,
//* only ISPLST1 to ISPLST8 need to be coded.
```

Arbeo

B

Chapter 2 listings

Example: B-1 SCLM01.PROJDEFS.SOURCE(FLM@CCBE)

```
*********************************************************************
*          SCLM LANGUAGE DEFINITION FOR                             *
*          ENTERPRISE COBOL WITH INTEGRATED CICS TRANSLATOR         *
*                                                                   *
*                                                                   *
**********************  GENERAL NOTES  ******************************
*                                                                   *
*  THIS LANGUAGE DEFINITION IS AN EXAMPLE THAT CAN SERVE AS A       *
*  REFERENCE IN THE CONSTRUCTION AND CUSTOMIZATION OF LANGUAGE      *
*  DEFINITIONS FOR A PARTICULAR APPLICATION AND ENVIRONMENT.        *
*                                                                   *
********************* CUSTOMIZATION NOTES ****************************
*                                                                   *
* LIST EACH "STATIC" COPY DATASET UNDER THE FLMSYSLB MACRO.         *
* USE THE LANGUAGE NAME (VALUE OF THE LANG KEYWORD) FOR THE LABEL   *
* ON THE FIRST FLMSYSLB MACRO.                                      *
*   EXAMPLE:                                                        *
*   CICSCBE  FLMSYSLB   XXXXX.XXXXX.XXXXX.XXXXX                     *
*            FLMSYSLB   YYYYY.YYYYY.YYYYY.YYYYY                     *
*                                                                   *
* THE ALCSYSLB=Y ON THE FLMLANGL MACRO WILL AUTOMATICALLY ADD THESE *
* DATASETS TO THE CONCATENATION OF IOTYPE=I,KEYREF=SINC ALLOCATIONS. *
* CHKSYSLB=BUILD WILL DEFER CHECKING OF FLMSYSLB DATASETS UNTIL     *
* BUILD TIME.                                                       *
*                                                                   *
* CUSTOMIZE THE 'OPTIONS' AND 'GOODRC' FIELDS TO YOUR STANDARDS.    *
*                                                                   *
* ADD THE 'DSNAME' FIELD IF THE TRANSLATOR IS IN A PRIVATE LIBRARY. *
*                                                                   *
* WHEN A NEW TRANSLATOR VERSION REQUIRES TOTAL RECOMPILATION FOR THIS *
* LANGUAGE, THE 'VERSION' FIELD ON FLMLANGL SHOULD BE CHANGED.      *
*                                                                   *
*                                                                   *
```

© Copyright IBM Corp. 2007. All rights reserved.

```
*********************************************************************
*-------------------------------------------------------------------*
* CHANGE ACTIVITY:                                                  *
* FLAG=REASON DATE        ID  DESCRIPTION                           *
* ----------------------------------------------------------------- *
* $01=OA17586 2006/09/14 RX: INTEGRATED CICS COMPILER CHANGES.      *
*                                                                   *
*********************************************************************
*
CICSCBE  FLMSYSLB  CICS.TS31.CICS.SDFHCOB
         FLMSYSLB  CICS.TS31.CICS.SDFHMAC
*
         FLMLANGL  LANG=CICSCBE,VERSION=2,ALCSYSLB=Y,CHKSYSLB=BUILD,   C
               LANGDESC='ENTERPRISE COBOL WITH CICS'
*
*********************************************************************
*       --INCLUDE SET SPECIFICATION FOR EXAMPLE PROJECT SCLM01--    *
*********************************************************************
*
          FLMINCLS   TYPES=(COPYLIB)
*
*
*********************************************************************
*         --PARSER TRANSLATOR--                                     *
*********************************************************************
*
          FLMTRNSL   CALLNAM='SCLM COBOL PARSE',                     C
               FUNCTN=PARSE,                                         C
               COMPILE=FLMLPCBL,                                     C
               PORDER=1,                                             C
               OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,)      *M34FN*
*          (* SOURCE     *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)                  *M34FN*
*         - CICS PRECOMPILE -           CODE    LINES   *18@01D*
*                                       COMMENT LINES   *14@01D*
*         --COBOL INTERFACE--           CODE    LINES   *29@01D*
*                                       COMMENT LINES   *18@01D*
*
*********************************************************************
* COBOL COMPILE AND CICS PRE-PROCESS    CODE    LINES   *32@01A*
* IN ONE STEP                           COMMENT LINES   *19@01A*
*********************************************************************
*
         FLMTRNSL   CALLNAM='COBOL COMPILER WITH CICS PREPROCESS',    C
               FUNCTN=BUILD,                                         C
               COMPILE=IGYCRCTL,                                     C
               VERSION=3.3.1,                                        C
               TASKLIB=TASKLIB,                                      C
               GOODRC=0,                                             C
               PORDER=1,                                             C
               OPTIONS=(RENT,NODYNAM,LIB,CICS(''COBOL3''))
*
*********************************************************************
*         --DDNAME ALLOCATION--                                     *
```

Archived

```
*********************************************************************
*
          FLMALLOC  IOTYPE=O,KEYREF=OBJ,RECFM=FB,LRECL=80,           C
              RECNUM=5000,DFLTTYP=OBJ,DDNAME=SYSLIN
*
          FLMALLOC  IOTYPE=I,KEYREF=SINC,DDNAME=SYSLIB
*
          FLMALLOC  IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,          C
              DDNAME=SYSIN
*
          FLMALLOC  IOTYPE=O,KEYREF=LIST,                            C
              RECFM=FBA,LRECL=133,                                   C
              RECNUM=25000,PRINT=Y,DFLTTYP=LIST,DDNAME=SYSPRINT
*
          FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,          C
              DDNAME=SYSUT1
*
          FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,          C
              DDNAME=SYSUT2
*
          FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,          C
              DDNAME=SYSUT3
*
          FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,          C
              DDNAME=SYSUT4
*
          FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,          C
              DDNAME=SYSUT5
*
          FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,          C
              DDNAME=SYSUT6
*
          FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,          C
              DDNAME=SYSUT7
*         (* TASKLIB*)
          FLMALLOC  IOTYPE=A,DDNAME=TASKLIB
           FLMCPYLB  ECOBOL.V340.SIGYCOMP
           FLMCPYLB  CICS.TS31.CICS.SDFHLOAD
*
*                                                                   *
* 5694-A01     COPYRIGHT IBM CORP 1980, 2007                        *
*                                                                   *
```

Example: B-2 SCLM01.PROJDEFS.SOURCE(COBCICS)

```
*********************************************************************
*         SCLM LANGUAGE DEFINITION FOR                              *
*         ENTERPRISE COBOL WITH CICS PRECOMPILER                    *
*                                                                   *
*********************  GENERAL NOTES  ***************************** *
*                                                                   *
*  THIS LANGUAGE DEFINITION IS AN EXAMPLE THAT CAN SERVE AS A       *
*  REFERENCE IN THE CONSTRUCTION AND CUSTOMIZATION OF LANGUAGE      *
*  DEFINITIONS FOR A PARTICULAR APPLICATION AND ENVIRONMENT.        *
```

Archived

```
*                                                                     *
******************** CUSTOMIZATION NOTES ****************************
*                                                                     *
* CICS OUTPUT IS PASSED VIA THE CICSTRAN DD ALLOCATION TO OS COBOL.   *
*                                                                     *
* LIST EACH "STATIC" COPY DATASET UNDER THE FLMSYSLB MACRO.           *
* USE THE LANGUAGE NAME (VALUE OF THE LANG KEYWORD) FOR THE LABEL     *
* ON THE FIRST FLMSYSLB MACRO.                                        *
*   EXAMPLE:                                                          *
*   CICSCBE  FLMSYSLB   XXXXX.XXXXX.XXXXX                            *
*            FLMSYSLB   YYYYY.YYYYY.YYYYY                            *
*                                                                     *
* THE ALCSYSLB=Y ON THE FLMLANGL MACRO WILL AUTOMATICALLY ADD THESE   *
* DATASETS TO THE CONCATENATION OF IOTYPE=I,KEYREF=SINC ALLOCATIONS.  *
* CHKSYSLB=BUILD WILL DEFER CHECKING OF FLMSYSLB DATASETS UNTIL       *
* BUILD TIME.                                                         *
*                                                                     *
* CUSTOMIZE THE 'OPTIONS' AND 'GOODRC' FIELDS TO YOUR STANDARDS.      *
*                                                                     *
* ADD THE 'DSNAME' FIELD IF THE TRANSLATOR IS IN A PRIVATE LIBRARY.   *
*                                                                     *
* WHEN A NEW TRANSLATOR VERSION REQUIRES TOTAL RECOMPILATION FOR THIS *
* LANGUAGE, THE 'VERSION' FIELD ON FLMLANGL SHOULD BE CHANGED.        *
*                                                                     *
*                                                                     *
***********************************************************************
*
COBCICS  FLMSYSLB  CICS.TS31.CICS.SDFHCOB
         FLMSYSLB  CICS.TS31.CICS.SDFHMAC
*
         FLMLANGL  LANG=COBCICS,VERSION=1,ALCSYSLB=Y,CHKSYSLB=BUILD,   C
               LANGDESC='ENTERPRISE COBOL WITH CICS'
*
***********************************************************************
*       --INCLUDE SET SPECIFICATION FOR EXAMPLE PROJECT SCLM01--      *
***********************************************************************
*
          FLMINCLS   TYPES=(COPYLIB)
*
* PARSER TRANSLATOR
*
          FLMTRNSL   CALLNAM='SCLM COBOL PARSE',                      C
               FUNCTN=PARSE,                                          C
               COMPILE=FLMLPCBL,                                      C
               PORDER=1,                                              C
               OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,)
*          (* SOURCE      *)
          FLMALLOC  IOTYPE=A,DDNAME=SOURCE
           FLMCPYLB @@FLMDSN(@@FLMMBR)
*
* BUILD TRANSLATOR(S)
*
*         - CICS PRECOMPILE -
          FLMTRNSL   CALLNAM='CICS PRE-COMPILE',                      C
               FUNCTN=BUILD,                                          C
```

```
             COMPILE=DFHECP1$,                                          C
             DSNAME=CICS.TS31.CICS.SDFHLOAD,                            C
             VERSION=2.1,                                               C
             GOODRC=4,                                                  C
             PORDER=3,                                                  C
             OPTIONS=(SOURCE,NOSEQ)
*  1       -- N/A --
           FLMALLOC  IOTYPE=N
*  2       -- N/A --
           FLMALLOC  IOTYPE=N
*  3       -- N/A --
           FLMALLOC  IOTYPE=N
*  4       -- N/A --
           FLMALLOC  IOTYPE=N
*  5       (* SYSIN *)
           FLMALLOC  IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,          C
             DDNAME=SYSIN
*  6       (* SYSPRINT *)
           FLMALLOC  IOTYPE=O,RECFM=FBA,LRECL=121,                    C
             RECNUM=35000,PRINT=Y
*  7       (* SYSPUNCH *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,                      C
             RECNUM=5000,DDNAME=CICSTRAN
*
*
*
*        --COBOL INTERFACE--
           FLMTRNSL   CALLNAM='COBOL',                                C
             FUNCTN=BUILD,                                            C
             COMPILE=IGYCRCTL,                                        C
             DSNAME=ECOBOL.V340.SIGYCOMP,                             C
             VERSION=3.4.0,                                           C
             GOODRC=4,                                                C
             PORDER=3,                                                C
             OPTIONS=(DMA,PRI,SIZE=512K,APOS,CNT=77,BUF=30K,OPT,      C
             XREF,LIB)
*
* DDNAME ALLOCATIONS
*
*  1       (* SYSLIN *)
           FLMALLOC  IOTYPE=O,KEYREF=OBJ,RECFM=FB,LRECL=80,           C
             RECNUM=5000,DFLTTYP=OBJ
*  2       (* N/A *)
           FLMALLOC  IOTYPE=N
*  3       (* N/A *)
           FLMALLOC  IOTYPE=N
*  4       (* SYSLIB *)
           FLMALLOC  IOTYPE=I,KEYREF=SINC,DDNAME=SYSLIB
*  5       (* SYSIN *)
           FLMALLOC  IOTYPE=U,KEYREF=SINC,DDNAME=CICSTRAN
*  6       (* SYSPRINT *)
           FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=FBA,LRECL=133,        C
             RECNUM=25000,PRINT=Y,DFLTTYP=LIST
*  7       (* SYSPUNCH *)
           FLMALLOC  IOTYPE=A
```

Archived

```
            FLMCPYLB NULLFILE
*  8       (* SYSUT1 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
*  9       (* SYSUT2 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 10       (* SYSUT3 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 11       (* SYSUT4 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 12       (* SYSTERM *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
* 13       (* SYSUT5 *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
* 14       (* SYSUT6 *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
*
* 5694-A01 (C) COPYRIGHT IBM CORP 1980, 2006
*
```

Example: B-3 ALLOCDBG Listing

```
/* REXX */
/*********************************************************************/
/* ALLOCDBG EXEC                                                     */
/* WRITTEN BY: PETER ECK                                             */
/* 08/31/2006                                                        */
/*                                                                   */
/* THIS REXX ROUTINE WILL ALLOCATE SYSDEBUG                          */
/*********************************************************************/
   parse upper arg prj grp member                /* parse arguments */

   dbugdsn = "'"prj"."grp".SYSDEBUG("member")'"

   /* If sysdebug member does not exist then create it               */
   If sysdsn(dbugdsn) <> 'OK' Then
   Do
      "ALLOC DA("dbugdsn") FI(SYSDEBUG) SHR"
      out.0 = 1
      out.1 = "dummy"
      "EXECIO * DISKW SYSDEBUG (STEM OUT. FINIS)"
      If rc > 0 Then
      Do
        Say "******   Error creating the dummy sysdebug member, RC="rc
        Exit 12
      End
      "FREE F(SYSDEBUG)"
   End
Exit 0
```

Archived

Example: B-4 COPYMEMB Listing

```
/* REXX */

   Arg Type Member CallMeth

   If Type = 'SYSDEBUG' Then
   Do
      SYSLINE.1 = " COPYGRP   OUTDD=OUTDD,INDD=INDD"
      SYSLINE.2 = " SELECT    MEMBER="Member
      SYSLINE.0 = 2
      Address TSO "EXECIO * DISKW SYSIN (STEM SYSLINE. FINIS)"

      Select
         When (CallMeth = 'TSOLNK') Then
         Do
            Say "Calling IEBCOPY with CALL"

            "CALL 'SYS1.LINKLIB(IEBCOPY)'"
            If (rc > 0) then
            Do
               Say 'Filecopy failed (IEBCOPY rc='||rc||')'
               ret_code = 1
            End
         End
         When (CallMeth = 'ISPLNK') Then
         Do
            Say "Calling IEBCOPY with SELECT"

            Address ISPEXEC "ISPEXEC SELECT PGM(IEBCOPY)"
            If (rc > 0) then
            Do
               Say 'Filecopy failed (IEBCOPY rc='||rc||')'
               ret_code = 2
            End
         End
         Otherwise
            NOP
      End
   End
Exit
```

Example: B-5 LIST Language Definition

```
*************************************************************************
*         DUMMY LIST TRANSLATOR FOR SYSDEBUG COPY                       *
*                                                                       *
         FLMLANGL    LANG=LIST,                                        C
               LANGDESC='DUMMY LIST TO INVOKE SYSDEBUG COPY'
*
*************************************************************************
*         --ALLOCATE SYSDEBUG MEMBER AT PROMOTE TO GROUP IF NEW         *
*************************************************************************
*
```

Archived

```
         FLMTRNSL   CALLNAM='ALLOC SYSDEBUG',                          C
               FUNCTN=COPY,                                           C
               COMPILE=SELECT,                                        C
               DSNAME=SCLM.PROJDEFS.REXX,                             C
               CALLMETH=ISPLNK,                                       C
               GOODRC=0,                                              C
               PDSDATA=Y,                                             C
               OPTIONS='CMD(EX ''SCLM.PROJDEFS.REXX(ALLOCDBG)''       C
               ''@@FLMPRJ @@FLMTOG @@FLMMBR'')'
*
*-------------------------------------------------------------------*
*-- COPY SIDE FILE AS MEMBER IS PROMOTED                          --*
*-------------------------------------------------------------------*
          FLMTRNSL FUNCTN=COPY,                                       C
               CALLNAM='COPY SIDE FILE',                              C
               CALLMETH=TSOLNK,                                       C
               COMPILE=COPYMEMB,                                      C
               DSNAME=SCLM.PROJDEFS.REXX,                             C
               GOODRC=0,                                              C
               PDSDATA=Y,                                             C
               OPTIONS='SYSDEBUG @@FLMMBR TSOLNK'
*
           FLMALLOC  IOTYPE=A,DDNAME=INDD,DISP=SHR
            FLMCPYLB @@FLMPRJ.@@FLMGRP.SYSDEBUG(@@FLMMBR)
*
           FLMALLOC  IOTYPE=A,DDNAME=OUTDD,DISP=SHR
            FLMCPYLB @@FLMPRJ.@@FLMTOG.SYSDEBUG(@@FLMMBR)
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSIN,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSPRINT,RECNUM=50000
*
* 5694-A01 (C) COPYRIGHT IBM CORP 1990, 2006                        *
*                                                                   *
```

Example: B-6 COBDTC Enhanced Language Definition

```
*********************************************************************
*         ENTERPRISE COBOL LANGUAGE DEFINITION FOR SCLM             *
*                                                                   *
*                                                                   *
*********************  GENERAL NOTES  ********************************
*                                                                   *
*  THIS LANGUAGE DEFINITION IS AN EXAMPLE THAT CAN SERVE AS A       *
*  REFERENCE IN THE CONSTRUCTION AND CUSTOMIZATION OF LANGUAGE      *
*  DEFINITIONS FOR A PARTICULAR APPLICATION AND ENVIRONMENT.        *
*                                                                   *
******************** CUSTOMIZATION NOTES *****************************
*                                                                   *
* LIST EACH "STATIC" COPY DATASET UNDER THE FLMSYSLB MACRO.         *
* USE THE LANGUAGE NAME (VALUE OF THE LANG KEYWORD) FOR THE LABEL   *
* ON THE FIRST FLMSYSLB MACRO.                                      *
*   EXAMPLE:                                                        *
*    COBE    FLMSYSLB   XXXXX.XXXXX.XXXXX                          *
```

```
*            FLMSYSLB   YYYYY.YYYYY.YYYYY                              *
*                                                                       *
* CUSTOMIZE THE 'OPTIONS' AND 'GOODRC' FIELDS TO YOUR STANDARDS.        *
*                                                                       *
* ADD THE 'DSNAME' FIELD IF THE TRANSLATOR IS IN A PRIVATE LIBRARY.     *
*                                                                       *
* CHANGING THE VERSION FIELD ON FLMLANGL WILL CAUSE ALL MEMBERS TO BE   *
* OUT OF DATE.  EACH MEMBER WILL BE RE-BUILT THE NEXT TIME BUILD IS     *
* INVOKED.                                                              *
*                                                                       *
* IF YOU DON'T WANT A COMPILER LISTING, CHANGE THE DDNAME=SYSPRINT      *
* TO AN IOTYPE=W AND REMOVE THE UNNECESSARY PARAMETERS.                 *
*                                                                       *
* THIS LANGUAGE DEFINITION IS COMPATIBLE WITH THE AD/CYCLE COBOL        *
* COMPILER.                                                             *
*                                                                       *
*************************************************************************
*-----------------------------------------------------------------------*
* CHANGE ACTIVITY:                                                      *
* FLAG=REASON  DATE       ID  DESCRIPTION                               *
* ----------------------------------------------------------------------*
* $01 M42P2691 1994/04/26 - : CHANGED DFLTTYP=COMPLIST TO LIST TO       *
*             MATCH TYPES DEFINED IN EXAMPLE PROJECTS.                  *
* $02 OW03966             - : V4.1 DEVELOPMENT APAR                     *
* $03 OA17586  2006/09/29 RX: REMOVED COB2 COMMENTS.                    *
*                                                                       *
*                                                                       *
*************************************************************************
*
* THE LINE LENGTH IS 80 CHARACTERS AND SEQUENCE NUMBERS ARE PLACED
* IN COLUMNS 1 TO 6 AND 73 TO 80.  THESE ARE ONLY USED BY WSP/2 1.2
* AND HIGHER.  SCLM IGNORES THESE PARAMETERS.
*
       FLMLANGL    LANG=COBEDTC,ALCSYSLB=Y,                           C
             LANGDESC='ENTERPRISE COBOL WITH DEBUG TOOL'
*
*************************************************************************
*        --PARSER TRANSLATOR--                                          *
*************************************************************************
*
        FLMTRNSL   CALLNAM='SCLM COBOL PARSE',                          C
             FUNCTN=PARSE,                                              C
             COMPILE=FLMLPCBL,                                          C
             PORDER=1,                                                  C
             CALLMETH=LINK,                                             C
             OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,)
*         (* SOURCE      *)
         FLMALLOC  IOTYPE=A,DDNAME=SOURCE
          FLMCPYLB @@FLMDSN(@@FLMMBR)
*
*************************************************************************
*        --ALLOCATE SYSDEBUG MEMBER IF NEW                              *
*************************************************************************
*
        FLMTRNSL   CALLNAM='ALLOC SYSDEBUG',                            C
```

```
               FUNCTN=BUILD,                                           C
               COMPILE=SELECT,                                         C
               DSNAME=SCLM.PROJDEFS.REXX,                              C
               CALLMETH=ISPLNK,                                        C
               GOODRC=0,                                               C
               PORDER=1,                                               C
               OPTIONS='CMD(EX ''SCLM.PROJDEFS.REXX(ALLOCDBG)''        C
               ''@@FLMPRJ @@FLMGRP @@FLMMBR'')'
*
*********************************************************************
*         --ENTERPRISE COBOL INTERFACE--                            *
*********************************************************************
*
          FLMTRNSL   CALLNAM='ENTERPRISE COBOL COMPILER',              C
               FUNCTN=BUILD,                                           C
               COMPILE=IGYCRCTL,                                       C
               DSNAME=ECOBOL.V340.SIGYCOMP,                            C
               VERSION=3.1,                                            C
               GOODRC=0,                                               C
               PORDER=1,                                               C
               OPTIONS=(XREF,LIB,APOST,NODYNAM,LIST,NONUMBER,NOSEQ,TESTC
               (ALL,SYM,SEP))
*
*********************************************************************
*         --DDNAME ALLOCATION--                                     *
*********************************************************************
*
           FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,                C
               RECNUM=5000,DFLTTYP=OBJ
*
           FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
*
           FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT2,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT3,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT4,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT5,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT6,RECNUM=5000
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT7,RECNUM=5000
*
           FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
            FLMCPYLB NULLFILE
*
           FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
            FLMCPYLB NULLFILE
*
           FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,KEYREF=LIST,             C
```

Archived

```
               RECFM=FBA,LRECL=133,LANG=LIST,                          C
               RECNUM=50000,PRINT=Y,DFLTTYP=LIST
*
          FLMALLOC  IOTYPE=A,DDNAME=SYSDEBUG,DISP=SHR
           FLMCPYLB @@FLMPRJ.@@FLMGRB.SYSDEBUG(@@FLMMBR)
*
```

Archived

Archived

C

Chapter 3 listings

Common bind exec example

Example: C-1 Common bind exec example

```
/* REXX */

   trace o
   Arg OPT GRP MEMBER EXP ENV

   /*  Create a bind control statement as a single long line.  Nothing
       Else seems to work  */

   rcode = 0

   /* Set standard BIND options */

   FLAG      = 'I'
   ISOLATION = 'CS'
   CACHESIZE = '256'
   ACQUIRE   = 'ALLOCATE'
   RELEASE   = 'DEALLOCATE'

   Select
      When (GRP = 'PROD') Then
      Do
         SUBSYS    = 'DI11'
         OWNER     = 'PRODDBA'
         ACTION    = 'REP'
         VALIDATE  = 'RUN'
         ISOLATION = 'CS'
         EXPLAIN   = 'NO'
         QUALIFIER = 'PRODDBA'
         PKLIST    = 'PROD.*'

         Call Bind_it
```

© Copyright IBM Corp. 2007. All rights reserved.

```
      End

      When (GRP = 'TEST') Then
      Do
         SUBSYS    = 'DI11'
         OWNER     = 'TESTDBA'
         ACTION    = 'REP'
         VALIDATE  = 'BIND'
         ISOLATION = 'CS'
         EXPLAIN   = 'NO'
         QUALIFIER = 'TESTDBA'
         PKLIST    = 'TEST.*'

         Call Bind_it
      End

      When (GRP = 'DEV1') Then
      Do
         SUBSYS    = 'DI11'
         OWNER     = 'DEVDBA'
         ACTION    = 'REP'
         VALIDATE  = 'BIND'
         ISOLATION = 'CS'
         EXPLAIN   = EXP
         QUALIFIER = 'DEVDBA'
         PKLIST    = 'DEV1.*'

         Call Bind_it
      End
      When (GRP = 'DEV2') Then
         NOP         /* no bind needed */
      Otherwise
         NOP         /* no bind needed */
   End

Exit RCODE

Bind_it:

   Select
      When (OPT = 'PLANBIND') Then
      Do
         DB2_Line = "BIND PLAN("MEMBER")"          ||,
                    " OWNER("OWNER")"              ||,
                    " QUALIFIER("QUALIFIER")"      ||,
                    " PKLIST("PKLIST")"            ||,
                    " VALIDATE("VALIDATE")"        ||,
                    " ISOLATION("ISOLATION")"      ||,
                    " EXPLAIN("EXPLAIN")"          ||,
                    " FLAG("FLAG")"                ||,
                    " CACHESIZE("CACHESIZE")"      ||,
                    " ACQUIRE("ACQUIRE")"          ||,
                    " RELEASE("RELEASE")"          ||,
                    " RETAIN"                      ||,
                    " ENABLE("ENV")"
```

Archived

```
                say DB2_line

   /*  Write the bind control statement to the data queue and execute  */
   /*  DB2I to perform the bind.                                       */

   queue DB2_Line
   queue "End"
   Address TSO "DSN SYSTEM("SUBSYS")"
   rcode = RC
End
When (OPT = 'PLANFREE') Then
Do
   /*  At this point SCLM passes the from group information so that    */
   /*  the plan at the from group can be FREEed if required           */
End
When (OPT = 'PACKBIND') Then
Do
   DB2_Line = "BIND PACKAGE("GRP") MEMBER("MEMBER")" ||,
              " OWNER("OWNER")"          ||,
              " ACTION("ACTION")"        ||,
              " VALIDATE("VALIDATE")"    ||,
              " ISOLATION("ISOLATION")"  ||,
              " EXPLAIN("EXPLAIN")"      ||,
              " QUALIFIER("QUALIFIER")"


   /*  Write the bind control statement to the data queue and execute  */
   /*  DB2I to perform the bind.                                       */

   queue DB2_Line
   queue "End"
   Address TSO "DSN SYSTEM("SUBSYS")"
   rcode = RC
End
When (OPT = 'PACKFREE') Then
Do
   /*  At this point SCLM passes the from group information so that    */
   /*  the package at the from group can be FREEed if required        */
End
Otherwise
   NOP
End

Return
```

Archived

Example DB2CLIST language definition

Example: C-2 SCLM07.PROJDEFS.SOURCE(PKGBIND)

```
************************************************************************
* FLM@BD2 - LANGUAGE DEFINITION FOR DB2 BIND/FREE CLIST                *
*                                                                      *
*********************  GENERAL NOTES  ******************************
*                                                                      *
*  THIS LANGUAGE DEFINITION IS AN EXAMPLE THAT CAN SERVE AS A          *
*  REFERENCE IN THE CONSTRUCTION AND CUSTOMIZATION OF LANGUAGE         *
*  DEFINITIONS FOR A PARTICULAR APPLICATION AND ENVIRONMENT.           *
*                                                                      *
********************* CHANGE ACTIVITY ******************************
*                                                                      *
*  Change activity =                                                   *
*    Pn = Reason    Release   Date    Origin   Comment                 *
*    --   --------  --------  ------  ------ : -------------------     *
*   $L0 = OY29671   M320      900730  432273 : SCLM CSP/DB2 SPE        *
*                                                                      *
*   OY39985 - 910218 - DB2 PROMOTE PROBLEM.  DB2OUT AND DSNTRACE       *
*                      ADDED.  PROMOTE TRANSLATORS REMOVED.  CALLMETH *
*                      = LINK ADDED.  GT4045 - AAB                     *
*                                                                      *
*  OW03966 - V4.1 DEVELOPMENT APAR                                     *
* @01 Z/OS 1.8 - ADDED LANGUAGE DESCRIPTION                            *
*                                                                      *
*                                                                      *
*
         FLMLANGL    LANG=PKGBIND,                                    C
               LANGDESC='DB2 BIND/FREE PACKAGE TRANSLATOR'
*
* INCLUDE-SET TO ALLOW DBRMs AND DB2 CLISTs WITH THE SAME NAME
*
         FLMINCLS    TYPES=(DBRM)
*
* PARSER TRANSLATOR
*
          FLMTRNSL   CALLNAM='PARSE DB2 CLIST',                       C
               FUNCTN=PARSE,                                          C
               COMPILE=FLMLSS,                                        C
               PORDER=1,                                              C
               OPTIONS=(PTABLEDD=,                                    C
               SOURCEDD=SOURCE,                                       C
               TBLNAME=FLMPDBRM,                                      C
               STATINFO=@@FLMSTP,                                     C
               LISTINFO=@@FLMLIS,                                     C
               LISTSIZE=@@FLMSIZ,                                     C
               CONTIN=0,                                              C
               EOLCOL=72)
*           (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
* BUILD TRANSLATOR(S)
```

Archived

```
*
          FLMTRNSL   CALLNAM='DB2 BIND',                                C
               FUNCTN=BUILD,                                           C
               COMPILE=FLMCSPDB,                                       C
               PORDER=1,                                               C
               GOODRC=4,                                               C
               CALLMETH=LINK,                                          C
               OPTIONS=(FUNCTN=BUILD,                                  C
               OPTION=BIND,                                            C
               SCLMINFO=@@FLMINF,                                      C
               PROJECT=@@FLMPRJ,                                       C
               ALTPROJ=@@FLMALT,                                       C
               DBRMTYPE=DBRM,                                          C
               GROUP=@@FLMGRP,                                         C
               MEMBER=@@FLMMBR)
*  1       -- ISRPROXY --
           FLMALLOC  IOTYPE=S,DDNAME=ISRPROXY,KEYREF=SINC,             C
               CATLG=Y,RECNUM=5000,LRECL=80,RECFM=FB
***** The following lines added for APAR OY39985 **********************
*  2       -- ISRDB2OT --
           FLMALLOC  IOTYPE=O,DDNAME=ISRDB2OT,KEYREF=OUT3,LANG=PKGOUT, C
               RECNUM=5000,LRECL=80,RECFM=FB,DFLTTYP=PKGOUT
*  3       -- DSNTRACE --
           FLMALLOC  IOTYPE=W,DDNAME=DSNTRACE,PRINT=I,                 C
               RECFM=F,LRECL=132,BLKSIZE=132
***** The preceding lines added for APAR OY39985 **********************
*  4       -- DUMMY DD - REQUIRED FOR FLMINCLS MACRO BUT NOT TRANSLATOR
           FLMALLOC  IOTYPE=I,DDNAME=DUMMYDD,KEYREF=SINC
*
* 5694-A01 (C) COPYRIGHT IBM CORP 1990, 2006
```

Archived

Example DB2OUT language definition

Example: C-3 SCLM07.PROJDEFS.SOURCE(PKGOUT)

```
************************************************************************
* FLM@BDO - LANGUAGE DEFINITION FOR DB2 BIND/FREE CLIST OUTPUT        *
*                                                                      *
*********************  GENERAL NOTES  ******************************
*                                                                      *
*  THIS LANGUAGE DEFINITION IS AN EXAMPLE THAT CAN SERVE AS A          *
*  REFERENCE IN THE CONSTRUCTION AND CUSTOMIZATION OF LANGUAGE         *
*  DEFINITIONS FOR A PARTICULAR APPLICATION AND ENVIRONMENT.           *
*                                                                      *
********************** CHANGE ACTIVITY ******************************
*                                                                      *
*  Change activity =                                                   *
*    Pn = Reason    Release   Date    Origin   Comment                 *
*    --   --------  --------  ------  ------ : ------------------      *
*   $M1 = OY39985   M320      901219  482212 : DB2 Promote - New Lang.*
*                                              def. macro added.       *
*   @01 = DEVL      z/OS 1.8  050516         : Add Language Descript   *
*                                                                      *
************************************************************************
*
         FLMLANGL    LANG=PKGOUT,                                      C
               LANGDESC='DB2 BIND/FREE PACKAGE TRANSLATOR'
*
* PROMOTE TRANSLATOR(S)
*
          FLMTRNSL   CALLNAM='DB2 PROM BIND',                          C
               FUNCTN=COPY,                                            C
               COMPILE=FLMCSPDB,                                       C
               PORDER=1,                                               C
               GOODRC=4,                                               C
               CALLMETH=LINK,                                          C
               PDSDATA=Y,                                              C
               OPTIONS=(FUNCTN=@@FLMFNM,                               C
               OPTION=BIND,                                            C
               SCLMINFO=@@FLMINF,                                      C
               PROJECT=@@FLMPRJ,                                       C
               ALTPROJ=@@FLMALT,                                       C
               DBRMTYPE=DBRM,                                          C
               GROUP=@@FLMGRP,                                         C
               TOGROUP=@@FLMTOG,                                       C
               MEMBER=@@FLMMBR)
*  1        -- ISRPROXY --
          FLMALLOC  IOTYPE=A,DDNAME=ISRPROXY
           FLMCPYLB @@FLMDSF
*  2        -- DSNTRACE --
          FLMALLOC  IOTYPE=W,DDNAME=DSNTRACE,PRINT=I,                  C
               RECFM=F,LRECL=132,BLKSIZE=132
*
          FLMTRNSL   CALLNAM='DB2 PROM FREE',                          C
               FUNCTN=PURGE,                                           C
               COMPILE=FLMCSPDB,                                       C
```

Archived

```
               PORDER=1,                                               C
               GOODRC=4,                                               C
               CALLMETH=LINK,                                          C
               PDSDATA=Y,                                              C
               OPTIONS=(FUNCTN=@@FLMFNM,                               C
               OPTION=FREE,                                            C
               SCLMINFO=@@FLMINF,                                      C
               PROJECT=@@FLMPRJ,                                       C
               ALTPROJ=@@FLMALT,                                       C
               DBRMTYPE=DBRM,                                          C
               GROUP=@@FLMGRP,                                         C
               TOGROUP=@@FLMTOG,                                       C
               MEMBER=@@FLMMBR)
*  1       -- ISRPROXY --
           FLMALLOC  IOTYPE=A,DDNAME=ISRPROXY
            FLMCPYLB @@FLMDSF
*  2       -- DSNTRACE --
           FLMALLOC  IOTYPE=W,DDNAME=DSNTRACE,PRINT=I,                 C
               RECFM=F,LRECL=132,BLKSIZE=132
*
* 5694-A01 (C) COPYRIGHT IBM CORP 1990, 2006
```

Archived

Multi-step version of COBOL with CICS and DB2 language definition

Example: C-4 COBCICD2 - multi-step version

```
*********************************************************************
*          SCLM LANGUAGE DEFINITION FOR                             *
*          ENTERPRISE COBOL WITH CICS AND DB2 PREPROCESSOR          *
*                                                                   *
**********************  GENERAL NOTES  ******************************
*                                                                   *
*  THIS LANGUAGE DEFINITION IS AN EXAMPLE THAT CAN SERVE AS A       *
*  REFERENCE IN THE CONSTRUCTION AND CUSTOMIZATION OF LANGUAGE      *
*  DEFINITIONS FOR A PARTICULAR APPLICATION AND ENVIRONMENT.        *
*                                                                   *
********************* CUSTOMIZATION NOTES ***************************
*                                                                   *
* CICS OUTPUT IS PASSED VIA THE CICSTRAN DD ALLOCATION TO COBOL.    *
* LIST EACH "STATIC" COPY DATASET UNDER THE FLMSYSLB MACRO.         *
* USE THE LANGUAGE NAME (VALUE OF THE LANG KEYWORD) FOR THE LABEL   *
* ON THE FIRST FLMSYSLB MACRO.                                      *
*   EXAMPLE:                                                        *
*   CICSCBE  FLMSYSLB   XXXXX.XXXXX.XXXXX                          *
*            FLMSYSLB   YYYYY.YYYYY.YYYYY                          *
*                                                                   *
* THE ALCSYSLB=Y ON THE FLMLANGL MACRO WILL AUTOMATICALLY ADD THESE *
* DATASETS TO THE CONCATENATION OF IOTYPE=I,KEYREF=SINC ALLOCATIONS. *
* CHKSYSLB=BUILD WILL DEFER CHECKING OF FLMSYSLB DATASETS UNTIL     *
* BUILD TIME.                                                       *
*                                                                   *
* CUSTOMIZE THE 'OPTIONS' AND 'GOODRC' FIELDS TO YOUR STANDARDS.    *
*                                                                   *
* ADD THE 'DSNAME' FIELD IF THE TRANSLATOR IS IN A PRIVATE LIBRARY. *
*                                                                   *
* WHEN A NEW TRANSLATOR VERSION REQUIRES TOTAL RECOMPILATION FOR THIS *
* LANGUAGE, THE 'VERSION' FIELD ON FLMLANGL SHOULD BE CHANGED.      *
*                                                                   *
*                                                                   *
*********************************************************************
*
COBCICD2 FLMSYSLB  CICS.TS31.CICS.SDFHCOB
         FLMSYSLB  CICS.TS31.CICS.SDFHMAC
*
         FLMLANGL  LANG=COBCICD2,VERSION=2,ALCSYSLB=Y,CHKSYSLB=BUILD,  C
               LANGDESC='ENTERPRISE COBOL WITH CICS AND DB2'
*
         FLMINCLS  TYPES=(COPYLIB,DCLGENC)
DCLGENC  FLMINCLS  TYPES=(DCLGENC)
*
* PARSER TRANSLATOR
*
          FLMTRNSL   CALLNAM='SCLM COBOL PARSE',                    C
               FUNCTN=PARSE,                                        C
               COMPILE=FLMLPCBL,                                    C
               PORDER=1,                                            C
```

Archived

```
             OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,SQL=DCLGENC)
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
* BUILD TRANSLATORS
*
*         --DB2 PREPROCESSOR INTERFACE--
           FLMTRNSL   CALLNAM='DB2 PREPROCESS',                          C
                 FUNCTN=BUILD,
                 COMPILE=DSNHPC,                                         C
                 DSNAME=DB2.V810.SDSNLOAD,                               C
                 VERSION=8.1.0,                                          C
                 GOODRC=4,                                               C
                 PORDER=3,                                               C
                 OPTIONS=(HOST(COB2),APOST)
*  1        -- N/A --
           FLMALLOC  IOTYPE=N
*  2        -- N/A --
           FLMALLOC  IOTYPE=N
*  3        -- N/A --
           FLMALLOC  IOTYPE=N
*  4        -- SYSLIB --
           FLMALLOC  IOTYPE=I,KEYREF=SINC,INCLS=DCLGENC
*  5        -- SYSIN --
           FLMALLOC  IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,             C
                 RECNUM=5000
*  6        -- SYSPRINT --
           FLMALLOC  IOTYPE=W,RECFM=FBA,LRECL=133,                       C
                 RECNUM=35000,PRINT=Y
*  7        -- N/A --
           FLMALLOC  IOTYPE=N
*  8        -- SYSUT1 --
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=800,RECNUM=9000
*  9        -- SYSUT2 --
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=800,RECNUM=9000
* 10        -- SYSUT3 --
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=800,RECNUM=9000
* 11        -- N/A --
           FLMALLOC  IOTYPE=N
* 12        -- SYSTERM --
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
* 13        -- N/A --
           FLMALLOC  IOTYPE=N
* 14        -- SYSCIN --
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,                         C
                 RECNUM=9000,DDNAME=DB2TRANS
* 15        -- N/A --
           FLMALLOC  IOTYPE=N
* 16        -- DBRMLIB--
           FLMALLOC  IOTYPE=P,DDNAME=DBRMLIB,MEMBER=@@FLMONM,            C
                 DFLTTYP=DBRM,KEYREF=OUT1,                               C
                 RECFM=FB,LRECL=80,RECNUM=5000,DIRBLKS=1
*
```

Archived

```
* BUILD TRANSLATOR(S)
*
*         - CICS PRECOMPILE -
          FLMTRNSL   CALLNAM='CICS PRE-COMPILE',                        C
               FUNCTN=BUILD,                                            C
               COMPILE=DFHECP1$,                                        C
               DSNAME=CICS.TS31.CICS.SDFHLOAD,                          C
               VERSION=3.1,                                             C
               GOODRC=4,                                                C
               PORDER=3,                                                C
               OPTIONS=(SOURCE,NOSEQ)
*  1       -- N/A --
           FLMALLOC  IOTYPE=N
*  2       -- N/A --
           FLMALLOC  IOTYPE=N
*  3       -- N/A --
           FLMALLOC  IOTYPE=N
*  4       -- N/A --
           FLMALLOC  IOTYPE=N
*  5       (* SYSIN *)
           FLMALLOC  IOTYPE=U,DDNAME=DB2TRANS
*  6       (* SYSPRINT *)
           FLMALLOC  IOTYPE=O,RECFM=FBA,LRECL=121,                      C
               RECNUM=35000,PRINT=Y
*  7       (* SYSPUNCH *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,                        C
               RECNUM=5000,DDNAME=CICSTRAN
*
*
*         --COBOL INTERFACE--
          FLMTRNSL   CALLNAM='COBOL',                                   C
               FUNCTN=BUILD,                                            C
               COMPILE=IGYCRCTL,                                        C
               DSNAME=ECOBOL.V340.SIGYCOMP,                             C
               VERSION=3.4.0,                                           C
               GOODRC=4,                                                C
               PORDER=3,                                                C
               OPTIONS=(NONUM,MAP,OFFSET,NOOPTIMIZE,                    C
               XREF,LIB)
*
* DDNAME ALLOCATIONS
*
*  1       (* SYSLIN *)
           FLMALLOC  IOTYPE=O,KEYREF=OBJ,RECFM=FB,LRECL=80,             C
               RECNUM=5000,DFLTTYP=OBJ
*  2       (* N/A *)
           FLMALLOC  IOTYPE=N
*  3       (* N/A *)
           FLMALLOC  IOTYPE=N
*  4       (* SYSLIB *)
           FLMALLOC  IOTYPE=I,KEYREF=SINC,DDNAME=SYSLIB
*  5       (* SYSIN *)
           FLMALLOC  IOTYPE=U,KEYREF=SINC,DDNAME=CICSTRAN
*  6       (* SYSPRINT *)
           FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=FBA,LRECL=133,          C
```

Archived

```
                RECNUM=25000,PRINT=Y,DFLTTYP=LIST
*  7        (* SYSPUNCH *)
            FLMALLOC  IOTYPE=A
             FLMCPYLB NULLFILE
*  8        (* SYSUT1 *)
            FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
*  9        (* SYSUT2 *)
            FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 10        (* SYSUT3 *)
            FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 11        (* SYSUT4 *)
            FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 12        (* SYSTERM *)
            FLMALLOC  IOTYPE=A
             FLMCPYLB NULLFILE
* 13        (* SYSUT5 *)
            FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 14        (* SYSUT6 *)
            FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 15        (* SYSUT7 *)
            FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* 16        (* SYSADATA *)
            FLMALLOC  IOTYPE=A
             FLMCPYLB NULLFILE
* 17        (* SYSJAVA *)
            FLMALLOC  IOTYPE=A
             FLMCPYLB NULLFILE
* 18        (* SYSDEBUG *)
            FLMALLOC  IOTYPE=A
             FLMCPYLB NULLFILE
* 19        (* SYSMDECK *)
            FLMALLOC  IOTYPE=A
             FLMCPYLB NULLFILE
*
* 5694-A01 (C) COPYRIGHT IBM CORP 1980, 2006
*
```

Archived

Single-step version of COBOL with CICS and DB2 language definition

Example: C-5 COBCICD2 - single-step version

```
************************************************************************
*          SCLM LANGUAGE DEFINITION FOR                                *
*          ENTERPRISE COBOL WITH INTEGRATED CICS TRANSLATOR            *
*                                                                      *
*                                                                      *
*********************  GENERAL NOTES  ******************************
*                                                                      *
*  THIS LANGUAGE DEFINITION IS AN EXAMPLE THAT CAN SERVE AS A          *
*  REFERENCE IN THE CONSTRUCTION AND CUSTOMIZATION OF LANGUAGE         *
*  DEFINITIONS FOR A PARTICULAR APPLICATION AND ENVIRONMENT.           *
*                                                                      *
******************** CUSTOMIZATION NOTES ****************************
*                                                                      *
* LIST EACH "STATIC" COPY DATASET UNDER THE FLMSYSLB MACRO.            *
* USE THE LANGUAGE NAME (VALUE OF THE LANG KEYWORD) FOR THE LABEL      *
* ON THE FIRST FLMSYSLB MACRO.                                         *
*   EXAMPLE:                                                           *
*   CICSCBE  FLMSYSLB   XXXXX.XXXXX.XXXXX.XXXXX                       *
*            FLMSYSLB   YYYYY.YYYYY.YYYYY.YYYYY                       *
*                                                                      *
* THE ALCSYSLB=Y ON THE FLMLANGL MACRO WILL AUTOMATICALLY ADD THESE    *
* DATASETS TO THE CONCATENATION OF IOTYPE=I,KEYREF=SINC ALLOCATIONS.   *
* CHKSYSLB=BUILD WILL DEFER CHECKING OF FLMSYSLB DATASETS UNTIL        *
* BUILD TIME.                                                          *
*                                                                      *
* CUSTOMIZE THE 'OPTIONS' AND 'GOODRC' FIELDS TO YOUR STANDARDS.       *
*                                                                      *
* ADD THE 'DSNAME' FIELD IF THE TRANSLATOR IS IN A PRIVATE LIBRARY.    *
*                                                                      *
* WHEN A NEW TRANSLATOR VERSION REQUIRES TOTAL RECOMPILATION FOR THIS  *
* LANGUAGE, THE 'VERSION' FIELD ON FLMLANGL SHOULD BE CHANGED.         *
*                                                                      *
*                                                                      *
************************************************************************
*----------------------------------------------------------------------*
* CHANGE ACTIVITY:                                                     *
* FLAG=REASON DATE       ID  DESCRIPTION                               *
* ---------------------------------------------------------------------*
* $01=0A17586  2006/09/14 RX: INTEGRATED CICS TRANSLATOR CHANGES.      *
*                                                                      *
*                                                                      *
************************************************************************
*
COBCICD2 FLMSYSLB  CICS.TS31.CICS.SDFHCOB               *  @01C*
         FLMSYSLB  CICS.TS31.CICS.SDFHMAC               *  @01C*
*
         FLMLANGL  LANG=COBCICD2,VERSION=2,ALCSYSLB=Y,CHKSYSLB=BUILD,  C
               LANGDESC='ENTERPRISE COBOL WITH CICS AND DB2'
*
         FLMINCLS TYPES=(SOURCE,COPYLIB,DCLGENC)
```

```
DCLGENC  FLMINCLS TYPES=(DCLGENC)
*
*********************************************************************
*         --PARSER TRANSLATOR--                                     *
*********************************************************************
*
          FLMTRNSL   CALLNAM='SCLM COBOL PARSE',                      C
               FUNCTN=PARSE,                                          C
               COMPILE=FLMLPCBL,                                      C
               PORDER=1,                                              C
               OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,SQL=DCLGENC,INCLSET)
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)                    *M34FN*
*         - CICS PRECOMPILE -            CODE    LINES   *18@01D*
*                                        COMMENT LINES   *14@01D*
*         --COBOL INTERFACE--            CODE    LINES   *29@01D*
*                                        COMMENT LINES   *18@01D*
*
*********************************************************************
* COBOL COMPILE AND CICS PRE-PROCESS     CODE    LINES   *32@01A*
* IN ONE STEP                            COMMENT LINES   *19@01A*
*********************************************************************
*
          FLMTRNSL   CALLNAM='COBOL COMPILER WITH CICS PREPROCESS',   C
               FUNCTN=BUILD,                                          C
               COMPILE=IGYCRCTL,                                      C
               VERSION=1.0,                                           C
               TASKLIB=TASKLIB,                                       C
               GOODRC=4,                                              C
               PORDER=1,                                              C
               OPTIONS=(NONUM,LIB,XREF(FULL),MAP,OFFSET,NOOPTIMIZE,CICSC
               (''COBOL3''),SQL)
*
*********************************************************************
*         --DDNAME ALLOCATION--                                     *
*********************************************************************
*
           FLMALLOC  IOTYPE=O,KEYREF=OBJ,RECFM=FB,LRECL=80,           C
               RECNUM=5000,DFLTTYP=OBJ,DDNAME=SYSLIN
*
           FLMALLOC  IOTYPE=I,KEYREF=SINC,DDNAME=SYSLIB
*
           FLMALLOC  IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,          C
               DDNAME=SYSIN
*
           FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=FBA,LRECL=133,        C
               RECNUM=25000,PRINT=Y,DFLTTYP=LIST,DDNAME=SYSPRINT
*
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,          C
               DDNAME=SYSUT1
*
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,          C
               DDNAME=SYSUT2
*
```

Archived

```
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,           C
               DDNAME=SYSUT3
*
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,           C
               DDNAME=SYSUT4
*
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,           C
               DDNAME=SYSUT5
*
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,           C
               DDNAME=SYSUT6
*
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000,           C
               DDNAME=SYSUT7
*
           FLMALLOC  IOTYPE=P,DDNAME=DBRMLIB,MEMBER=@@FLMONM,          C
               DFLTTYP=DBRM,KEYREF=OUT1,                               C
               RECFM=FB,LRECL=80,RECNUM=5000,DIRBLKS=1
*
           FLMALLOC  IOTYPE=I,DDNAME=DCLGENC,KEYREF=SINC,INCLS=DCLGENC
*
*          (* TASKLIB*)
           FLMALLOC  IOTYPE=A,DDNAME=TASKLIB
            FLMCPYLB  ECOBOL.V340.SIGYCOMP
            FLMCPYLB  CICS.TS31.CICS.SDFHLOAD
            FLMCPYLB  DB2.V810.SDSNLOAD
            FLMCPYLB  DB2.V810.SDSNEXIT
*                                                                      *
*                                                                      *
* 5694-A01 (C) COPYRIGHT IBM CORP 1980, 2006                           *
*                                                                      *
```

Archived

PLI with DB2 language definition

Example: C-6 PLEDB2 - single-step version

```
*********************************************************************
*        SAMPLE    SCLM LANGUAGE DEFINITION FOR                     *
*                  PL/I WITH INTEGRAL DB2 PRE-PROCESSING            *
*                                                                   *
*        ENTERPRISE PL/I AND DB2 V7.1 AND HIGHER                    *
*                                                                   *
*        UTILISE THE TASKLIB DDNAME IF BOTH YOUR PL/I COMPILER AND  *
*        DB2 PRE-PROCESSOR DO NOT RESIDE IN THE LINKLIST AS YOU CAN *
*        ONLY SPECIFY ONE LIBRARY ON THE DSNAME PARAMETER           *
*                                                                   *
*  NOTE: IF YOU DO NOT WANT DB2 PRE-PROCESSING, REMOVE THE SQL OPTION *
*        FROM THE PARMS AND COMMENT OUT THE DBRMLIB FLMALLOC        *
*                                                                   *
*                                                                   *
*********************  GENERAL NOTES  ***************************
*-------------------------------------------------------------------*
* CHANGE ACTIVITY:                                                  *
* FLAG=REASON  DATE      ID  DESCRIPTION                            *
*-------------------------------------------------------------------*
* $01 OA04093  2003/08/05   : NEW RELEASE                           *
* $02          2005/05/16   : ADDED LANGUAGE DESCRIPTION            *
* $03 OA17586  2006/09/29 RX: INTEGRATED COMPILER.                  *
*              ADDED DB2*.**.SDSNEXIT TO THE TASKLIB TO INCLUDE     *
*              CORRECT DSNHDECP MODULE.                             *
*                                                                   *
*********************************************************************
         FLMLANGL    LANG=PLEDB2,VERSION=1,ALCSYSLB=Y,CHKSYSLB=BUILD,  C
              LANGDESC='PLI ENTERPRIZE WITH DB2'
*
         FLMINCLS TYPES=(PLI,PLINCL,DCLGENP)
DB2DCLS  FLMINCLS TYPES=(DCLGENP)
*
*********************************************************************
*         --PARSER TRANSLATOR--                                     *
*********************************************************************
*
          FLMTRNSL   CALLNAM='SCLM PL/I PARSE',                      C
              FUNCTN=PARSE,                                          C
              COMPILE=FLMLPGEN,                                      C
              PORDER=1,                                              C
              OPTIONS=(SOURCEDD=SOURCE,                              C
              STATINFO=@@FLMSTP,                                     C
              LISTINFO=@@FLMLIS,                                     C
              LISTSIZE=@@FLMSIZ,                                     C
              LANG=I(I))
*           (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
*********************************************************************
*         --ENTERPRISE PL/I--                                       *
```

Archived

```
*********************************************************************
*
          FLMTRNSL   CALLNAM='ENTERPRISE PL/I COMPILER',              C
               FUNCTN=BUILD,                                          C
               COMPILE=IBMZPLI,                                       C
               TASKLIB=TASKLIB,                                       C
               VERSION=4.0,                                           C
               GOODRC=4,                                              C
               PORDER=1,                                              C
               OPTIONS=(MACRO,OBJECT,SOURCE,XREF,PP(SQL))
*
*********************************************************************
*         --DDNAME ALLOCATION--                                     *
*********************************************************************
*
           FLMALLOC  IOTYPE=O,DDNAME=SYSLIN,KEYREF=OBJ,               C
               RECNUM=5000,DFLTTYP=OBJ
*
           FLMALLOC  IOTYPE=I,DDNAME=SYSLIB,KEYREF=SINC
*
           FLMALLOC  IOTYPE=W,DDNAME=SYSUT1,RECNUM=5000
*
           FLMALLOC  IOTYPE=S,DDNAME=SYSIN,KEYREF=SINC,RECNUM=2000
*
           FLMALLOC  IOTYPE=A,DDNAME=SYSTERM
            FLMCPYLB NULLFILE
*
           FLMALLOC  IOTYPE=A,DDNAME=SYSPUNCH
            FLMCPYLB NULLFILE
*
           FLMALLOC  IOTYPE=O,DDNAME=SYSPRINT,KEYREF=LIST,            C
              DFLTTYP=LIST,PRINT=Y,RECNUM=5000
*
           FLMALLOC  IOTYPE=P,DDNAME=DBRMLIB,MEMBER=@@FLMONM,         C
              DFLTTYP=DBRM,KEYREF=OUT1,                               C
              RECFM=FB,LRECL=80,RECNUM=5000,DIRBLKS=1
*
           FLMALLOC  IOTYPE=A,DDNAME=TASKLIB
            FLMCPYLB  EPLI.V350.SIBMZCMP
            FLMCPYLB  DB2.V810.SDSNEXIT
            FLMCPYLB  DB2.V810.SDSNLOAD
*
           FLMALLOC  IOTYPE=I,DDNAME=DB2DCLS,KEYREF=SINC,INCLS=DB2DCLS
*                                                                     *
* 5694-A01 (C) COPYRIGHT IBM CORP 1980, 2006                          *
*                                                                     *
```

Archived

D

Chapter 13 listings

Example: D-1 JCL: Invoke CICS Preprocessor and COBOL Compiler

```
//USERIDC  JOB (ASO5CR,T12,C531),'USERID',NOTIFY=USERID,CLASS=A,
//    MSGCLASS=O,MSGLEVEL=(1,1)
//*
//*      THIS PROCEDURE CONTAINS 2 STEPS
//*      1.   EXEC THE CICS TS PREPROCESSOR
//*      2.   EXEC THE ENTERPRISE COBOL COMPILER
//*
//*      CHANGE THE JOB NAME AND THE ACCOUNTING INFORMATION TO MEET THE
//*      REQUIREMENTS OF YOUR INSTALLATION.
//*
//*      CHANGE 'PROGNAME' TO THE NAME OF THE CICS/COBOL PROGRAM YOU
//*      WANT TO COMPILE.  CHANGE 'USERID' TO YOUR USERID.
//*
//*      CHANGE 'DEVLEV' TO THE GROUP THAT CONTAINS THE PROGRAM TO BE COMPILED.
//*
//* STEP 1: TRN STATEMENT; STEP 2: EXEC PGM STATEMENT
//*
//TRN    EXEC PGM=DFHECP1$,
//*
//* STEP 3: STEPLIB STATEMENT
//*
//       REGION=2048K
//STEPLIB  DD  DSN=CICS.TS31.CICS.SDFHLOAD,DISP=SHR
//*
//* STEP 4: SYSIN STATEMENT
//*
//SYSIN    DD  DSN=USERID.DEVLEV.SOURCE(PROGNAME),DISP=SHR
//*
//* STEP 5: SYSPRINT STATEMENT
//*
//SYSPRINT DD  SYSOUT=*
//*
```

© Copyright IBM Corp. 2007. All rights reserved.

```
//* STEP 6: SYSPUNCH STATEMENT
//*
//SYSPUNCH DD  DSN=&&SYSCIN,
//             DISP=(,PASS),UNIT=SYSDA,
//             DCB=BLKSIZE=0,
//             SPACE=(CYL,(5,2))
//*
//* STEP 7: COB STATEMENT;  STEP 8:  EXEC PGM STATEMENT
//* STEP 9: PARM STATEMENT; STEP 10: COND STATEMENT
//*
//COB    EXEC PGM=IGYCRTCL,REGION=2048K,COND=(4,GT),
//       PARM='NOTRUNC,NODYNAM,LIB,SIZE=256K,BUF=32K,APOST,DMAP,XREF'
//*
//* STEP 11: STEPLIB STATEMENT
//*
//STEPLIB  DD  DSN=Ecobol.V340.Sigycomp,DISP=SHR
//*
//* STEP 12: SYSLIB STATEMENT; STEP 13: DD STATEMENT CONCATENATION
//*
//SYSLIB   DD  DSN=CICS.TS31.CICS.SDFHCOB,DISP=SHR
//         DD  DSN=CICS.TS31.CICS.SDFHMAC,DISP=SHR
//*
//* STEP 14: SYSPRINT STATEMENT
//*
//SYSPRINT DD  SYSOUT=*
//*
//* STEP 15: SYSIN STATEMENT
//*
//SYSIN    DD  DSN=&&SYSCIN,DISP=(OLD,DELETE)
//*
//* STEP 16: SYSLIN STATEMENT
//*
//SYSLIN   DD  DSN=USERID.DEVLEV.OBJ(PROGNAME),DISP=SHR
//*
//* STEP 17: SYSUT1 STATEMENT
//*
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(5,2))
//*
//* STEP 18: SYSUT2 STATEMENT
//*
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(5,2))
//*
//* STEP 19: SYSUT3 STATEMENT
//*
//SYSUT3   DD  UNIT=SYSDA,SPACE=(CYL,(5,2))
//*
//* STEP 20: SYSUT4 STATEMENT
//*
//SYSUT4   DD  UNIT=SYSDA,SPACE=(CYL,(5,2))
//*
//* STEP 21: SYSUT5 STATEMENT
//*
//SYSUT5   DD  UNIT=SYSDA,SPACE=(CYL,(5,2))
```

Archived

Example: D-2 SCLM Language Definition: Invoke CICS Preprocessor and COBOL

```
*********************************************************************
*          SCLM LANGUAGE DEFINITION FOR
*        ENTERPRISE COBOL WITH CICS TS PREPROCESSOR 3.1
*
* CICS TS OUTPUT IS PASSED VIA THE CICSTRAN DD ALLOCATION TO ENTERPRISE COBOL.
*
* POINT THE FLMSYSLB MACRO(S) AT ALL 'STATIC' COPY DATASETS.
* CUSTOMIZE THE 'OPTIONS' AND 'GOODRC' FIELDS TO YOUR STANDARDS.
* ADD THE 'DSNAME' FIELD IF THE TRANSLATOR IS IN A PRIVATE LIBRARY.
* WHEN A NEW TRANSLATOR VERSION REQUIRES TOTAL RECOMPILATION FOR THIS
* LANGUAGE, THE 'VERSION' FIELD ON FLMLANGL SHOULD BE CHANGED.
*********************************************************************
*
COBCICS  FLMSYSLB  CICS.TS31.CICS.SDFHCOB
*
FLMLANGL  LANG=COBCICS,VERSION=CICSTS31,ALCSYSLB=Y
*
* PARSER TRANSLATOR
*
          FLMTRNSL   CALLNAM='SCLM COBOL PARSE',                    C
               FUNCTN=PARSE,                                        C
               COMPILE=FLMLPCBL,                                    C
               PORDER=1,                                            C
               OPTIONS=(@@FLMLIS,@@FLMSTP,@@FLMSIZ,)
*          (* SOURCE      *)
           FLMALLOC  IOTYPE=A,DDNAME=SOURCE
            FLMCPYLB @@FLMDSN(@@FLMMBR)
*
* BUILD TRANSLATORS
*         - CICS PRECOMPILE - STEP NAME TRN
*
* STEP 1
          FLMTRNSL   CALLNAM='CICS PRE-COMPILE',                    C
               FUNCTN=BUILD,                                        C
* STEP 2
               COMPILE=DFHECP1$,                                    C
* STEP 3  (* STEPLIB *)
             DSNAME=CICS.TS31.CICS.SDFHLOAD,                        C
               VERSION=3.1,                                         C
* STEP 10 (* COND *)
               GOODRC=4,                                            C
               PORDER=1
* STEP 4  (* SYSIN *)
           FLMALLOC  IOTYPE=S,KEYREF=SINC,RECFM=FB,LRECL=80,        C
               DDNAME=SYSIN
* STEP 5   (* SYSPRINT *)
           FLMALLOC  IOTYPE=O,RECFM=FBA,LRECL=121,                  C
               RECNUM=35000,PRINT=Y,DDNAME=SYSPRINT
*
* STEP 6   (* SYSPUNCH *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,                    C
               RECNUM=5000,DDNAME=SYSPUNCH
*
*
```

Archived

```
* STEP 7   (*COBOL INTERFACE - STEP NAME COB *)
* STEP 8
         FLMTRNSL   CALLNAM='COBOL COMPILE',                          C
              FUNCTN=BUILD,                                           C
              COMPILE=IGYCRCTL,                                       C
* STEP 11  (* STEPLIB *)
              DSNAME=ECOBOL.V340.SIGYCOMP,                            C
              VERSION=3.4.0,                                          C
              GOODRC=4,                                               C
* STEP 22
              PORDER=3,                                               C
* STEP 9   (* PARMS *)
              OPTIONS=(NOTRUNC,NODYNAM,LIB,SIZE=256K,BUF=32K,APOST,   C
              DMAP,XREF)* DDNAME ALLOCATIONS
* STEP 16
*  1       (* SYSLIN *)
           FLMALLOC  IOTYPE=O,KEYREF=OBJ,RECFM=FB,LRECL=80,           C
              RECNUM=5000,DFLTTYP=OBJ
* STEP 22
*  2       (* N/A *)
           FLMALLOC  IOTYPE=N
* STEP 22
*  3       (* N/A *)
           FLMALLOC  IOTYPE=N
* STEP 12; STEP 13
*  4       (* SYSLIB *)
           FLMALLOC  IOTYPE=I,KEYREF=SINC
* STEP 15
*  5       (* SYSIN *)
           FLMALLOC  IOTYPE=U,KEYREF=SINC,DDNAME=SYSPUNCH
* STEP 14
*  6       (* SYSPRINT *)
           FLMALLOC  IOTYPE=O,KEYREF=LIST,RECFM=VBA,LRECL=137,        C
              RECNUM=500000,PRINT=Y,DFLTTYP=LIST
* STEP 22
*  7       (* SYSPUNCH *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
* STEP 17
*  8       (* SYSUT1 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* STEP 18
*  9       (* SYSUT2 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* STEP 19
* 10       (* SYSUT3 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* STEP 20
* 11       (* SYSUT4 *)
           FLMALLOC  IOTYPE=W,RECFM=FB,LRECL=80,RECNUM=5000
* STEP 22
* 12       (* SYSTERM *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
```

Archived

```
* STEP 21
* 13       (* SYSUT5 *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
* STEP 22
* 14       (* SYSUT6 *)
           FLMALLOC  IOTYPE=A
            FLMCPYLB NULLFILE
*
* 5694-A01 (C) COPYRIGHT IBM CORP 1980, 2006
*
```

Archived

Archived

E

DDname substitution lists

This appendix provides a list of the DDname substitution lists for the most common compilers and utility programs.

To make the lists readable, they have been split into two tables. Each table lists the relative DD number, shown in the column DD #, then under the column heading for the compiler/utility program, the DD name that is in that position is listed.

Enterprise PL/I is not listed, as it uses a different method to provide DDname substitution. This is covered in 2.2.1, “Using ddnames and ddname substitution lists” on page 41.

HLASM, COBOL, PL/I and DB2 DDname substitutions

Table E-1 contains the DDname substitutions lists for High Level Assembler, Enterprise COBOL and COBOL for OS/390, PL/I for MVS (Not Enterprise PL/I) and for the DB2 precompiler (DSNHPC).

Table E-1 DDname substitutions

DD #	HLASM	Enterprise COBOL	COBOL for OS/390	PL/I for MVS	DB2 Precompile
1	SYSLIN	SYSLIN	SYSLIN	SYSLIN	*Not applicable*
2	*Not applicable*	*Not applicable*	*Not applicable*	*Not applicable*	*Not applicable*
3	*Not applicable*	*Not applicable*	*Not applicable*	*Not applicable*	*Not applicable*
4	SYSLIB	SYSLIB	SYSLIB	SYSLIB	SYSLIB
5	SYSIN	SYSIN	SYSIN	SYSIN	SYSIN
6	SYSPRINT	SYSPRINT	SYSPRINT	SYSPRINT	SYSPRINT
7	SYSPUNCH	SYSPUNCH	SYSPUNCH	SYSPUNCH	*Not applicable*
8	SYSUT1	SYSUT1	SYSUT1	SYSUT1	SYSUT1
9	*Not applicable*	SYSUT2	SYSUT2	*Not applicable*	SYSUT2

© Copyright IBM Corp. 2007. All rights reserved.

DD #	HLASM	Enterprise COBOL	COBOL for OS/390	PL/I for MVS	DB2 Precompile
10	*Not applicable*	SYSUT3	SYSUT3	*Not applicable*	SYSUT3
11	*Not applicable*	SYSUT4	SYSUT4	*Not applicable*	*Not applicable*
12	SYSTERM	SYSTERM	SYSTERM	*Not applicable*	SYSTERM
13	*Not applicable*	SYSUT5	SYSUT5	*Not applicable*	*Not applicable*
14	*Not applicable*	SYSUT6	SYSUT6	SYSCIN	SYSCIN
15	*Not applicable*	SYSUT7	SYSUT7		*Not applicable*
16	SYSADATA	SYSADATA	SYSADATA		DBRMLIB
17	*Not applicable*	SYSJAVA	SYSIDL		
18	*Not applicable*	SYSDEBUG	SYSDEBUG		
19	*Not applicable*	SYSMDECK			
20	ASMAOPT				

Archived

CICS, Binder, DFSMS utilities, C/C++ and PL/X DDname substitutions

Table E-2 contains the DDname substitutions lists for the CICS precompiler, the Linkage editor/Binder, DFSMS™ utilities such as IEBGENER, C/C++ compiler and the PL/X-390 compiler.

Table E-2 DDname substitutions

DD #	CICS	Binder	DFSMS Utilities	C/C++	PL/x
1	*Not applicable*	SYSLIN	*Not applicable*	SYSIN	*Not applicable*
2	*Not applicable*	member name	*Not applicable*	SYSLIN	*Not applicable*
3	*Not applicable*	SYSLMOD	*Not applicable*	SYSMSGS*	*Not applicable*
4	*Not applicable*	SYSLIB	*Not applicable*	SYSLIB	SYSLIB
5	SYSIN	*Not applicable*	SYSIN	USERLIB	SYSIN
6	SYSPRINT	SYSPRINT or SYSLOUT	SYSPRINT	SYSPRINT	SYSPRINT
7	SYSPUNCH	*Not applicable*	*Not applicable*	SYSCPRT	SYSPUNCH
8		*Not applicable*	SYSUT1	SYSPUNCH	*Not applicable*
9		*Not applicable*	SYSUT2	SYSUT1	SYSUT2
10		*Not applicable*	SYSUT3	SYSUT4	*Not applicable*
11		*Not applicable*	SYSUT4	SYSUT5	*Not applicable*
12		SYSTERM		SYSUT6	SYSTERM
13		SYSDEFSD		SYSUT7	*Not applicable*
14				SYSUT8	*Not applicable*
15				SYSUT9	*Not applicable*
16				SYSUT10	SYSLOGIC
17				SYSUT14	*Reserved*
18				SYSUT15	*Reserved*
19				SYSUT16	*Reserved*
20				SYSUT17	ASMLIN
21				SYSEVENT	ASMLIB
22				TEMPINC	ASMPRINT
23					ASMPUNCH
24					ASMUT1
25					ASMTERM
26					SYSADATA

* SYSMSGS ddname is no longer used, but is kept in the list for compatibility with old assembler macros.

Archived

F

Additional material

This Redbooks publication refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this IBM Redbooks publication is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

ftp://www.redbooks.ibm.com/redbooks/SG247392

Alternatively, you can go to the IBM Redbooks publications Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks publications form number, SG247392.

Using the Web material

The additional Web material that accompanies this IBM Redbooks publication includes the following files:

File name	*Description*
SG247392.zip	Zipped Project definition and code samples

There is a README.txt file that is contained in the zip file that contains additional information.

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	4 MB for the downloaded zip file and unpacked files
Operating System:	Windows 2000/XP
Processor:	Pentium®
Memory:	128 MB

© Copyright IBM Corp. 2007. All rights reserved.

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

The extracted files are all in binary format. They are the output of the TSO TRANSMIT command.

Use your mainframe file transfer protocol to upload the binary files. You must use the following attributes: FB, LRECL=80, BLKSIZE=3120.

After each file is uploaded, issue the following command from the TSO READY prompt or ISPF Command shell (option 6):

```
RECEIVE INDA(xxxx)
```

In this command, *xxxx* is the name of the file.

Alternatively, list all the files in ISPF option 3.4 and next to each one, issue the command:

```
RECEIVE INDA(/)
```

You will receive the following messages, if you issue the command against the source file as shown in Example F-1.

Example: F-1 Receive INDA(xxxx) messages

```
INMR901I Dataset SCLM.PROJDEFS.REXX from DOHERTL on NODENAME
INMR906A Enter restore parameters or 'DELETE' or 'END' +
```

You can then reply as shown in Example F-2.

Example: F-2 Receive INDA(xxxx) reply

```
da(redbooks.sclm.projdefs.rexx)
```

Archived

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this IBM Redbooks publication.

IBM Redbooks publications

For information about ordering these publications, see “How to get IBM Redbooks publications” on page 753. Note that some of the documents referenced here may be available in softcopy only.

- *Supporting On Demand Business Applications with the IBM Problem Determination Tools (APA, DT, DT with Advanced Facilities, FA, File Export, FM,WS),* SG24-7192
- *Rational Application Developer V6 Programming Guide*, SG24-6449
- *Experience J2EE! Using WebSphere Application Server V6.1*, SG24-7297

Other publications

These publications are also relevant as further information sources:

- *Software Configuration and Library Manager (SCLM) Guide and Reference,* SC34-4817
- *SCLM Developer Toolkit Installation and Customization Guide,* SC31-6970
- *IBM Merge Tool for z/OS and z/OS User’s Guide,* SC27-1694
- *Enhanced Access Control for SCLM for z/OS User’s Guide,* SC27-1591
- *Breeze Installation Guide*, SC31-8819-08

Online resources

These Web sites are also relevant as further information sources:

- UNIX System Services:

  `http://www-03.ibm.com/servers/eserver/zseries/zos/unix/release/nmas.html#Header_2`
- IBM packaging rule in relation to HFS directories:

  `http://submit.boulder.ibm.com/w3sdf/mvsrules/`

How to get IBM Redbooks publications

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

© Copyright IBM Corp. 2007. All rights reserved.

Help from IBM

IBM Support and downloads:

ibm.com/support

IBM Global Services:

ibm.com/services

Archived

Index

© Copyright IBM Corp. 2007. All rights reserved.

C

D

E

F

G

H

I

J

K

L

Archived

M

N

O

P

R

S

T

U

V

W

Archived

Z

Archived

IBM

Redbooks

Getting Started with SCLM: A Practical Guide to SCLM and SCLM Advanced Edition

(1.5” spine)
1.5”<-> 1.998”
789 <->1051 pages

Archived

Archived

Archived

Getting Started with SCLM: A Practical Guide to SCLM and SCLM Advanced Edition

Understand the basics of SCLM project setup, use, and administration

Extend SCLM functionality with SCLM Advanced Edition

Develop from your desktop with SCLM Developer Toolkit

This IBM Redbooks publication describes and documents a number of different aspects of the Software Configuration and Library Manager (SCLM).

Part 1 of the book focuses on setting up an SCLM project using commonly used languages such as COBOL and PL/I. Additionally, migration techniques are discussed for those customers considering migrating to SCLM from other vendor Software Configuration Management products.

Part 2 describes basic usage of SCLM functions such as Edit, Build, and Promote.

Part 3 goes a bit beyond the basics and looks at some of the newer functions that are being added to SCLM, along with writing user exits and setting up SCLM for debugging with IBM Debug Tool.

Parts 4, 5, and 6 concentrate on the SCLM Advanced Edition products such as Breeze, Enhanced Access Control, SCLM Developer Toolkit, and SCLM Administrator Toolkit. These sections describe what these products are and how they can be set up and used to aid your application development.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7392-00 ISBN 0738489468