

SCLM Developer Toolkit



Installation and Customization Guide

Version 3.1

SCLM Developer Toolkit



Installation and Customization Guide

Version 3.1

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page 103.

| **Second Edition (December 2007)**

| This edition applies to IBM SCLM Developer Toolkit, Version 3 Release 1, Program Number 5655-S72 and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

| The IBM SCLM Developer Toolkit Web site is at
| <http://www.ibm.com/software/awdtools/sclmsuite/devtoolkit/>

| The latest edition of this document is always available from the Web site.

© Copyright International Business Machines Corporation 2005, 2007. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	v
--------------------------	----------

Tables	vii
-------------------------	------------

About this document	ix
--------------------------------------	-----------

Who should use this document	ix
Changes from the previous edition.	ix
Where to find more information	x
Publications	x
Softcopy publications	x
IBM Systems Center publications	xi

Installation overview	xiii
--	-------------

TCP/IP considerations	xiii
SMP/E installation	xiv
Batch job considerations	xiv
Separate SCLM installation	xiv

Part 1. Installing SCLM Developer Toolkit	1
--	----------

Chapter 1. Installing and customizing SCLM Developer Toolkit on z/OS	3
---	----------

Step 1: Check z/OS software requirements	4
Step 2: Configuration considerations	4
Step 3: Run the setup JCL	6
Step 4: Customize the SCLM Developer Toolkit configuration files	8
Customize the ISPF configuration file	8
Customize the TRANSLATE configuration file	9
Override settings in the TRANSLATE.conf file.	13
Step 5a: Configure the SCLM Developer Toolkit HTTP Server	13
HTTP server configuration file customization	14
HTTP server environment file customization	15
HTTP server JCL/STARTED TASK customization	15
Step 5b: Configure Remote Systems Explorer	18
Customizing the RSE Environment file	18
Customizing the RSE environment setup script	20
Activating the RSE Environment file	20
Step 6: Configure long/short name table VSAM file	21
Step 7: Install and customize Ant	23
Step 8a: Run the IVP to check correct HTTP installation and customization	25
Testing connection to the HTTP server	29
Step 8b: Run the IVP to check correct RSE installation and customization	30

Chapter 2. Installing the Eclipse-based client onto the PC	33
---	-----------

Preparing for installation	33
Media requirements	33
Hardware and software requirements.	33

Installing SCLM Developer Toolkit	34
Step 1. Install from CD or electronic image	34
Step 2. Install SCLM Developer Toolkit	35

Part 2. Customizing SCLM Developer Toolkit	37
---	-----------

Chapter 3. SCLM customization for the SCLM administrator	39
---	-----------

Language translators for JAVA/J2EE support	39
JAVA/J2EE build summary	40
JAVA/J2EE build objects generated	41
SCLM language definitions	42
SCLM types	43
SCLM member formats	45
JAVA/J2EE Ant XML build skeletons	53
Mapping J2EE projects to SCLM	54
Recommendations for mapping J2EE projects to SCLM	57
SCLM Developer Toolkit deployment.	58
WebSphere Application Server (WAS) deployment	59
SCLM to Unix System Services deployment	59
Secure deployment	60
Other deployment options	60
ASCII or EBCDIC storage options	61
ASCII/EBCDIC language translators	61
\$GLOBAL member	62
SITE and project-specific options	63
Example of using combinations of the TRANSLATE.conf overrides	69
Example of using combinations of the BIDIPROP overrides	71

Chapter 4. SCLM security.	73
--	-----------

Build/Promote/Deploy security flag and process flow	73
Security rules and surrogate user ID	73
Build rule format	74
Promote rule format	74
Deploy rule format	74
SAF/RACF BUILD, PROMOTE, DEPLOY, and PROFILE rules	75

Chapter 5. CRON-initiated Builds and Promotes	77
--	-----------

STEPLIB and PATH requirements	77
CRON Build job execution	78
CRON Build job samples	78

Appendix A. SCLM overview	81
--	-----------

SCLM Concepts	81
File naming	81

Type.	81
Language	82
SCLM properties	82
SCLM project structure	82
ARCHDEF	83
JAVA/J2EE concepts	83

Appendix B. SQLJ Support 85

What is SQL?.	85
What is DB2?.	85
What is JDBC?	85
What is SQLJ?	85
Comparing JDBC and SQLJ	86
What is a Serialized Profile?.	87
What is a DBRM?	87
SQLJ Program Preparation	88
Translation	88
Example:	89
Customization	89
Binding.	89
SCLM DT types and translators	90
Tailoring the build process	90
Tailoring the Build Script.	91

Appendix C. Long/short name translation table. 97

Technical summary of the SCLM Translate program	97
Single long/short name record processing	98
FINDLONG Processing	98
FINDSHORT Processing	98
TRANSLATE Processing	99
Multiple long/short name record processing	99
IMPORT processing	99
MIGRATE processing.	100

Bibliography. 101

Notices 103

Trademarks	104
----------------------	-----

Glossary 107

Index 111

Figures

1. Sample ISPF.conf	9	19. Sample EJB Application (EJB) ARCHDEF	48
2. User of TRANSLATE.conf keywords	12	20. Sample EAR Application (EAR) ARCHDEF	49
3. Pass and exec directives	16	21. J2EE Ant build script	49
4. Sample rsed.envvars	20	22. J2EE Build script JAR sample	52
5. Sample Long/Short Translate VSAM file JCL	21	23. J2EE Build script WAR sample	52
6. HTTP server logon prompt	26	24. J2EE Build script EJB sample.	53
7. Host installation and customization welcome screen	26	25. J2EE Build script EAR sample	53
8. An example of validation responses (part 1)	27	26. Multiple types	56
9. An example of validation responses (part 2)	28	27. SCLM build hierarchy	57
10. An example of validation responses (part 3)	28	28. SCLM Language Translators and ASCII/EBCDIC	61
11. Server Connection successful message.	29	29. Sample SITE specific SCLM project setting	66
12. Change directory command	30	30. Sample PROJECT specific SCLM project setting	67
13. IVP validation responses	31	31. Sample CRON members	77
14. Sample translators	39	32. Sample CRON Build Exec.	79
15. Sample Jar application (JAR) ARCHDEF	41	33. Sample Build parameter file	80
16. J2EE Build script JAR sample	41	34. Multiple types	82
17. Sample Jar application (JAR) ARCHDEF	47	35. Sample REXX for Translate module invocation	100
18. Sample Web application (WAR) ARCHDEF	47		

Tables

1. Publications	x	7. Required operating systems	34
2. IBM Systems Center Publications	xii	8. Customer-defined variables	50
3. httpd.conf customization	14	9. \$GLOBAL variables	62
4. httpd.env customization	15	10. SITE/Project options	67
5. RSE Environment file customization	18	11. Long/Short name translation parameters	98
6. Required hardware	34		

About this document

This document contains the configuration procedure for the IBM® SCLM Developer Toolkit product, which combines standard z/OS® installation procedures with z/OS UNIX® System Services and IBM z/OS HTTP server configuration. It also describes the Eclipse client installation on Windows®.

Hereafter, the following names will be used in this manual:

- The IBM z/OS HTTP server is called the “HTTP server.”
- Rational® Developer for System z is called “Rational Developer.”

Sometimes, “IBM SCLM Developer Toolkit” is shortened to “Developer Toolkit” or “SCLMDT.”

Who should use this document

Part 1 of this document is written for system programmers who are installing, configuring and administering the IBM SCLM Developer Toolkit. Readers should be familiar with the z/OS UNIX System Services environment (z/OS UNIX System Services file system), structure, security software (for example, Resource Access Control Facility [RACF®]) profiles needed to support z/OS UNIX System Services, started tasks (or the equivalent for the installed security product), and, if used, the HTTP server.

Additionally, a chapter is provided in Part 1 if you are installing the client component on Windows.

Part 2 of this document contains information for the administrator of any SCLM projects that will be used with the SCLM Developer Toolkit. This includes projects that use Java® and z/OS UNIX System Services component languages, as well as traditional SCLM projects. These administrators should also be familiar with the z/OS UNIX System Services environment and z/OS UNIX System Services file system structures, REXX Script, and the Java Compiler and SCLM project and language definitions.

Changes from the previous edition

Third edition

- SQLJ support has been added to allow SQLJ source code to be stored in SCLM and built. This provides the ability to include .sqlj source files within existing J2EE archdefs (mixed with existing Java source if required) when building and creating project JAR files.
- Additional changes to clarify the steps required for installation.
- Messages appendix has been removed.
- Other minor changes to the documentation.

The changes are marked with revision bars.

Where to find more information

Where necessary, this document references information in other books, using shortened versions of the book title. For complete titles and order numbers of the books for all products that are part of z/OS, see the *z/OS Information Roadmap*. Direct your request for copies of any IBM publication to your IBM representative or to the IBM branch office serving your locality.

There is also a (U.S.) toll-free customer support number (1-800-879-2755) available Monday through Friday from 6:30 a.m. through 5:00 p.m. mountain time. You can use this number to:

- Order or inquire about IBM publications
- Resolve any software manufacturing or delivery concerns
- Activate the program reorder form for faster and more convenient ordering of software updates.

Publications

Table 1. Publications

Short Title Used in This Document	Title of Publication	Order Number
HTTP Server Guide	<i>HTTP Server Planning, Installing and Using</i>	SC31-8690
Rational Developer for System z Host Configuration Guide	<i>Rational Developer for System z Host Configuration Guide</i>	SC31-6930
Communications Server for z/OS V1R2 TCP/IP Implementation Guide	<i>Communications Server for z/OS V1R2 TCP/IP Implementation Guide</i>	SG24-5227
z/OS UNIX System Services Planning	<i>z/OS UNIX System Services Planning</i>	GA22-7800
z/OS UNIX System Services Messages	<i>z/OS UNIX System Services Messages and Codes</i>	GA22-7807
z/OS UNIX System Services Commands	<i>z/OS UNIX System Services Command Reference</i>	GA22-7802
IBM Ported Tools for z/OS User's Guide	<i>IBM Ported Tools for z/OS User's Guide</i>	SA22-7985
SCLM Project Manager's Guide	<i>z/OS ISPF Software Configuration and Library Manager Project Manager's and Developer's Guide</i>	SC34-4817
SCLM Developer Toolkit: Program Directory (GI10-3352-00)	<i>SCLM Developer Toolkit: Program Directory</i>	GI10-3352
SCLM Reference	<i>z/OS ISPF Software Configuration and Library Manager Reference</i>	SC34-4818

Softcopy publications

The z/OS library is available on the z/OS Collection Kit, SK2T-6700. This softcopy collection contains a set of z/OS and related unlicensed product books. The CD-ROM collection includes the IBM Library Reader™, a program that you can use to read the softcopy books.

Softcopy z/OS publications are also available for Web browsing. PDF versions of the z/OS publications for viewing or printing using Adobe Acrobat Reader are available at this URL:

- <http://www.ibm.com/servers/eserver/zseries/zos>

Select "Library".

You can also find Rational Developer for System z, Eclipse, and SCLM Advanced Edition information at the following URLs:

- <http://www.ibm.com/software/awdtools/devzseries> – Rational Developer for System z Web site
- <http://www.eclipse.org> – Eclipse home page
- <http://www.ibm.com/software/awdtools/sclmsuite>

IBM Systems Center publications

IBM Systems Center produces Redbooks™ that can be helpful in setting up and using z/OS UNIX System Services. You can order these publications through the usual channels, or you can view them with a Web browser from this URL:

<http://www.redbooks.ibm.com>

These books have not been subjected to any formal review, nor have they been checked for technical accuracy, but they represent current product understanding (at the time of their publication) and provide valuable information about a wide range of z/OS topics. You must order them separately.

Where to find more information

A selected list of these books follows:

Table 2. IBM Systems Center Publications

Title of Publication	Order Number	Comments
<i>P/390, R/390, S/390 Integrated Server: OS/390 New User's Cookbook</i>	SG24-4757-01	Despite the title, it is oriented toward the system programmer, and describes considerations for the z/OS UNIX System Services environment.
<i>Debugging UNIX System Services, Lotus Domino, Novell Network Services</i>	SG24-5613-00	Provides an overview of the z/OS UNIX System Services environment along with tips and suggestions for setup and problem analysis.
<i>OS/390 e-business Infrastructure: IBM HTTP Server V5.1 for OS/390</i>	SG24-5603-00	Provides an overview of Web servers in general with specific details for the OS/390® server along with hints and tips for setup and customization.
<i>ABCs of z/OS System Programming Vol 9</i>	SG24-6989-00	Describes UNIX System Services for system programmers.
e-business Enablement Cookbook for OS/390 Volumes 1, 2, and 3	SG24-5664-00 SG24-5981-00 SG24-5980-00	

Installation overview

This manual contains the installation procedure for all components of SCLM Developer Toolkit. There are two installation components to SCLM Developer Toolkit; the z/OS host component and the client component. The client component is an Eclipse plug-in that can be installed into an existing Eclipse, installed with the Eclipse that is shipped with the standalone SCLM Developer Toolkit installation, or if Rational Developer for System z has been installed, the plug-in is an installable part of that installation. Eclipse is an open extensible Integrated Development Environment (IDE), which will be referred through this document as the Eclipse IDE.

The installation procedure is a combination of standard z/OS installation procedures, z/OS UNIX System Services file set up, HTTP server configuration, and, if installed as a plug-in with Rational Developer for System z, RSE configuration.

The manual is structured into the following chapters:

- Chapter 1, "Installing and customizing SCLM Developer Toolkit on z/OS," on page 3
- Chapter 3, "SCLM customization for the SCLM administrator," on page 39
- Chapter 4, "SCLM security," on page 73
- Chapter 5, "CRON-initiated Builds and Promotes," on page 77

The appendixes, starting on page 81

For a high-level description of SCLM and JAVA/J2EE concepts, see Appendix A, "SCLM overview," on page 81.

TCP/IP considerations

When you are setting up the SCLM Developer Toolkit server, you also need to consider your site's installation of TCP/IP. SCLM Developer Toolkit can use an HTTP server or, if it is installed as a plug-in in Rational Developer for System z, an RSE connection.

If the SCLM Developer Toolkit is configured to use an HTTP server, then the TCPIP.DATA file must be available. The z/OS UNIX System Services Planning guide documents where the system finds this file.

However, if you use another method of defining the location of this file (such as the System Resolver), you must add a //SYSTCPD DD card to your SCLM Developer Toolkit server job. A TCP port must be available and it is recommended to reserve the port number. To understand more about the System Resolver and TCPIP.DATA, see the following publications:

- z/OS UNIX System Services Planning
- Communications Server for z/OS V1R2 TCP/IP Implementation Guide

SMP/E installation

This manual does not cover the implementation aspects of the SCLM Developer Toolkit. Rather, it is intended to guide you through a successful configuration of the product. The manual assumes that the System Modification Program/Extended (SMP/E) installation of the SCLM Developer Toolkit is complete. The SMP/E instructions for the SCLM Developer Toolkit are in the IBM SCLM Developer Toolkit Program Directory. Before you begin the SCLM Developer Toolkit installation, note that the following actions are recommended for the SMP/E installation:

- The root z/OS UNIX System Services file system is read-only
- The SCLM Developer Toolkit directory is set up as a separate file system, mounted onto the root file system at /usr/lpp/SCLMDT (or whatever name you choose for your SCLM Developer Toolkit root directory).

These recommendations conform with those specified in the z/OS UNIX System Services Planning guide. See the section called “Deciding How to Mount Your Root z/OS UNIX System Services file system for Execution” for details.

Upon successful completion of the SMP/E installation, follow the directions in the subsequent chapters to complete the installation and customization on z/OS, and to install the Eclipse-based client on the PC.

Batch job considerations

The SCLM Developer Toolkit uses SDSF to retrieve job completion status and job output. Because not all customers have JES2 or SDSF, additional support has been added to the SCLM Developer Toolkit to use the OUTPUT command. Shipped with z/OS, the OUTPUT command only lets you retrieve job output that begins with the logged on user ID. If you want to use the OUTPUT facility fully, then the supplied TSO/E exit IKJEFF53 might need to be modified so that you can retrieve job output you own, but that does not begin with your user ID. For more information about this exit, see the z/OS TSO/E Customization Guide.

Separate SCLM installation

This manual does not cover the implementation and loading of the SCLM product.

Part 1. Installing SCLM Developer Toolkit

Chapter 1. Installing and customizing SCLM

Developer Toolkit on z/OS 3

Step 1: Check z/OS software requirements 4

Step 2: Configuration considerations 4

Step 3: Run the setup JCL 6

Step 4: Customize the SCLM Developer Toolkit

configuration files 8

 Customize the ISPF configuration file 8

 Customize the TRANSLATE configuration file . . . 9

 Example of the TRANSLATE configuration file 13

 Override settings in the TRANSLATE.conf file. . 13

Step 5a: Configure the SCLM Developer Toolkit

HTTP Server 13

 HTTP server configuration file customization . . 14

 HTTP server environment file customization . . 15

 HTTP server JCL/STARTED TASK customization 15

 Customizing an existing HTTP server for

 SCLM support 16

 Start the SCLM Developer Toolkit HTTP

 server 16

 Enabling trace on the HTTP server 16

Step 5b: Configure Remote Systems Explorer . . . 18

 Customizing the RSE Environment file 18

 Customizing the RSE environment setup script 20

 Activating the RSE Environment file 20

Step 6: Configure long/short name table VSAM file 21

Step 7: Install and customize Ant 23

Step 8a: Run the IVP to check correct HTTP

installation and customization 25

 Testing connection to the HTTP server 29

Step 8b: Run the IVP to check correct RSE

installation and customization 30

Chapter 2. Installing the Eclipse-based client

onto the PC 33

Preparing for installation 33

 Media requirements 33

 Hardware and software requirements. 33

 Prerequisites for SCLM Developer Toolkit . . 33

Installing SCLM Developer Toolkit 34

 Step 1. Install from CD or electronic image . . . 34

 Step 2. Install SCLM Developer Toolkit 35

Chapter 1. Installing and customizing SCLM Developer Toolkit on z/OS

This chapter provides a list of the tasks required to install SCLM Developer Toolkit on your host z/OS system and the tasks involved in installing the Eclipse client plug-in of the SCLM Developer Toolkit.

To install the SCLM Developer Toolkit on your host z/OS system, use the following steps:

- “Step 1: Check z/OS software requirements” on page 4
- “Step 2: Configuration considerations” on page 4
- “Step 3: Run the setup JCL” on page 6
- “Step 4: Customize the SCLM Developer Toolkit configuration files” on page 8
- “Step 5a: Configure the SCLM Developer Toolkit HTTP Server” on page 13, or
- “Step 5b: Configure Remote Systems Explorer” on page 18
- “Step 6: Configure long/short name table VSAM file” on page 21
- “Step 7: Install and customize Ant” on page 23
- “Step 8a: Run the IVP to check correct HTTP installation and customization” on page 25, or
- “Step 8b: Run the IVP to check correct RSE installation and customization” on page 30

These steps must be implemented by the z/OS Systems Programmer.

The SCLM Developer Toolkit can connect to the z/OS host either by using an HTTP server or by using an RSE (Remote Systems Explorer) connection. If the SCLM Developer Toolkit is running under Rational Developer for System z, then the communication mechanism is RSE. Configuring SCLM Developer Toolkit for RSE communication is covered in “Step 5b: Configure Remote Systems Explorer” on page 18. If SCLM Developer Toolkit is running under any other installation of Eclipse, such as Rational® Application Developer (RAD), then the communication mechanism is HTTP. This is described in “Step 5a: Configure the SCLM Developer Toolkit HTTP Server” on page 13.

For additional information about configuring SCLM Developer Toolkit for specific SCLM projects, see Chapter 3, “SCLM customization for the SCLM administrator,” on page 39. This chapter contains additional customization for:

- Site or SCLM project-specific settings
- JAVA/J2EE language support

Step 1: Check z/OS software requirements

To successfully install the SCLM Developer Toolkit, the following system requirements must be in place when you begin your installation:

- z/OS V1.7 or above with the following PTFs applied:
 - ISPF PTF that addresses APAR OA20345 to enable correct log output messaging and to provide additional BUILD service processing information.
 - ISPF/SCLM PTF that addresses APAR OA21104 with SCLMINFO enhancements and build mode information for syntax checking.
 - ISPF PTF that addresses APAR OA16924, which enhances the SCLMINFO service.
 - If you want to store files with long filenames in SCLM, the PTF that addresses ISPF APAR OA11426 and provides support for long/short filename (long/short-name) translation. This is not required on z/OS V1.8.
 - If you want to use secure build, promote, and deploy (see Chapter 4, “SCLM security,” on page 73): the ISPF/SCLM PTF that addresses APAR OA16804.
- z/OS UNIX System Services (and TCP/IP)
- REXX runtime or REXX alternate libraries (REXX.**.SEAGLPA, or REXX.**.SEAGALT). If you do not have either the REXX Library or the REXX Alternate Library installed, you can install the REXX Alternate Library to fulfill the REXX Library requirement. The REXX Alternate Library is available as a free download from: <http://www-1.ibm.com/support/>.
The REXX runtime library REXX.**.SEAGLPA (or if not installed, the alternate library REXX.**.SEAGALT) must be added to the STEPLIB in the HTTP server's JCL if they are not already defined as LINKLIST data sets. Your site might use a different high-level qualifier to REXX.
These data sets must be APF authorized.
- Ant runtime installed in UNIX System Services, if performing JAVA/J2EE builds. (Download available from <http://Ant.apache.org/>).

See “Step 7: Install and customize Ant” on page 23.

- One of the following versions of z/OS Java:
 - Java V1.4.0 (5655-I56) or later.
 - Java V1.4.2 (5655-M30) or later.
 - Java V5 – 31bit (5655-N98) or later.
 - Java V5 – 64bit (5655-N99) or later.

Setting MEMLIMIT

z/OS uses region sizes to determine the amount of storage available to running programs. For the 64-bit product, allow 256 MB or greater to a running SDK. Use the MEMLIMIT parameter for this setting. For information about the MEMLIMIT parameter, see Limiting Storage use above the bar in z/Architecture.

Step 2: Configuration considerations

Consider the following points before you configure your system.

1. Each user of the SCLM Developer Toolkit must define a RACF OMVS segment (or equivalent) that specifies a valid non-zero uid, home directory, and shell command.

If this is not specified, you can not log on to the SCLM Developer Toolkit through the Eclipse IDE.

2. Set MAXPROCUSER in BPXPRMxx parmlib member to a minimum of 50. This can be checked and set dynamically (until the next IPL) with the following commands (as described in z/OS MVS System Commands SA22-7627):

```
DISPLAY OMVS,0  
SETOMVS MAXPROCUSER=50.
```

Setting a value that is too low can cause SCLM translations and possibly other activities to fail.

Configuration considerations

3. It is recommended that you have the BWB* modules saved in a data set that is part of the LINKLIST. Alternatively, this data set can be added:
 - to the STEPLIB in the HTTP server (see “Step 5a: Configure the SCLM Developer Toolkit HTTP Server” on page 13), or
 - in the STEPLIB statement in the rsd.envvars of the RSE server (see “Step 5b: Configure Remote Systems Explorer” on page 18).
4. The 64 bit versions of Java can use the storage available above the 2GB bar. However, this storage is unavailable by default. Set MEMLIMIT in the SMFPRMxx parmlib member to NOLIMIT, or any valid value greater than the default, 0M. Refer to the *z/OS MVS Initialization and Tuning Reference* (SA22-7592) for more information.

Step 3: Run the setup JCL

1. Customize and run BWBINST1, which is stored in the SBWBSAMP data set. Follow the customization instructions within the member.

This job performs the following tasks:

 - Creates CONFIG, LOGS, and WORKAREA directories in the z/OS UNIX System Services file system at the directory you specify.
 - Creates the PROJECT directory in the z/OS UNIX System Services file system under the CONFIG directory. For more information about the use of the PROJECT directory, see “SITE and project-specific options” on page 63.
 - Copies the sample HTTP server configuration files from members BWBHTTPC and BWBHTTPE in the SBWBSAMP library to files httpd.conf and httpd.env in the CONFIG directory. These two files require customization. See “Step 5a: Configure the SCLM Developer Toolkit HTTP Server” on page 13.
 - Copies the Sample ISPF configuration table from member BWBISPF in the SBWBSAMP library to file ISPF.conf in the CONFIG directory. This file requires customization. See “Step 4: Customize the SCLM Developer Toolkit configuration files” on page 8.
 - Copies the sample Translation configuration table from member BWBTRANC in the SBWBSAMP library to file TRANSLATE.conf in the CONFIG directory. This file requires customization. See “Step 4: Customize the SCLM Developer Toolkit configuration files” on page 8.

The recommended base directory for the configuration files is /etc/SCLMDT. The part of the directory up to SCLMDT must exist before running this job.

You must have read and write access to the WORKAREA and LOGS directories. By default you must have read and write access to /var/SCLMDT/WORKAREA and /var/SCLMDT/LOGS. The WORKAREA is used for transfer of files, ASCII/EBCDIC conversions, and for JAVA/J2EE builds.

Temporary directories of format /var/SCLMDT/WORKAREA/userid/* are created during use of the Developer Toolkit. The following directories can be created under a directory of your user ID in the WORKAREA directory, depending on the type of functions you are performing:

- /EDIT
- /JobOutput
- /TRANSFER
- /VERSION

Note:

| Some temporary session files might be created in the /tmp directory. Ensure all
| users have write access to the /tmp directory.

SCLM Developer Toolkit removes any temporary files it creates in the WORKAREA directory. However, temporary output is sometimes left over, for example, if there is a communication error while processing. For this reason, it is recommended that you clear out the WORKAREA and LOGS directories from time to time.

Run the setup JCL

To do this, use the following commands in OMVS:

```
cd /var/SCLMDT/WORKAREA
rm -r *
```

Where */var/SCLMDT/WORKAREA* is saved depends on where you create the WORKAREA directory.

This removes all entries and the same process can be used on the LOGS directory.

Step 4: Customize the SCLM Developer Toolkit configuration files

The files ISPF.conf and TRANSLATE.conf are saved at the default directory location */etc/SCLMDT/CONFIG* and might require further customization.

Customize the ISPF configuration file

For the SCLM Developer Toolkit to run ISPF and SCLM services, a valid ISPF environment must be established. The ISPF configuration contains the required allocations for SCLM Developer Toolkit to establish a TSO/ISPF environment session for the user.

You must customize the ISPF configuration file *ISPF.conf* that is stored in the CONFIG directory to host site requirements for ISPF data set allocation. The provided sample *ISPF.conf* has instructions to complete customization so your user site can:

- Include the minimum ISPF data set allocations for SCLM Developer Toolkit operation. This means allocating the minimum ISPF data sets that bring ISPF up in a 3270 emulator. In the sample provided, these are specified as the *isp.sisp** data sets. You might need to change these for your site-specified data set names.
- Add additional DDNAME file allocations or concatenate additional ISPF data sets.
- Launch a customer defined allocation executable (*exec*) to provide further data set allocation by project or user ID. A sample *exec* is provided in member *BWBISPF2* in the *SBWBSAMP* library.
- SCLM Developer Toolkit uses the standard allocated ISPF/SCLM skeletons, for example, *FLMLIBS*, so ensure that required skeleton libraries are allocated to the *ISPSLIB* DD in *ISPF.conf*.

The allocations for each of the ISPF DDs must be specified on a single line with each data set separated by a comma. Comment lines can be added by beginning the line with an asterisk (*). See the sample *ISPF.conf* below for an example.

```

* REQUIRED:
* Below is the minimum requirements for ISPF allocation.
* Change the default ISPF data set names below to match your host site.
* Add additional dsn concatenations on same line and separate by comma.
* Order of data sets listed is search order in concatenation.
* The sclmdt loadlib data set is required to be added to the ISPLLIB
* concatenation to access the JAVA/J2EE SCLM Language translators.
* Change BWB.SBWBLOAD to the appropriate data set where the
* BWBxxx load modules are stored.
*
* 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=ISP.SISPCLIB,BZZ.SBZZCLIB
ispmllib=ISP.SISPMENU
isptlib=ISP.SISPTENU
ispplib=ISP.SISPPENU
ispslib=BZZ.SBZZSENU,ISP.SISPSLIB
ispllib=BWB.SBWBLOAD,BZZ.SBZZLOAD

```

Figure 1. Sample ISPF.conf

Customize the TRANSLATE configuration file

Review the TRANSLATE configuration file TRANSLATE.conf that is stored in the CONFIG directory. Follow the instructions contained within the sample if different ASCII/EBCDIC conversion codepages are required, other than the default of ASCII=ISO8859-1 and EBCDIC=IBM-1047.

The TRANSLATE.conf file provides keywords to determine how code is stored within SCLM. The configuration file contains keywords that determine how files are transferred to the host depending on their language definition. Specific keywords determine if files of a certain language type are binary, transferred, and stored or whether the text based source remains in ASCII format rather than the default translation from ASCII to EBCDIC.

Additionally SCLM language definitions control 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.

Note: Default language definitions have been provided as a guide for determining long-to-short name conversions, and/or BINARY transferred language definitions.

The following keywords are valid within the TRANSLATE.conf file:

Keyword	Description
CODEPAGE	Determines the ASCII and EBCDIC codepages to use in translation Format: <ul style="list-style-type: none"> CODEPAGE ASCII = ISO8859-1 CODEPAGE EBCDIC = IBM-1047 There must be a CODEPAGE keyword for both ASCII and EBCDIC for SCLM Developer Toolkit to determine how to convert files being transferred.
TRANVRLS	Indicates whether SCLM should allow the Translate table VSAM

Customize the SCLMDT configuration files

data set to be shared across systems when the level of DFSMS installed is 1.3 or later. The default is NO.

SCLM uses VSAM Record Level Sharing (RLS) to allow the sharing of the VSAM data sets. To maintain the integrity of the VSAM data sets in a shared environment, the VSAM data sets must be allocated for RLS and all hardware and software to support RLS must be in place for the system. The Translate table must be defined with the correct STORAGECLASS for RLS use. (See the DFSMS documentation for hardware and software requirements.)

TRANVRLS = YES

Specifies that the shortname longname translate table is defined for cross system sharing using VSAM record level sharing.

TRANVRLS =NO

Specifies that the shortname longname translate table is not defined for cross system sharing using VSAM record level sharing. This is the default.

TRANLANG Determines which SCLM language types require no ASCII/EBCDIC translation to the host (file will be binary transferred).

If files were ASCII text in the eclipse client then when added to SCLM, they will remain in that ASCII codepage.

Format:

- TRANLANG JAVABIN
- TRANLANG DOC
- TRANLANG JPEG

In the above examples dummy Language Translators would be set up in SCLM for these languages. See Chapter 3, “SCLM customization for the SCLM administrator,” on page 39 for more information about SCLM Language translators.

LONGLANG Determines which SCLM language types require long name to short name conversion. Long name to short name translation implies the long name 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.

Format:

- LONGLANG JAVA
- LONGLANG J2EEPART
- LONGLANG DOC
- LONGLANG SQLJ

If the SCLM language is not specified in the LONGLANG keyword, the client file is assumed to be already in host short name format (8 characters or less) and is stored as is.

Note: Comment lines can be added by beginning the line with an asterisk (*).

Figure 2. User of TRANSLATE.conf keywords

See “Example of the TRANSLATE configuration file” for an example including the Figure 2 on page 12.

Example of the TRANSLATE configuration file

As stated above, the TRANSLATE configuration files control a number of things.

Two of these are:

- Whether or not a part is translated from ASCII to EBCDIC.
- Whether or not a part has its name translated from a long name to an 8 character short name.

To show how these two settings could be used to help in deciding how to initially set up this config file here is an example.

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.

- Create an SCLM Language translator based on the sample BWBTRANJ in the SBWBSAMP library. You could call it BINARY (as this is how it will be stored), or if you want to be specific, call it DOC.
- 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 checked out in SCLM, it is transferred down to the Eclipse workspace without any EBCDIC to ASCII conversion. After it is checked out, Word is started allowing you 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.

Override settings in the TRANSLATE.conf file

It is possible to override values set in the TRANSLATE.conf file at a SITE and SCLM Project level. For an explanation of this feature, see “SITE and project-specific options” on page 63.

Step 5a: Configure the SCLM Developer Toolkit HTTP Server

This section describes the setup and customization of an HTTP server if you are planning on configuring SCLM Developer Toolkit to communicate with the z/OS host via an HTTP server. An HTTP server is used when the SCLM Developer Toolkit is not installed with Rational Developer for System z, but under any other installation of Eclipse, such as Rational Application Developer (RAD).

Alternatively if you plan to use SCLM Developer Toolkit with Rational Developer for System z, then RSE (Remote Systems Explorer) is used as the communication mechanism. This is covered in “Step 5b: Configure Remote Systems Explorer” on page 18.

It is recommended that the HTTP server is a dedicated Web server to support this interface, though you can optionally incorporate the required SCLM/HTTP configuration directives into an existing HTTP server. See “Customizing an existing HTTP server for SCLM support” on page 16.

Configure the SCLMDT HTTP Server

By default the SCLM/HTTP server is configured to use port 80 though you can choose another suitable dedicated port during customization (1024 or higher as port numbers lower than this are reserved for internal systems use).

If you change the default port number it must be changed in the HTTP server JCL.

The sample setup requires the end user to supply a valid z/OS user ID and password when accessing the host system using this interface.

Note: For additional information about configuring IBM HTTP Web servers, review these IBM manuals:

- HTTP Server Planning, Installing and Using (SC31-8690)
- OS/390 e-business Infrastructure: IBM HTTP Server V5.1 for OS/390 (SG24-5603-00)

The following sections outline the steps for customizing the supplied samples and starting the HTTP server.

By default the HTTP server configuration file and the environment file are stored in the SCLM Developer Toolkit CONFIG directory. Optionally these files can be copied to another user directory or existing server configuration and environment files. In all cases the HTTP server started task must be customized to reflect the appropriate directory.

HTTP server configuration file customization

Customize the sample HTTP configuration file `httpd.conf` (which is stored in the CONFIG directory specified by setup job BWBINST1) by following the instructions in the configuration file for the changes that are needed. The following directives need to be reviewed:

Table 3. httpd.conf customization

Directive	Description of change
Port	Leave as port 80 or change to a valid port number as specified in the HTTP server JCL. It is recommended to reserve the port number.
Protection	Change SCLMDTWB to the name of the HTTP server job. For information about using the Protection directive, see the HTTP Server Guide.
PidFile AccessLog ErrorLog	Change /var/SCLMDT to the appropriate path if a different path to the default was selected.
Pass and Exec Directives	Change /var/SCLMDT to the appropriate path if a different path to the default was selected. Change /usr/lpp/SCLMDT to the appropriate path of the bin installation directory if a different path to the default was selected.

Table 3. *httpd.conf* customization (continued)

Directive	Description of change
Non-standard codepage translation in SCLM Developer Toolkit	<p>If you require different ASCII/EBCDIC codepage translation other than standard default (IBM-1047/ISO8859-1) the following parameters must be coded in the <i>httpd.conf</i> file for the HTTP server:</p> <pre>DefaultFsCp ebclic-codepage DefaultNetCp ascii-codepage</pre> <p>For example, for Japanese translation the required codepages would be:</p> <pre>DefaultFsCp IBM-939 DefaultNetCp IBM-932C</pre>

HTTP server environment file customization

Customize the sample HTTP environment variables file *httpd.env* (which is stored in the installation directory specified by install job BWBINST1) by following the instructions in the environment file for the changes that are needed.

Table 4. *httpd.env* customization

Directive	Description of change
PATH	Ensure the PATH directive has the correct Java path directory. Also ensure the current directory indicated by a '.' is included in the PATH (for example, /bin.: /usr/sbin: /usr/lpp/internet/bin: /usr/lpp/internet/sbin: /usr/lpp/java/J1.4/bin).
CGI_DTWORK	This directive determines the WORKAREA directory path that is used for temporary files. The default is: CGI_DTWORK=/var/SCLMDT
CGI_DTCONF	This directive determines the CONFIG directory path where the configuration files are stored. The default is: CGI_DTCONF=/etc/SCLMDT
CGI_TRANTABLE	This directive determines the name of the translate table used in short to long name translation. This VSAM file is discussed in "Step 6: Configure long/short name table VSAM file" on page 21. The default is: CGI_TRANTABLE=BWB.LSTRANS.FILE

HTTP server JCL/STARTED TASK customization

Before the HTTP server can be submitted, the following tasks must be performed.

1. Copy the sample batch job BWBSRVR from the installed sample library, SBWBSAMP, to a JCL Library or PROCLIB data set and customize to your site-specific standards by following the instructions in the sample.
2. Issue the CAPS OFF command to ensure that case sensitive values do not get changed to upper case.
3. It is recommended to make this HTTP server job a started task, but it can also be run as a standalone job to test the HTTP server JCL.
4. If performing foreground Java builds it is recommended to use a region size of 512M in the HTTP server job.
5. RACF considerations:
 - Create a RACF OMVS segment for the user ID assigned to the HTTP server.

Configure the SCLMDT HTTP Server

- The HTTP server owning user ID requires execute access to the /usr/lpp/internet/sbin files and read/write access to the LOGS directory referenced in the httpd.conf file. This is defaulted to /var/SCLMDT/LOGS.
6. The default port to be used is 80. If you change this to a specific dedicated port you must also change the port number in the httpd.conf configuration file to match the port number in the started task JCL.
 7. If the BWB* modules are not stored in the LINKLIST then edit the STEPLIB to specify the load library containing these modules. By default these are in the SBWBLOAD library.
 8. Module BWBTSOW must be stored in an APF authorized load library.
 9. Ensure a REXX/370 runtime environment on the host exists or alternatively use the REXX/370 Alternate Library.

Note: The user ID assigned to the HTTP server must have READ authority to the BPX.SERVER resource in the FACILITY class. If this resource is not defined, UID 0 is required.

Customizing an existing HTTP server for SCLM support

Follow the instructions below if you optionally choose to incorporate the SCLM Developer Toolkit support into an existing HTTP server.

Add the following pass/exec directives in the httpd.conf configuration file:

Pass	/J2EPUT/	/var/SCLMDT/WORKAREA/*
Pass	/DWGET/	/var/SCLMDT/WORKAREA/*
Pass	/DWTRANSFER/	/var/SCLMDT/WORKAREA/*
Pass	/BWBIVP.html	/usr/lpp/SCLMDT/bin/BWBIVP.html
Pass	/SCLMDW.html	/usr/lpp/SCLMDT/bin/SCLMDW.html
Pass	/DT*	/usr/lpp/SCLMDT/bin/DT*
Exec	/BWBCALL	/usr/lpp/SCLMDT/bin/BWBCALL
Exec	/BWBIVP.cgi	/usr/lpp/SCLMDT/bin/BWBIVP.cgi

Figure 3. Pass and exec directives

Note: You should customize the example as follows:

- Replace /usr/lpp/SCLMDT with your bin installation directory.
- Replace /var with your WORKAREA directory.

Start the SCLM Developer Toolkit HTTP server

Start the Web server by submitting the job or if it is a started task procedure enter the following command from the z/OS console:

```
Start server_proc_name
```

where server_proc_name is the STC name.

(Ensure the procedure is a member in a PROCLIB data set.)

Check the HTTP server successfully initialized: The server JOBLLOG should contain the following messages:

```
IMW0234I Starting.. httpd
IMW0235I Server is ready.
```

Enabling trace on the HTTP server

To enable trace on the HTTP server, modify the server JCL PARM statement to include one of the tracing levels. For example:

Enabling trace on the HTTP server

```
PARM=('ENVAR("_CEE_ENVFILE=//DD:ENV")/-vv -r //DD:CONF -B -p 80')
```

Enabling trace on the HTTP server

The level of tracing provided is:

- v trace for first level
- vv trace for second level
- mtv for third level
- debug for maximum tracing.

For more information about tracing see the “HTTP Server Planning, Installing and Using” manual.

Note: Trace has an impact on performance and should only be carried out if advised by an IBM representative.

Step 5b: Configure Remote Systems Explorer

This section describes the setup and customization of Remote Systems Explorer (RSE) that can be used by the client to access SCLM on a z/OS host. RSE is used when the SCLM Developer Toolkit is installed with Rational Developer for System z.

If you are not using SCLM Developer Toolkit with Rational Developer for System z, then the mechanism used to communicate with the z/OS host is HTTP. This is described in “Step 5a: Configure the SCLM Developer Toolkit HTTP Server” on page 13.

The installation and configuration of the RSE component of Rational Developer for System z is covered in the *Program Directory for IBM Rational Developer for System z (GI11-8298-00)* and *IBM Rational Developer for System z Host Configuration Guide (SC31-6930-02)*. This section guides you through the steps to add SCLM Developer Toolkit specific settings to enable it to work through an RSE connection.

You need to know the customization directory of the RSE component of Rational Developer for System z before you begin as modifications will need to be made to files contained there.

Customizing the RSE Environment file

Locate the rsed.envvars file that your RSE connection will be using, by default in /usr/lpp/wd4z/rse/lib. At the bottom of the file, copy in the SCLM Developer Toolkit member containing the RSE environment variables. This member is BWBRSED and can be found in the installed sample library SBWBSAMP.

The following environment variables used by SCLM Developer Toolkit should already have been defined in rsed.envvars during base RSE customization:

- _CMDSERV_BASE_HOME
- _CMDSERV_CONF_HOME
- _CMDSERV_WORK_HOME

Customize the variables by following the instructions in the table below:

Table 5. RSE Environment file customization

Directive	Description of change
CGI_DTCONF	Determines the base path of the /CONFIG directory (configuration files are stored here). The default is: CGI_DTCONF=\$_CMDSERV_CONF_HOME

Table 5. RSE Environment file customization (continued)

Directive	Description of change
CGI_DTWORK	Determines the base path of the /WORKAREA directory (workarea files are stored here). The default is: CGI_DTWORK=\$_CMDSERV_WORK_HOME
CGI_TRANTABLE	Determines the name of the translate table used in short to long name translation. This VSAM file is discussed in “Step 6: Configure long/short name table VSAM file” on page 21. The default is: CGI_TRANTABLE=BWB.LSTRANS.FILE
STEPLIB	Determines where MVS load modules are run from, linklist or steplib. If SCLMDT's BWB load modules must be located through steplib, ensure that the following directives, which are part of your default rsed.envvars file, are set up correctly: <ul style="list-style-type: none"> • \$_CMDSERV_BASE_LOAD=BWB.SBWBLOAD • STEPLIB=\$_CMDSERV_BASE_LOAD <p>If other load modules, like those for Bizz, must be accessed through steplib, add the following line to the end of rsed.envvars:</p> <ul style="list-style-type: none"> • if the last steplib directive defined earlier in rsed.envvars equals STEPLIB=NONE STEPLIB=BZZ.SBZZLOAD • if the last steplib directive defined earlier in rsed.envvars does not equal STEPLIB=NONE STEPLIB=\$STEPLIB:BZZ.SBZZLOAD <p>Additional data sets are separated by a colon (:).</p>
_SCLM_DT	Determines the path of the /bin installation directory. The default is: \$_CMDSERV_BASE_HOME
_SCLM_J2EPUT	Determines the path of the /WORKAREA directory for put requests. The default is: \$CGI_DTWORK/WORKAREA Using the default value for CGI_DTWORK this resolves to /var/SCLMDT/WORKAREA.
_SCLM_DWGET	Determines the path of the /WORKAREA directory for get requests. The default is: \$CGI_DTWORK/WORKAREA Using the default value for CGI_DTWORK this resolves to /var/SCLMDT/WORKAREA.
_SCLM_DWTRANSFER	Determines the path of the /WORKAREA directory for transfer requests. The default is: \$CGI_DTWORK/WORKAREA Using the default value for CGI_DTWORK this resolves to /var/SCLMDT/WORKAREA.

Table 5. RSE Environment file customization (continued)

Directive	Description of change
_SCLM_BASE	Determines the path of the /WORKAREA directory for all other requests. The default is: \$CGI_DTWORK/WORKAREA Using the default value for CGI_DTWORK this resolves to /var/SCLMDT/WORKAREA.
_SCLM_BWBCALL	Determines the location of BWBCALL and BWBCALLR scripts. The default is: \$_SCLM_DT/bin/BWBCALL Using the default value for _SCLM_DT this resolves to /usr/lpp/SCLMDT/bin/BWBCALL.
_BPX_SHAREAS=YES	Specifies that the child process created by spawn will run in the same address space as the parent's under certain conditions. This value should always be: _BPX_SHAREAS=YES

Comment lines can be added by beginning the line with a hash (#). See the sample rsed.envvars additions below.

```
#ANT_HOME=/u/antdirectory/Ant/apache-Ant.1.6.2
CGI_TRANTABLE=BWB.LSTRANS.FILE
#CGI_DTCONF=$_CMDSERV_CONF_HOME
#CGI_DTWORK=$_CMDSERV_WORK_HOME
#_SCLM_DT=$_CMDSERV_BASE_HOME
#_SCLM_J2EEPUT=$CGI_DTWORK/WORKAREA
#_SCLM_DWGET=$CGI_DTWORK/WORKAREA
#_SCLM_DWTRANSFER=$CGI_DTWORK/WORKAREA
#_SCLM_BASE=$CGI_DTWORK/WORKAREA
#_SCLM_BWBCALL=$_SCLM_DT/bin/BWBCALL
#STEPLIB=$STEPLIB:BZZ.SBZZLOAD
_BPX_SHAREAS=YES
```

Figure 4. Sample rsed.envvars

Customizing the RSE environment setup script

RSE itself has multiple connection methods, directly via a daemon or using a script started by REXEC or SSH. If you use the REXEC/SSH connection method, changes have to be made to setup.env.zseries, the script that sets up the RSE environment variables. This script is stored in the same location as rsed.envvars (default /usr/lpp/wd4z/rse/lib). At the bottom of the file, copy in the SCLMDT member containing the RSE environment export statements. This member is BWBREXEC and can be found in the installed sample library SBWBSAMP.

Activating the RSE Environment file

In order to pick up the SCLM Developer Toolkit variables it is recommended to:

- Disconnect any RSE connections on client machines
- Close Rational Developer for System z on any client machines
- Stop and Start INETD. This requires UID 0 and BPX.DAEMON permission.

Note: If the new variables are not used after disconnecting all clients, stop and start INETD with an authorized userid. This requires BPX.DAEMON and possibly UID 0 permission, as described in *IBM Rational Developer for System z Host Configuration Guide (SC31-6930-02)*.

Step 6: Configure long/short name table VSAM file

SCLM Developer Toolkit provides the ability to store long name files (which are files with names greater than 8 characters or in mixed case) into SCLM. This is achieved through the use of a VSAM file that contains the mapping of the long file name to the 8 character member name used in SCLM.

For versions previous to z/OS V1.8, this facility is provided via a base ISPF/SCLM PTF that addresses APAR OA11426. This PTF must be applied before the long-to-short name translation can be used. If you are on z/OS V1.8 (or higher), you do not need this PTF.

If this installation of SCLM Developer Toolkit is going to use this facility then the PTF providing the facility must be installed and the VSAM file containing the mapping from long-to-short name can then be allocated. To do so, the following sample JCL, found in member FLM02LST in the ISPF sample library ISP.SISPSAMP, can be modified and submitted.

You need update authority on this file.

```
//FLM02LST JOB <JOB PARAMETERS>
//* -----
//* ALLOCATION OF LONGNAME/SHORTNAME VSAM FILE
//*
//* THIS JOB ALLOCATES THE LONGNAME TO SHORTNAME TRANSLATE FILE.
//* THIS TRANSLATE FILE IS REQUIRED FOR THE FOLLOWING SCLM SUITE
//* PRODUCTS - SCLM DEVELOPER TOOLKIT AND SCLM ADMIN TOOLKIT.
//* THE ONE TRANSLATE FILE IS RECOMMENDED TO BE DEFINED AND USED
//* FOR ALL SCLM PROJECTS.
//*
//*
//* CAUTION: THIS IS NEITHER A JCL PROCEDURE NOR A COMPLETE JOB.
//* BEFORE USING THIS SAMPLE, YOU WILL HAVE TO MAKE THE
//* FOLLOWING MODIFICATIONS:
//*
//* 1) ADD THE JOB PARAMETERS TO MEET YOUR SYSTEM REQUIREMENTS
//*
//* 2) CHANGE ALL REFERENCES OF HLQ.LSTRANS.FILE BELOW TO YOUR
//*    REQUIRED NAMING CONVENTION FOR THE SCLM TRANSLATE FILE.
//*
//* 3) MODIFY CYLINDERS (PRIMARY SECONDARY)
//*
//* 4) SPECIFY THE VOLUME VVVVVV ON WHICH IT WILL BE ALLOCATED
//* -----
```

Figure 5. Sample Long/Short Translate VSAM file JCL (Part 1 of 2)

Configure long/short name table VSAM file

```
//IDCAMS EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DELETE HLQ.LSTRANS.FILE
SET MAXCC=0
DEFINE CLUSTER(NAME(HLQ.LSTRANS.FILE)
               RECSZ(58 2048)
               INDEXED
               CYLINDERS(1 1)
               VOLUMES(VVVVVV)
               SHR(3,3)
               KEYS (8 0))
DATA(NAME(HLQ.LSTRANS.FILE.DATA))
INDEX(NAME(HLQ.LSTRANS.FILE.INDEX))

/* DEFINE ALTERNATE INDEX WITH NONUNIQUE KEYS -> ESDS */
DEFINE ALTERNATEINDEX (
    NAME(HLQ.LSTRANS.FILE.AIX)
    RELATE(HLQ.LSTRANS.FILE)
    RECORDSIZE(58 2048)
    CYLINDERS(1 1)
    VOLUMES(VVVVVV)
    KEYS(50 8)
    NONUNIQUEKEY
    UPGRADE )
DATA (
    NAME(HLQ.LSTRANS.FILE.AIX.DATA) )
INDEX (
    NAME(HLQ.LSTRANS.FILE.AIX.INDEX) )

/**
/** -----
/** NOTE: THE FOLLOWING STEP WILL GET RC=4 DUE TO THE ALTERNATE
/** INDEX BEING EMPTY. THE MESSAGES RETURNED ARE AS FOLLOWS.
/**
/** IDC3300I ERROR OPENING HLQ.LSTRANS.FILE
/** IDC3351I ** VSAM OPEN RETURN CODE IS 100
/** IDC0005I NUMBER OF RECORDS PROCESSED WAS 1
/** IDC0001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 4
/**
/** -----
//IDCAM2 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//INITREC DD *
INITREC1
/*
//SYSIN DD *
        REPRO INFILE(INITREC) -
            OUTDATA
```

Figure 5. Sample Long/Short Translate VSAM file JCL (Part 2 of 2)

For more information about the Long to Short name translation process see Appendix C, “Long/short name translation table,” on page 97.

Step 7: Install and customize Ant

This step is required if you plan to use the JAVA/J2EE build support in SCLM.

Ant is freely available and can be downloaded from <http://ant.apache.org/>. Ant text files and scripts are distributed in ASCII format and require an ASCII/EBCDIC translation to run on z/OS in UNIX System Services. A sample translate script has been supplied in the SCLM Developer Toolkit SBWBSAMP library in sample member BWBTRANT and a sample copy job to copy the translate script into the appropriate Ant directory in sample member BWBCPANT. Follow the steps below to implement Ant on z/OS:

1. Download in binary format, the latest Ant compressed file into the z/OS UNIX System Services file system and unzip into the appropriate directory. It is recommended that you download the .zip version of ANT due to problems that might be encountered on z/OS when unzipping versions of suffix format tar.gz or tar.bz2. Use the JAR extract command `jar -xf ANTfile.zip` to unzip on z/OS. A Java bin directory must be in your local z/OS USS PATH to use the JAR command. Otherwise, fully qualify the command with the java bin location (for example, `/usr/lpp/java/J1.4/bin/jar -xf ANTfile.zip`).
2. Customize the install copy member BWBCPANT to include the Ant installation directory and run the job to copy the translate script into the directory (review the instructions contained within the sample member BWBCPANT).
3. Using whatever tool you use to list files in z/OS Unix Systems Services (OMVS or ISHELL, for example), to check for successful translation, locate a text file, such as the file README, within the ANT directory. Then browse to this file using your preferred browsing method, such as OBROWSE or OMVS. If the file is readable, then the translation was successful.
4. Change file permissions for all files under the Ant installation directory to enable all users to read and execute.

For example

```
cd /u/antdirectory/Ant/apache-Ant.1.6.2; chmod -R 755 *
```

5. Before using Ant, set the z/OS UNIX System Services environment variables JAVA_HOME and ANT_HOME.

Note: The JAVA_HOME specified in ANT will be the Java version used at compile time for Java/J2EE projects and will override the JAVA_BIN variable set in the \$GLOBAL file.

- a. JAVA_HOME is required to point to the Java home directory, for example:

```
JAVA_HOME=/usr/lpp/java/IBM/J1.4
```

- b. ANT_HOME is required to point to the Ant installation directory, for example:

```
ANT_HOME=/u/antdirectory/Ant/apache-Ant.1.6.2
```

6. Ant will look for an Ant configuration file in the directory /etc, so we recommend creating an Ant configuration file named ant.conf and adding the variables JAVA_HOME and ANT_HOME. That is, in file /etc/ant.conf set:

```
JAVA_HOME=/usr/lpp/java/IBM/J1.4
```

```
ANT_HOME=/u/antdirectory/Ant/apache-Ant.1.6.2
```

A sample ant.conf file is provided in member BWBANTC in the installed sample library SBWBSAMP. This can be copied to the /etc/ directory as file ant.conf and the JAVA_HOME and ANT_HOME variables modified to the correct value for your installation.

Install and customize Ant

Note: The above directory paths are only sample directory paths. Ensure the correct directory paths are used.

These variables can also be set in other ways:

- Add them to the `rsd.envvars` to expose them to Rational Developer for System z. This applies if you have configured RSE.
- Define them in the system wide profile (`/etc/profile`).
- Modify the `/bin/Ant` file in the `ANT_HOME` directory to have an export statement at the top of the file. For example:

```
export JAVA_HOME=/usr/lpp/java/IBM/J1.4
```

To test that the Ant initialization has been successful:

1. Add the Ant and Java bin directories to the environment variable `PATH`. This `PATH` variable can be added to your `.profile` or you can enter the following `PATH` definition below at the UNIX System Services command line.

Example:

```
export PATH=/u/antdirectory/Ant/apache-Ant.1.6.2/bin:/usr/lpp/java/IBM/J1.4/bin:$PATH
```

2. Run Ant to display the version.

Example:

```
Ant -version
```

This displays the Ant version if Ant is successfully installed.

Note: Setting the `PATH` statement in this way is necessary for testing, not for operational use. During normal SCLM Developer Toolkit build processing, the `ANT_HOME` and `JAVA_HOME` environment variables are set dynamically from the values set in the `$GLOBAL` member. For a detailed description of the `$GLOBAL` member parameters, see “`$GLOBAL` member” on page 62.

Step 8a: Run the IVP to check correct HTTP installation and customization

This Installation Verification Process (IVP) applies if you have configured the HTTP server. The HTTP server must be running and the IVP `pass/exec` directives configured in the `httpd.conf` file for successful verification processing.

From a browser, type the location URL address:

```
http://hostname:portnumber/BWBIVP.html
```

Where:

hostname Is the TCP/IP host name the HTTP server is running on.

port number Is the port used in the job and the `httpd.conf` file (default port 80).

If the HTTP server is running you will be prompted for a valid TSO user ID and password for the system the Web server is started on (Figure 6 on page 26).

Run the IVP to check correct HTTP installation and customization

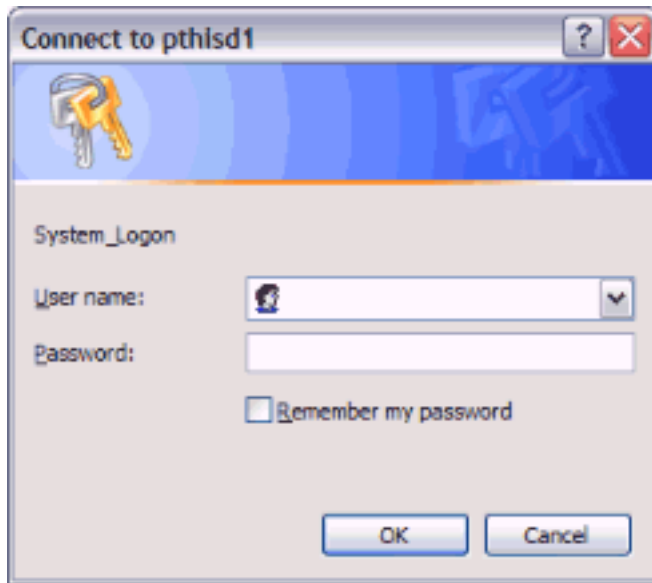


Figure 6. HTTP server logon prompt

After you have entered your TSO user ID and password the browser will initially display the html welcome screen (Figure 7).

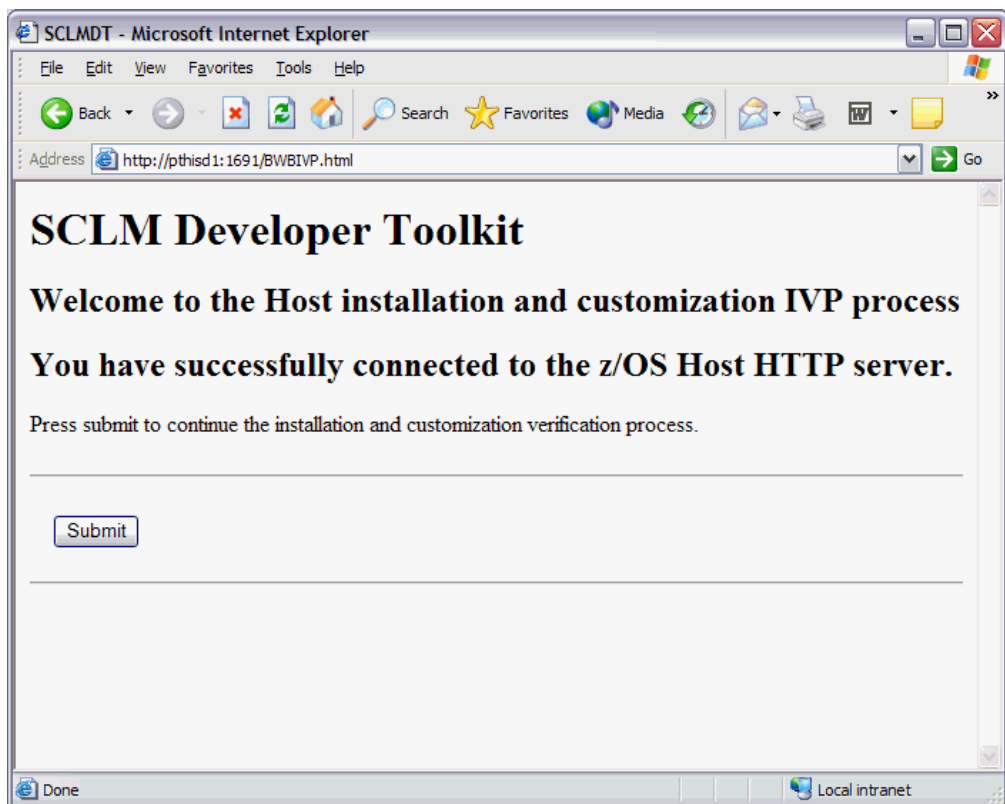


Figure 7. Host installation and customization welcome screen

Run the IVP to check correct HTTP installation and customization

If you fail to connect then check that:

- The HTTP server has successfully initialized.
- The z/OS UNIX System Services file system mount point containing the SCLM Developer Toolkit installation is mounted.
- The hostname:port are correct (try pinging the Hostname).
- There are no firewall restrictions.
- The PASS directive in the httpd.conf file is set correctly:

```
Pass      /BWBIVP.html          /usr/lpp/SCLMDT/bin/BWBIVP.html
```

After you receive the welcome screen, continue with the IVP, which checks and validates your installation and customization process.

The sample screens (Figure 8 through to Figure 10 on page 28) give an example of possible validation responses.

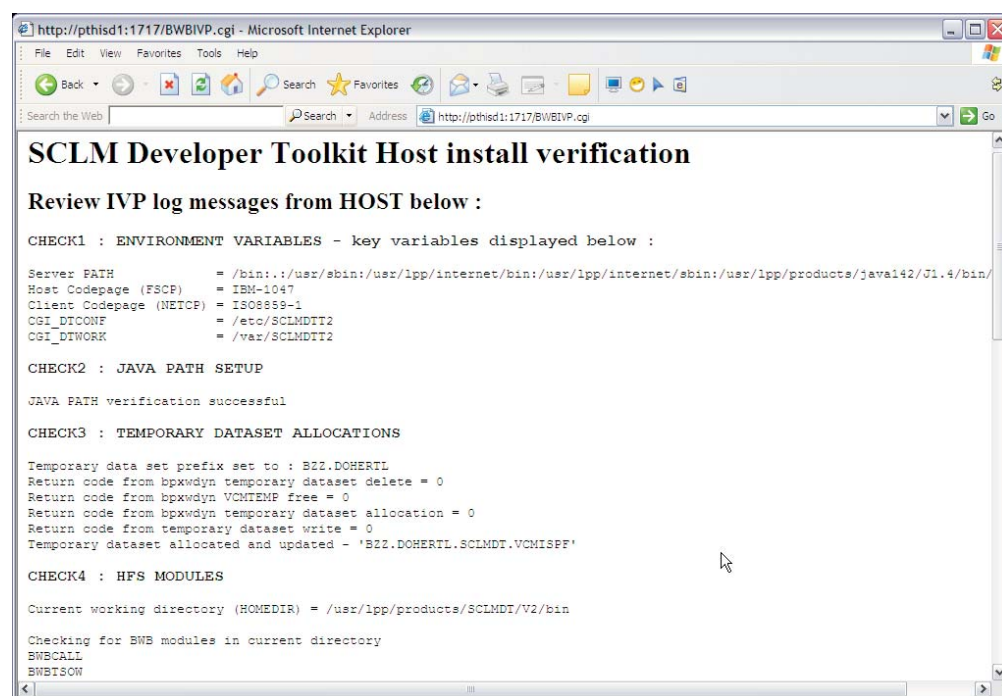
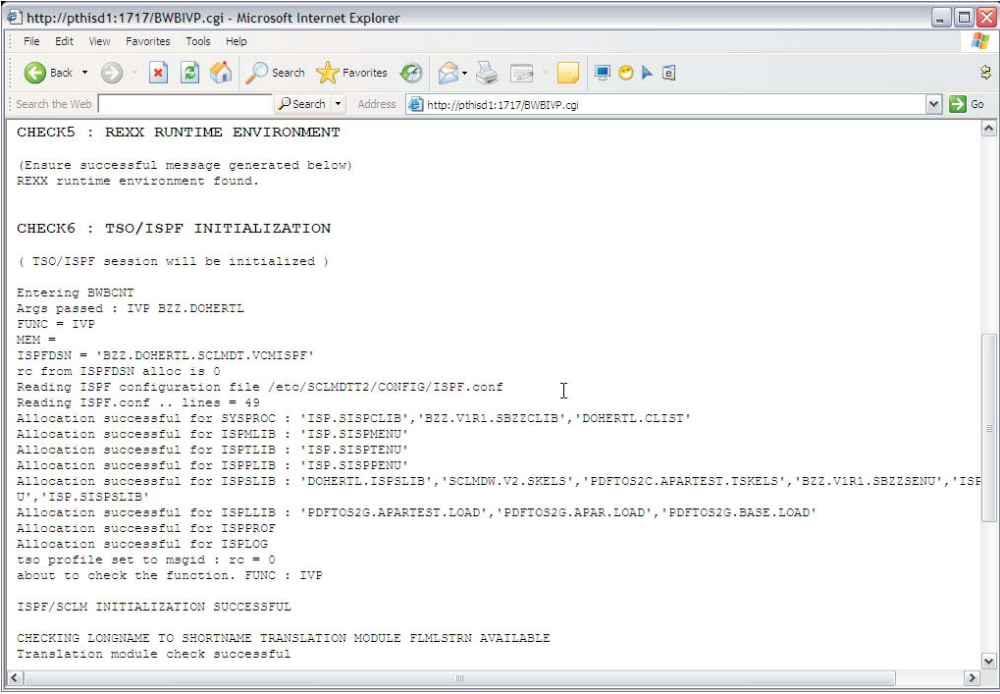


Figure 8. An example of validation responses (part 1)

Run the IVP to check correct HTTP installation and customization



```
http://pthisd1:1717/BWBIVP.cgi - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Home Search
Search the Web Search Address http://pthisd1:1717/BWBIVP.cgi Go

CHECK5 : REXX RUNTIME ENVIRONMENT
(Ensure successful message generated below)
REXX runtime environment found.

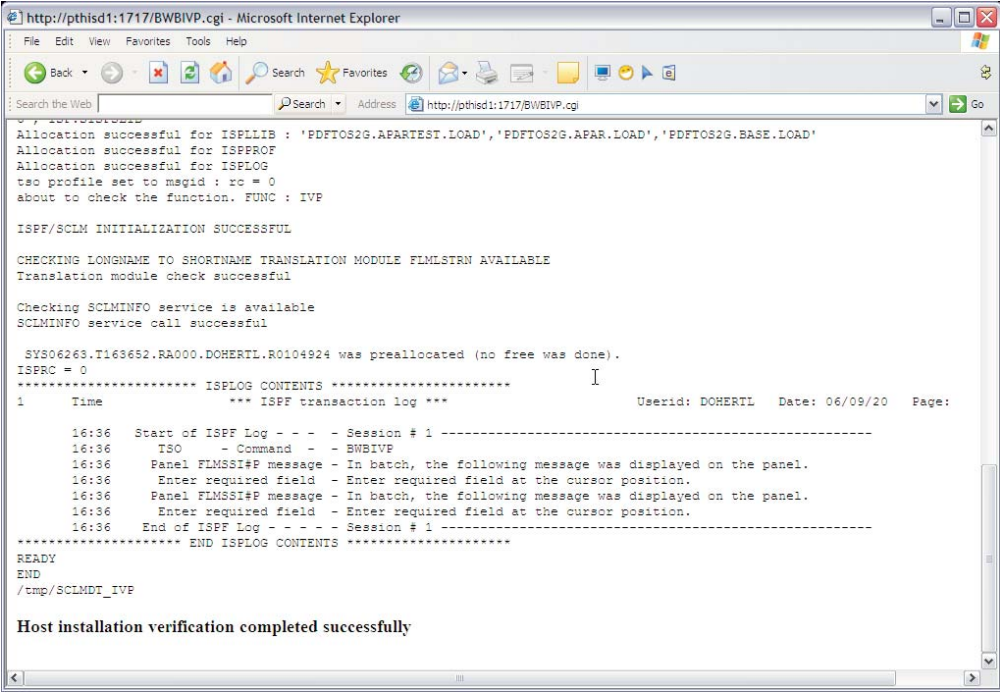
CHECK6 : TSO/ISPF INITIALIZATION
( TSO/ISPF session will be initialized )

Entering BWBCNT
Args passed : IVP BZZ.DOHERTL
FUNC = IVP
MEM =
ISPFDSN = 'BZZ.DOHERTL.SCLMDT.VCMISPF'
rc from ISPFDSN alloc is 0
Reading ISPF configuration file /etc/SCLMDT2/CONFIG/ISPF.conf
Reading ISPF.conf .. lines = 49
Allocation successful for SYSPROC : 'ISP.SISPCLIB','BZZ.V1R1.SBZZCLIB','DOHERTL.CLIST'
Allocation successful for ISPFMLIB : 'ISP.SISPMENU'
Allocation successful for ISPFLLIB : 'ISP.SISPPENU'
Allocation successful for ISPFSTRN : 'DOHERTL.ISPCLIB','SCLMDW.V2.SKELS','PDFTOS2C.APARTEST.TSKELS','BZZ.V1R1.SBZZSENU','ISP
U','ISP.SISPSLIB'
Allocation successful for ISPFLLIB : 'PDFTOS2G.APARTEST.LOAD','PDFTOS2G.APAR.LOAD','PDFTOS2G.BASE.LOAD'
Allocation successful for ISPFPROF
Allocation successful for ISPFLOG
tsso profile set to msqid : rc = 0
about to check the function. FUNC : IVP

ISPF/SCLM INITIALIZATION SUCCESSFUL

CHECKING LONGNAME TO SHORTNAME TRANSLATION MODULE FLMLSTRN AVAILABLE
Translation module check successful
```

Figure 9. An example of validation responses (part 2)



```
http://pthisd1:1717/BWBIVP.cgi - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Home Search
Search the Web Search Address http://pthisd1:1717/BWBIVP.cgi Go

Allocation successful for ISPFLLIB : 'PDFTOS2G.APARTEST.LOAD','PDFTOS2G.APAR.LOAD','PDFTOS2G.BASE.LOAD'
Allocation successful for ISPFPROF
Allocation successful for ISPFLOG
tsso profile set to msqid : rc = 0
about to check the function. FUNC : IVP

ISPF/SCLM INITIALIZATION SUCCESSFUL

CHECKING LONGNAME TO SHORTNAME TRANSLATION MODULE FLMLSTRN AVAILABLE
Translation module check successful

Checking SCLMINFO service is available
SCLMINFO service call successful

SYS06263.T163652.RA000.DOHERTL.R0104924 was preallocated (no free was done).
ISPRC = 0
***** ISPFLOG CONTENTS *****
1      Time      *** ISPF transaction log ***      Userid: DOHERTL      Date: 06/09/20      Page:
16:36 Start of ISPF Log - - - Session # 1 -----
16:36 TSO - Command - BWBIVP
16:36 Panel FLMSIIF message - In batch, the following message was displayed on the panel.
16:36 Enter required field - Enter required field at the cursor position.
16:36 Panel FLMSIIF message - In batch, the following message was displayed on the panel.
16:36 Enter required field - Enter required field at the cursor position.
16:36 End of ISPF Log - - - Session # 1 -----
***** END ISPFLOG CONTENTS *****
READY
END
/tmp/SCLMDT_IVP

Host installation verification completed successfully
```

Figure 10. An example of validation responses (part 3)

Testing connection to the HTTP server

At any time the server connection can be tested without running the full IVP check.

From a browser, type the location URL address:

`http://hostname:portnumber/SCLMDW.html`

Where:

hostname Is the TCP/IP host name the HTTP server is running on.

port number Is the port used in the job and the httpd.conf file (default port 80).

You are prompted for a valid user ID and password for the system the Web server is started on.

The browser then displays the message shown in Figure 11.

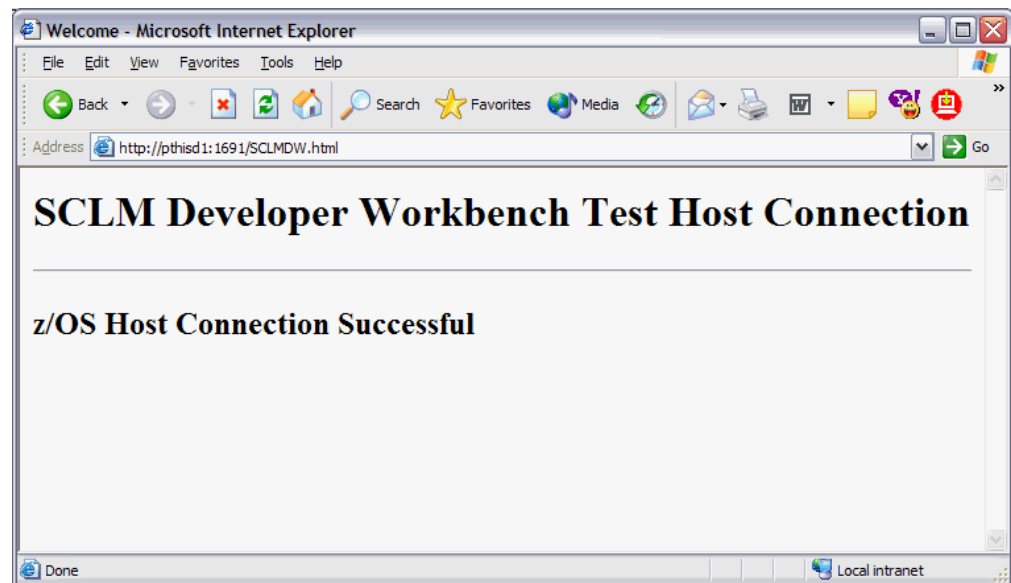


Figure 11. Server Connection successful message

Step 8b: Run the IVP to check correct RSE installation and customization

This Installation Verification Process (IVP) applies if you have configured the RSE connection. This connection is used when Developer Toolkit is installed as a Rational Developer for System z plug-in. A z/OS RSE connection must be configured and running. The SCLM Developer Toolkit directives must be configured in the `rsd.envvars` file for successful verification processing. For more information about this, see “Step 5b: Configure Remote Systems Explorer” on page 18.

Note: Rational Developer for System z also provides several IVP tests for the RSE component. Refer to the *IBM Rational Developer for System z Host Configuration guide* (SC31-6930-02) for more information on these.

Follow these steps to invoke the SCLM Developer Toolkit IVP:

1. In Rational Developer for System z, ensure the Remote Systems Explorer perspective is open. For the z/OS connection that will be connecting to SCLM, right click on the USS Shells node and then select Launch Shell.
2. In the command line in the shell, change directory to the installation directory of the SCLM Developer Toolkit z/OS UNIX System Services file system modules. By default this will be `/usr/lpp/SCLMDT/bin`. To do this use the `cd` command as shown in Figure 12.

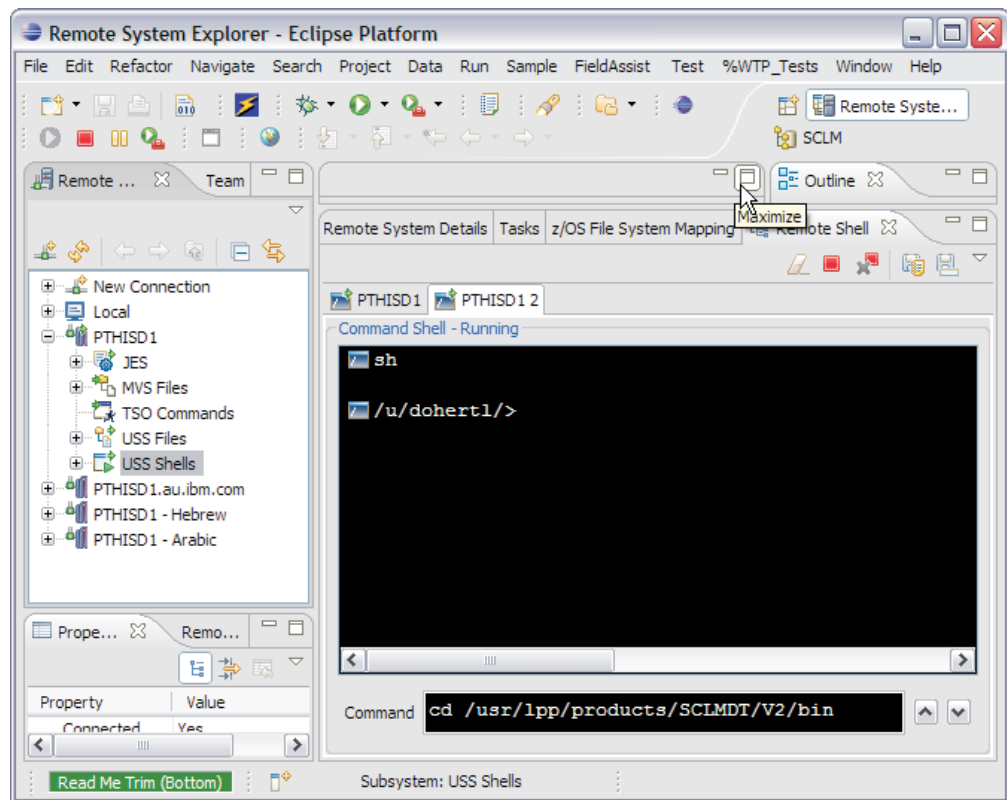


Figure 12. Change directory command

Run the IVP to check correct RSE installation and customization

3. In the command line enter BWBIVPR.cgi to run the IVP script. The script will run in the shell and will give you a number of validation responses as it steps through the different tests in the IVP. After this completes, you can scroll back up to examine all of the responses to ensure that the IVP worked successfully.
4. An example of the IVP validation responses is shown in Figure 13.

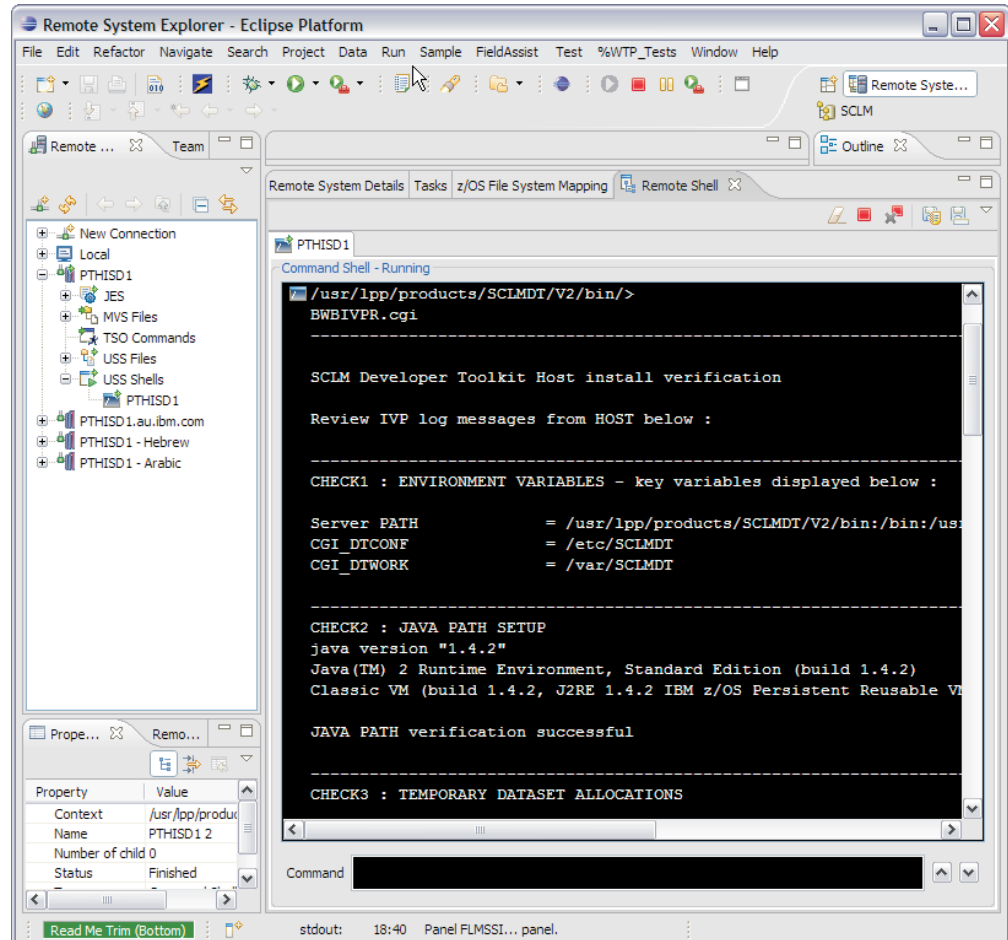


Figure 13. IVP validation responses

Chapter 2. Installing the Eclipse-based client onto the PC

If you have already installed Rational Developer for System z onto your PC, you do not need to install the client because the SCLM Developer Toolkit plug-in must already be installed. However if you have not installed SCLM Developer Toolkit as part of your Rational Developer for System z installation, then you can install just the SCLM Developer Toolkit plug-in into an existing Eclipse by following the instructions contained in this chapter.

Note: SCLM Developer Toolkit only supports the IBM Java Runtime Environment (JRE) Version 1.5.

Furthermore, if you want to install the SCLM Developer Toolkit into a different Eclipse other than Rational Developer for System z, you must carry out the installation steps that follow.

Preparing for installation

To prepare for installation, you need to meet the following media requirements and hardware and software requirements.

Media requirements

To install SCLM Developer Toolkit onto your workstation, you must have access to the SCLM Developer Toolkit installation CD or electronic images.

Hardware and software requirements

The following information gives hardware and software requirements for SCLM Developer Toolkit.

Prerequisites for SCLM Developer Toolkit

IBM SCLM Developer Toolkit is a licensed program to support users who want to store and build distributed code in SCLM on z/OS, as well as work with traditional z/OS artefacts through an IDE.

Hardware requirements: SCLM Developer Toolkit requires the hardware as shown in Table 6 on page 34.

Installing the Eclipse-based client onto the PC

Table 6. Required hardware

Processor	Intel Pentium III 800 MHz or higher is required
Memory	768MB RAM is required, though 1GB RAM or higher is recommended
Disk space	250 MB minimum disk space is required to install SCLM Developer Toolkit. This is assuming you are installing the complete Eclipse installation along with the SCLM Developer Toolkit plug-in. You will be installing SCLMDT using IBM Installation manager so you will need an additional 120 MB minimum disk space to install this (if you do not already have it installed). Installation also requires an additional 700MB of temporary space for product update installation files and other files.
Monitor and video card	A VGA display of 1024x768 or higher is required

Supported operating systems: SCLM Developer Toolkit requires any of the following operating systems:

Table 7. Required operating systems

Windows XP	Professional with Service Pack 2
Windows Server 2003	Standard or Enterprise Edition with Service Pack 1
Windows Vista	Business Edition

Installing SCLM Developer Toolkit

Install SCLM Developer Toolkit using IBM Installation Manager. If you do not have IBM Installation Manager, it is provided on the SCLM Developer Toolkit CD.

The following steps work you through installing the IBM Installation Manager and then installing the SCLM Developer Toolkit.

Step 1. Install from CD or electronic image

You can install SCLM Developer Toolkit directly from CD or electronic image. After the electronic image has been extracted, the image will contain the same contents as the CD. Installation is performed by IBM Installation Manager and this is installed if necessary.

- Insert the installation CD and run `install.exe` (if it hasn't auto-started for you already). Or if you are installing from an extracted electronic image, go to the directory where the image is extracted and run `install.exe`.
 - If you do not have IBM Installation Manager installed on your system, you will be prompted to install this first. This is used to install the Eclipse workbench and all future updates. Follow the on-screen instructions to install the IBM Installation Manager.
 1. Accept the license agreement and click **Next**.
 2. Change the default installation location if required and Click **Next**.
 3. Click **Install** to begin the installation.
 4. When the installation completes, click **OK** to continue with the SCLM Developer Toolkit client installation.
 5. The Installation Manager starts up automatically and is already set up to install the SCLM Developer Toolkit. Proceed to Step 2.

- If you do have IBM Installation Manager installed it starts up automatically and is already set up to install the SCLM Developer Toolkit.

IBM Installation Manager is installed and you can proceed to Step 2.

- Alternatively if you have IBM Installation Manager installed you can start it and point to a CD or extracted electronic image to install SCLM Developer Toolkit.
 1. Start the IBM Installation Manager (**Start > All Programs > IBM Installation Manager > IBM Installation Manager**).
 2. In the Installation Manager window, go to **File > Preferences > Repositories** and click the **Add Repository** button.
 3. Click the **Browse** button and select the disk1 directory on the CD or from the extracted image.
 4. Click **OK** to save the repository.
 5. Click **OK** to save your changes and close the Preferences Window.
 6. Click the **Install Packages** button on the Quick Start panel. Proceed to Step 2.

Step 2. Install SCLM Developer Toolkit

From the **Install Packages** screen of the IBM Installation Manager:

1. On the Install panel, select the SCLM Developer Toolkit V3.1 from the list of available offerings if it is not already selected and click **Next**.
2. On the License panel, accept the license agreement and click **Next**.
3. On the Location panel:
 - a. **Specify the Shared Resources Directory:** Assuming you have not installed any other offerings yet using the IBM Installation Manager, specify the directory you want to use as your Common Component Directory.

Note: This directory is used by all offerings installed by the Installation Manager, and is the directory where the majority of files (for example, all Eclipse features and plug-ins) are installed, so make sure you have plenty of disk space here.

After this location has been specified you will not be able to change it on the Location panel for other offerings you install later.

Click **Next**.

- b. **Specify the Installation Directory:** On the next screen specify the installation directory where you want to install the offering.

Note: Only certain files specific to this offering are installed to this location. The majority of the offering (for example, all of the Eclipse plug-ins and features) is installed to the Common Component Directory which was specified previously.

Click **Next**.

- c. **Extend an Existing Eclipse Installation:** On the next screen you can choose to install the SCLM Developer Toolkit plug-in into an existing Eclipse. By default SCLM Developer Toolkit is installed into its own installation of Eclipse. If you want to add the plug-in to an existing Eclipse specify the installation location of the Eclipse IDE.

Note: SCLM Developer Toolkit only supports the IBM Java Runtime Environment (JRE) Version 1.5. Installing into an existing Eclipse environment that uses an unsupported JRE will cause problems running SCLM Developer Toolkit.

Click **Next**.

Installing the Eclipse-based client onto the PC

4. On the Features panel:
 - a. **Select required languages:** Select any offered language packs you want to install. Click **Next**.
 - b. **Select additional features:** On the next screen select any additional features that are offered that you might require. The SCLM Developer Toolkit is the default feature and is selected by default.
5. Review the Summary panel and click **Install** to begin the install.
6. When the **Installation completed** panel appears, click **Finish**. This exits the Installation Manager.

IBM SCLM Developer Toolkit is installed.

To launch the workbench from the Windows Start menu, select **Start > All Programs > IBM SCLM Developer Toolkit > IBM SCLM Developer Toolkit > IBM SCLM Developer Toolkit**.

Part 2. Customizing SCLM Developer Toolkit

Chapter 3. SCLM customization for the SCLM

administrator	39
Language translators for JAVA/J2EE support	39
JAVA/J2EE build summary	40
JAVA/J2EE build objects generated	41
SCLM language definitions	42
SCLM types	43
SCLM member formats	45
\$GLOBAL	45
J2EE ARCHDEF	46
J2EE Ant build script	49
JAVA/J2EE Ant XML build skeletons	53
Mapping J2EE projects to SCLM	54
Recommendations for mapping J2EE projects to SCLM	57
SCLM Developer Toolkit deployment	58
WebSphere Application Server (WAS)	
deployment	59
SCLM to Unix System Services deployment	59
Secure deployment	60
Public key authentication	60
Other deployment options	60
ASCII or EBCDIC storage options	61
ASCII/EBCDIC language translators	61
\$GLOBAL member	62
SITE and project-specific options	63
Options Definition	67
Example of using combinations of the TRANSLATE.conf overrides	69
Example of using combinations of the BIDIPROP overrides	71

Chapter 4. SCLM security

Build/Promote/Deploy security flag and process flow	73
Security rules and surrogate user ID	73
Build rule format	74
Promote rule format	74
Deploy rule format	74
SAF/RACF BUILD, PROMOTE, DEPLOY, and PROFILE rules	75

Chapter 5. CRON-initiated Builds and Promotes

STEPLIB and PATH requirements	77
CRON Build job execution	78
CRON Build job samples	78

Chapter 3. SCLM customization for the SCLM administrator

This chapter looks at how the SCLM administrator can customize SCLM. It contains the following sections:

- “Language translators for JAVA/J2EE support.”
- “JAVA/J2EE build summary” on page 40.
- “JAVA/J2EE Ant XML build skeletons” on page 53.
- “Mapping J2EE projects to SCLM” on page 54.
- “SCLM Developer Toolkit deployment” on page 58.
- “ASCII or EBCDIC storage options” on page 61.

Language translators for JAVA/J2EE support

SCLM Developer Toolkit requires five new language translators defined in SCLM for JAVA/J2EE support. These language translators are shipped in the SBWBSAMP members as shown below:

Sample Translator	Description
BWBTRANJ	Sample default member translator. No parsing. Similar to SCLM FLM@TEXT. This translator can be customized to create language definitions J2EEPART, J2EEBIN, BINARY, and TEXT.
BWBTRANS	Sample SQLJ language translator. LANG=SQLJ
BWBTRAN1	Sample Java language translator. LANG=JAVA.
BWBTRAN2	Sample JAVA/J2EE language translator incorporating Ant (for multiple Java compiles and JAR, WAR, and EAR builds).
BWBTRAN3	Sample J2EE language translator for SCLM ARCHDEF J2EE support. LANG=J2EEOBJ.

Figure 14. Sample translators

The SCLM administrator will need to copy these samples, rename if required, and then generate them into the PROJDEFS.LOAD library for each SCLM project where Java support is required. These translators are required to be added/compiled in the Project Definition.

A sample project definition for JAVA/J2EE projects and host components is provided in sample BWBSCLM.

The LOADLIB data set containing the BWB* modules must be included in the ISPF ISPLLIB concatenation to access the JAVA/J2EE language translator modules. The ISPLLIB concatenation is customized in the configuration file ISPF.conf.

SCLM DATASETS for JAVA/J2EE:

It is recommended that you create SCLM target source data sets of RECFM=VB, LRECL=1024 for any JAVA/J2EE source that is to be stored in SCLM from the Toolkit client to allow long record types. The editors on the Eclipse-based client create files of variable record length, and to maintain integrity the Host target data

sets in SCLM should also be of RECFM=VB. Using Fixed record length data sets (RECFM=FB) will result in imported members having white spaces appended to end of record.

JAVA/J2EE build summary

Here is a summary of the process that occurs for Java and J2EE builds using the supplied translators.

Note: You can build JAVA/J2EE members or ARCHDEFS directly in TSO/ISPF on the host as well as via the Developer Toolkit Client.

The ARCHDEF contains the members that make up the JAVA/J2EE project and are a short-name representation of how the project exists in an Eclipse workspace.

The ARCHDEF itself is built which invokes a pre-build verify language translator (J2EEANT). The translator reads the J2EE build script, which is referenced in the ARCHDEF by the SINC keyword, and overlays the properties specified into the skeleton Ant XML referenced by properties SCLM-ANTXML **A**. The build script, when generated by the SCLM Developer Toolkit, is stored in SCLM with a language of J2EEANT **1**.

An ARCHDEF generates Java Classes for Java source identified with the INCLD keyword in the ARCHDEF **2**, and each ARCHDEF can also generate a J2EE archive file such as a JAR, WAR, or EAR file. The J2EE object created is dependent on the appropriate build script referenced and use of the ARCHDEF keyword OUT1 **3**, **B**.

When the ARCHDEF is built the pre-build verify language translator associated with the build script (in SCLM type J2EEBLD) runs and determines what parts of the ARCHDEF are required to be rebuilt (including nested ARCHDEFS **4** identified through the use of the INCL keyword in the ARCHDEF). Those parts are then copied into the z/OS UNIX System Services file system workarea and Ant compiles and generates the required JAVA/J2EE objects specified by the build script and ARCHDEF. Any external jar or class references that your IDE project needs to resolve are done so from the path defined in the CLASSPATH_JARS property **C**.

SCLM then processes each individual ARCHDEF component running each language translator associated with the component. The Language translator JAVA, associated with Java source, copies the class files created back into SCLM.

Finally, the ARCHDEF translator determines what J2EE objects have been generated (JAR, WAR, EAR) and copies these parts back into SCLM.

It is essential to create a separate ARCHDEF for each application component that might make up an enterprise application (EAR). That is, an EAR which contains a WAR which contains an EJB JAR should have an ARCHDEF for the JAR, an ARCHDEF for the WAR with an INCL of the EJB JAR ARCHDEF. The EAR ARCHDEF then should include an INCL of the WAR ARCHDEF.

Figure 15 on page 41 shows the corresponding JAR sample.

```

*
* Initially generated on 10/05/2006 by SCLM DT V2
*
LKED   J2EEOBJ          * J2EE Build translator
*
* Source to include in build
*
INCLD  AN000002 V2TEST  * com/Angelina.java          *
INCLD  V2000002 V2TEST  * com/V2Java1.java  2          *
INCLD  V2000003 V2TEST  * V2InnerClass.java          *
*
* Nested SCLM controlled jars to include          *
*
INCL  V2JART1 ARCHDEF   * DateService.jar  4          *
*
* Build script and generated outputs
*
SINC   V2JARB1  1  J2EEBLD  * J2EE JAR Build script  *
OUT1   *        J2EEJAR   * V2TEST.jar  3          *
LIST   *        J2EELIST

```

Figure 15. Sample Jar application (JAR) ARCHDEF

Figure 16 shows the corresponding JAR script.

```

<ANTXML>
<project name="JAVA Project" default="jar" basedir=".">
<property name="env" environment="env" value="env"/>
<property name="SCLM_ARCHDEF" value="V2JAR1"/>
<property name="SCLM_ANTXML" value="BWBJAVA"  A />
<property name="SCLM_BLDMP" value="YES"/>
<property name="JAR_FILE_NAME" value="V2TEST.jar"  B />
<property name="CLASSPATH_JARS" value="/var/SCLMDT/CLASSPATH"/>  C
<property name="ENCODING" value="IBM-1047"/>
</ANTXML>

```

Figure 16. J2EE Build script JAR sample

JAVA/J2EE build objects generated

The following objects are generated:

- Compilation of all Java Source into output classes, stored in SCLM type JAVACLAS.
- Classes stored in SCLM and long/short name stored in Translate tables.
- Optional Jar created (contains classes and might contain other Java project components such as XML/HTML etc in a packaged structure).
- Jar objects stored in SCLM and long/short name stored in Translate table.
- Jar structure determined by the ARCHDEF used. The long names associated with the members in the ARCHDEF determine the Jar packaging format.
- Optional EJB JAR (contains Classes and might contain other Java project components such as XML/HTML/JSP etc in a packaged structure).
- Optional Web WAR file based on J2EE web.xml file in J2EE project and stored in SCLM as above.
- Optional EAR file for deployment based on application.xml in J2EE project and stored in SCLM as above.
- All listing outputs are stored in SCLM type J2EELIST.

SCLM language definitions

The sample translators define the following languages:

Translator	Description
J2EEPART	Language type that specifies a JAVA/J2EE component and defined by sample BWBTRANJ. No particular parsing occurs on build of this language definition. Non-Java source or J2EE components that require ASCII/EBCDIC language conversion can be generically slotted under this language definition if no particular build parsing is required (for example html, XML, .classpath, .project, definition tables). Optionally language definition of TEXT can be used.
J2EEBIN	Language type that specifies JAVA/J2EE Binary or ASCII stored component and defined by sample BWBTRANJ. No particular parsing occurs on build of this language definition. JAVA/J2EE binary files and text files that you want to be stored as ASCII can be generically slotted under this language definition if no particular build parsing is required.
JAVA	Language type for Java source and defined by sample BWBTRAN1. The Java translator determines what type of build has been issued against Java source.

Note: This language definition must be assigned to Java programs if you want to store the Java source in EBCDIC on the host (that is, the source might be viewed and edited directly on the host through ISPF). The advantage of defining programs with this language definition is being able to edit and view the source directly on the z/OS host. The disadvantages are that codepage conversions need to take place when migrating or importing projects from the client to the host.

- Scenario 1: Build issued against individual Java program.
The Java translator compiles source into output classes. Class is stored in SCLM in type JAVACLAS. Javac compile output is stored in type JAVALIST.
Any classpath dependencies can be satisfied by storing dependent JARs in the classpath directory specified in \$GLOBAL member parameter CLASSPATH_JARS. For more information about this, see “\$GLOBAL member” on page 62.
- Scenario 2: Build against ARCHDEF (ARCHDEF calls J2EEANT build script referenced by the SINC keyword) leaves the Ant script specified to do the build. The Java translator itself, when invoked by the ARCHDEF, just copies the output classes into SCLM. An ANT build summary file is stored in JAVALIST. Individual Java components have an output table stored in JAVALIST.

SQLJ	Language type for SQLJ source code defined by sample BWBTRANS. SQLJ Members defined with this language translator invoke the SQLJ language translator at build time. SQLJ source is converted to java source, and compiled into classes and serialized objects (.ser files) in type SQLJSER. Optionally, DBRM members can also be generated into type DBRMLIB.
JAVABIN	Language type that is similar to Java and used when storing Java source as ASCII in SCLM.

J2EEANT	<p>This is the main build translator for JAVA/J2EE builds and this verify translator is invoked when a J2EE ARCHDEF is built. The translator gets invoked because the JAVA/J2EE build script, stored in SCLM type J2EEBLD, is saved in SCLM with a language of J2EEANT. It is then referenced via the SINC keyword in the ARCHDEF.</p> <p>This verify translator determines what parts are required to be built (including nested ARCHDEFs) and depending on the build modes copies these parts into the z/OS UNIX System 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.</p>
J2EEOBJ	<p>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 built in translator J2EEANT and copies these objects into SCLM with the generated short name provided. This translator is referenced by the LKED keyword in the ARCHDEF itself.</p>

Note: All objects such as JAR, WAR, and EAR have their internal zipped source parts in ASCII to distribute to all platforms.

SCLM types

There are a number of SCLM types that need to be created for JAVA/J2EE support. Some of these types are mandatory types and must be created for JAVA/J2EE support to function.

Recommended data set attributes for some typical types

For the following SCLM TYPES the default data set attributes of DSORG=PO TRACKS(1,5) DIR=50 BLKSIZE=0 (system determined) are recommended.

Also, the following record format and record length attributes are recommended:

J2EEBLD recfm=FB lrecl=256

JAVALIST recfm=VB lrecl=255

J2EELIST recfm=VB lrecl=255

JAVACLAS recfm=VB lrecl=256

J2EEJAR recfm=VB lrecl=256

J2EEWAR recfm=VB lrecl=256

J2EEEAR recfm=VB lrecl=256

DBRMLIB recfm=VB lrecl=256

SQLJSER recfm=VB lrecl=256

Additional source dataset types for Java/J2EE recfm=VB lrecl=1024.

Type	Description
J2EEBLD	This SCLM Type is required for Java and J2EE build and deploy processes.

SCLM types

The J2EEBLD type contains:

- J2EEBLD build scripts used to drive the Ant build and deploy process.
- Java and J2EE ANTXML scripts to be invoked for builds and deploys.

Note: Sample Java and J2EE ANTXML scripts are supplied. Generally these scripts require little or no user customization. Site- and user-dependent variables are customized in the J2EEBLD scripts themselves to override default ANTXML variables. (For more information, see “JAVA/J2EE Ant XML build skeletons” on page 53.)

This contains a \$GLOBAL member which is required for both Java and J2EE builds (see “\$GLOBAL member” on page 62).

DBRMLIB	Technically a DB2 type. DBRMLIB is required for SQLJ support. Must be FB=80. Stores the Database request modules.
SQLJSER	This type is required for the J2EE/SQLJ build process. The SQLJSER type stores SQLJ serialized profiles.
ARCHDEF	This contains JAVA/J2EE ARCHDEF members. The long name parts in each ARCHDEF member outline the JAVA/J2EE project structure. The ARCHDEF for a given project can be dynamically created from the client when migrating in new projects or updated when adding new parts to an existing project. The SCLM ARCHDEF is the primary SCLM file for defining the elements of a JAVA/J2EE project. In regards to JAVA/J2EE applications the ARCHDEF represents how the J2EE application is structured in the Client IDE project workspace. The Project file structure of the application is replicated in the ARCHDEF (using the SCLM host short name to map the long name structure). Additional keywords in the ARCHDEF such as LINK, SINC, and OUT1 indicate to SCLM the J2EE nature of this project and source include a JAVA/J2EE build script to facilitate build processing of this project.
JAVALIST	This SCLM type is required for the Java build process. The JAVALIST type contains listing outputs from Java builds.
J2EELIST	This SCLM type is required for the J2EE build process. The J2EELIST type contains listing outputs from J2EE builds.
JAVACLAS	This SCLM type is required for both Java and J2EE build processes. The JAVACLAS type contains output class files from builds associated with the JAVA, J2EEANT language definitions.
J2EEJAR	This SCLM type is required for JAVA/J2EE builds (language definition J2EEANT). The J2EEJAR type contains JAR output from builds associated with the J2EEANT language definition.
J2EEWAR	This SCLM type is required for the J2EE build process.

The J2EEWAR type contains WAR output from builds associated with the J2EEANT language definition.

J2EEEAR This SCLM type is required for the J2EE build process.

The J2EEEAR type contains EAR output from builds associated with the J2EEANT language definition.

<Java/J2EE> types

A separate SCLM type is required for each JAVA/J2EE project to be stored in SCLM. This is to avoid conflicts in same-named files that occur with JAVA/J2EE projects. For more information about this, see “Mapping J2EE projects to SCLM” on page 54.

SCLM member formats

This section describes SCLM member formats:

- “\$GLOBAL”
- “J2EE ARCHDEF” on page 46
- “J2EE Ant build script” on page 49

\$GLOBAL

The \$GLOBAL format is of type J2EEBLD and language J2EEPART. It must use the name \$GLOBAL and variables are defined in tagged language format.

\$GLOBAL specifies the default properties for the SCLM project for JAVA/J2EE build processing. This must be stored in the SCLM type J2EEBLD.

For a detailed description of the \$GLOBAL member parameters, see “\$GLOBAL member” on page 62.

\$GLOBAL sample

\$GLOBAL sample:

```
<property name="ANT_BIN" value="/usr/lpp/Ant/apache-Ant-1.6.0/bin/Ant"/>
<property name="JAVA_BIN" value="/usr/lpp/java/IBM/J1.4/bin"/>
<property name="CGI_DTCONF" value="/etc/SCLMDT/CONFIG"/>
<property name="CGI_DTWORK" value="/var/SCLMDT/WORKAREA"/>
<property name="CLASSPATH_JARS" value="/var/SCLMDT/CLASSPATH"/>
```

J2EE ARCHDEF

The J2EE ARCHDEF format is of type ARCHDEF and language ARCHDEF.

LKED	J2EEOBJ	
INCLD	<i>SourceFile</i>	<i>SourceType</i>
INCL	<i>ArchdefName</i>	<i>ArchdefType</i>
SINC	<i>BuildScriptname</i>	J2EEBLD
OUT1	*	<i>J2EEOutputObjectType</i>
LIST	*	J2EELIST

The ARCHDEF uses standard SCLM architecture keywords to tell SCLM how to process the build of the ARCHDEF.

LKED Indicates this is a LEC ARCHDEF and gives the language of the ARCHDEF translator to be invoked (for J2EE ARCHDEFs, this is always J2EEOBJ).

INCLD

SCLM include of J2EE component. *SourceFile* is the name of the source member (for example, Java source) that is included in this ARCHDEF. *SourceType* is the SCLM type that contains the member. In an SCLM Developer Toolkit generated ARCHDEF there will be a comment that gives the full file name of the file as it existed in the project on the workbench.

INCL SCLM include of another nested ARCHDEF, such as the ARCHDEF that contains the manifest for an EJB application.

SINC Source include of the J2EEBLD build script. *BuildScriptName* is the name of the build script. The source type is always J2EEBLD.

OUT1 Indicates the J2EE object type created by this ARCHDEF. The member name is always *. The *J2EEOutputObjectType* is set to either J2EEEAR, J2EEWAR, or J2EEJAR. The member created will be given a name of the generated short name for the JAR, WAR, or EAR file.

LIST Summary component listing and audit of the ARCHDEF built. The member name is always *. The source type is always J2EELIST. The member will be given a name of the same value as the ARCHDEF member name.

J2EE ARCHDEF samples: This section shows JAR, WAR, EJB, and EAR samples.

```

*
* Initially generated on 10/05/2006 by SCLM DT V2
*
LKED  J2EE0BJ          * J2EE Build translator
*
* Source to include in build
*
INCLD AN000002 V2TEST  * com/Angelina.java          *
INCLD V2000002 V2TEST  * com/V2Java1.java          *
INCLD V2000003 V2TEST  * V2InnerClass.java          *
*
* Build script and generated outputs
*
SINC  V2JARB1 J2EEBLD  * J2EE JAR Build script      *
OUT1  *       J2EEJAR  * V2TEST.jar                *
LIST  *       J2EELIST

```

Figure 17. Sample Jar application (JAR) ARCHDEF

```

*
* Initially generated on 5 Sep 2006 by SCLM DT V2
*
LKED  J2EE0BJ          * J2EE Build translator
*
* Source to include in build
*
INCLD DA000026 SAMPLE5 * JavaSource/service/dateController.java *
INCLD XX000001 SAMPLE5 * .classpath                      *
INCLD XX000002 SAMPLE5 * .project                        *
INCLD XX000003 SAMPLE5 * .websettings                     *
INCLD XX000004 SAMPLE5 * .website-config                  *
INCLD OP000002 SAMPLE5 * WebContent/operations.html      *
INCLD MA000001 SAMPLE5 * WebContent/META-INF/MANIFEST.MF  *
INCLD IB000001 SAMPLE5 * WebContent/WEB-INF/ibm-web-bnd.xmi *
INCLD IB000002 SAMPLE5 * WebContent/WEB-INF/ibm-web-ext.xmi *
INCLD WE000001 SAMPLE5 * WebContent/WEB-INF/web.xml       *
INCLD MA000002 SAMPLE5 * WebContent/theme/Master.css      *
INCLD BL000001 SAMPLE5 * WebContent/theme/blue.css        *
INCLD BL000002 SAMPLE5 * WebContent/theme/blue.html      *
INCLD LO000013 SAMPLE5 * WebContent/theme/logo_blue.gif  *
*
* Build script and generated outputs
*
SINC  SAMPLE5 J2EEBLD  * J2EE WAR Build script      *
OUT1  *       J2EEWAR  * Sample5.war                *
LIST  *       J2EELIST

```

Figure 18. Sample Web application (WAR) ARCHDEF

```

LKED  J2EE0BJ
*
INCLD XX000001 SAMPLE3 * .classpath *
INCLD XX000002 SAMPLE3 * .project *
INCLD MA000004 SAMPLE3 * ejbModule/META-INF/MANIFEST.MF *
INCLD EJ000004 SAMPLE3 * ejbModule/META-INF/ejb-jar.xml *
INCLD IB000003 SAMPLE3 * ejbModule/META-INF/ibm-ejb-jar-bnd.xmi *
INCLD XX000008 SAMPLE3 * ejbModule/com/ibm/ejs/container/_EJSWrapper *
* _Stub.java *
INCLD XX000009 SAMPLE3 * ejbModule/com/ibm/ejs/container/_EJSWrapper *
* _Tie.java *
INCLD XX000010 SAMPLE3 * ejbModule/com/ibm/websphere/csi/_CSIServant *
* _Stub.java *
INCLD XX000011 SAMPLE3 * ejbModule/com/ibm/websphere/csi/_Transactio *
* nalObject_Stub.java *
INCLD DA000005 SAMPLE3 * ejbModule/myEJB/DateBean.java *
INCLD DA000006 SAMPLE3 * ejbModule/myEJB/DateBeanBean.java *
INCLD DA000007 SAMPLE3 * ejbModule/myEJB/DateBeanHome.java *
INCLD EJ000001 SAMPLE3 * ejbModule/myEJB/EJSRemoteStatelessDateBeanH *
* ome_1a4c4c85.java *
INCLD EJ000002 SAMPLE3 * ejbModule/myEJB/EJSRemoteStatelessDateBean_ *
* _1a4c4c85.java *
INCLD EJ000003 SAMPLE3 * ejbModule/myEJB/EJSStatelessDateBeanHomeBea *
* nHomeBean_1a4c4c85.java *
INCLD XX000012 SAMPLE3 * ejbModule/myEJB/_DateBeanHome_Stub.java *
INCLD XX000013 SAMPLE3 * ejbModule/myEJB/_DateBean_Stub.java *
INCLD XX000014 SAMPLE3 * ejbModule/myEJB/_EJSRemoteStatelessDateBean *
* Home_1a4c4c85_Tie.java *
INCLD XX000015 SAMPLE3 * ejbModule/myEJB/_EJSRemoteStatelessDateBean *
* _1a4c4c85_Tie.java *
INCLD XX000016 SAMPLE3 * ejbModule/org/omg/stub/javax/ejb/_EJBHome_S *
* ub.java *
INCLD XX000017 SAMPLE3 * ejbModule/org/omg/stub/javax/ejb/_EJBObject *
* _Stub.java *
INCLD XX000018 SAMPLE3 * ejbModule/org/omg/stub/javax/ejb/_Handle_St *
* ub.java *
INCLD XX000019 SAMPLE3 * ejbModule/org/omg/stub/javax/ejb/_HomeHandl *
* e_Stub.java *
INCLD DA000008 SAMPLE3 * ejbModule/services/DateBeanServices.java *
INCLD XX000020 SAMPLE3 * ejbModule/services/_DateBeanServices_Stub.j *
* ava *
*
SINC  SAMPLE3  J2EEBLD * J2EE EJB JAR Build script *
OUT1  *        J2EEJAR * DateService.jar *
*
LIST  *        J2EELIST

```

Figure 19. Sample EJB Application (EJB) ARCHDEF

```

LKED    J2EEOBJ
*
INCLD   XX000001 SAMPLE6 * .classpath *
INCLD   XX000002 SAMPLE6 * .project *
INCLD   AP000001 SAMPLE6 * META-INF/application.xml *
INCL    SAMPLE3  ARCHDEF * DateService.jar *
INCL    SAMPLE5  ARCHDEF * Sample5.war *
*
SINC    SAMPLE6  J2EEBLD * J2EE EAR Build script *
OUT1    *        J2EEEAR * Sample6.ear *
LIST    *        J2EELIST

```

Figure 20. Sample EAR Application (EAR) ARCHDEF

J2EE Ant build script

The J2EE Ant build Script format is of type J2EEBLD and language J2EEANT. It can be any name up to 8 characters and variables are defined in tagged language format. The build scripts are very similar for JAR, WAR and EAR. The syntax below is shown for a WAR build script. For JAR and EAR, build script variables are the same except for using EAR_NAME and JAR_NAME instead of WAR_NAME.

```

<ANTXML>
<project name="J2EE Project type" default="web-war" basedir=".">
<property name="env" environment="env" value="env"/>
<property name="SCLM_ARCHDEF" value="ARCHDEF name"/>
<property name="SCLM_ANTXML" value="ANTXML name"/>
<property name="SCLM_BLDMAP" value="Include Buildmap"/>
<property name="JAVA_SOURCE" value="Include Java Source"/>
<property name="J2EE_HOME" value="{env.J2EE_HOME}"/>
<property name="JAVA_HOME" value="{env.JAVA_HOME}"/>
<property name="CLASSPATH_JARS" value="Classpath Directory location"/>
<property name="CLASSPATH_JARS_FILES" value="Jar/class filenames"/>
<property name="ENCODING" value="Codepage"/>
<property name="DEBUG_MODE" value="debug_mode"/>

<!-- WAR file name to be created by this build process -->
<!-- include suffix of .war -->
<property name="WAR_NAME" value="War name" />

<path id="build.class.path">
<pathelement location="."/>
<pathelement location="{J2EE_HOME}/lib/j2ee.jar"/>
<pathelement location="{CLASSPATH_JARS}/jdom.jar"/>
<fileset dir="." includes="**/*.jar"/>
<fileset dir="{CLASSPATH_JARS}" includes="**/*.jar, **/*.zip"/>
</path>

</ANTXML>

```

Figure 21. J2EE Ant build script

The SCLM Build scripts overlay customer-defined variables dynamically on build request when running the Ant build script. These variables are set to values shown in Table 8 on page 50.

Table 8. Customer-defined variables

Variable	Description
J2EE Project name	Java/J2EE project type being built. This is a temporary project name set in the build script for Ant to use during the build. The values this will be set to are: <ul style="list-style-type: none"> • J2EE EAR Project • J2EE WAR Project • J2EE EJB Project • JAVA Project This variable does not need to be customized.
SCLM_ARCHDEF	ARCHDEF name or the ARCHDEF being built
SCLM_ANTXML	Name of skeleton Ant XML to use for build
SCLM_BLDMAP	Value of Yes or No. If Yes then include the SCLM build map in MANIFEST directory in JAR, WAR, or EAR. Provides audit and build map of parts included.
JAVA_SOURCE	Value of Yes or No. If Yes then include Java source in JAR, WAR, or EAR.
CLASSPATH_JARS	z/OS UNIX System Services classpath directory used for resolving classpath dependencies during build. All jars located in this directory will be used in the classpath.
CLASSPATH_JARS_FILES	Names of individual JAR and Class files to be included in the build. This can be in the form of a list: <pre><property name="CLASSPATH_JARS_FILES" value="V2J4.jar,V2J3.jar" /></pre>
ENCODING	Either ASCII or EBCDIC codepage for JAVA This is the codepage JAVA source is stored on the z/OS host. For example: <ul style="list-style-type: none"> • For ASCII JAVA standard codepage should be ISO8859-1 • For EBCDIC JAVA standard codepage should be IBM-1047
JAR_FILE_NAME EJB_NAME WAR_NAME EAR_NAME	Name of JAR, EJB JAR, WAR, or EAR.
DEBUG_MODE	Set to 'on' to force Developer Toolkit to not remove any build files from the WORKAREA directory. This is useful if you need to check the structure of a built Java/J2EE application.

CLASSPATH dependencies: Java source within an ARCHDEF can have classpath dependencies upon other Java libraries or classes. If these dependencies are on Java components contained within the same ARCHDEF structure, then these classpath dependencies are resolved as part of the ARCHDEF build (whether build mode is conditional or forced).

However, a J2EE ARCHDEF component might have classpath dependencies on external JARs or even on members contained in other ARCHDEFs. In this case the J2EE build script associated with the ARCHDEF can control classpath dependencies with the following keywords:

CLASSPATH_JARS

This is a directory name in the z/OS UNIX System Services file system which might include all external dependent JAR files and classes for that

particular ARCHDEF build.

CLASSPATH dependencies

This directory can be updated with CLASSPATH files via the Client Team function 'Upload jar files' to copy JAR files from the client into the classpath directory. Also available is the function 'Copy file from SCLM to classpath' to copy SCLM stored JAR files into the classpath directory.

CLASSPATH_JARS_FILES

This keyword can be used to selectively choose individual JAR or class files to be used in the classpath. If this keyword is used, then the listed JAR/class files are retrieved from SCLM or if not located in SCLM then the directory referenced by CLASSPATH_JARS is searched for retrieval. If this keyword is used then only those files listed are used in the classpath.

J2EE sample scripts: The following samples show JAR, WAR, EJB, and EAR scripts.

```
<ANTXML>
<project name="JAVA Project" default="jar" basedir=".">
<property name="env" environment="env" value="env"/>
<property name="SCLM_ARCHDEF" value="V2JAR1"/>
<property name="SCLM_ANTXML" value="BWBJAVA"/>
<property name="SCLM_BLDMAP" value="YES"/>
<property name="JAR_FILE_NAME" value="V2TEST.jar"/>
<property name="CLASSPATH_JARS" value="/var/SCLMDT/CLASSPATH"/>
<property name="ENCODING" value="IBM-1047"/>
</ANTXML>
```

Figure 22. J2EE Build script JAR sample

```
<ANTXML>
<project name="J2EE WAR Project" default="web-war" basedir=".">
<property name="env" environment="env" value="env"/>
<property name="SCLM_ARCHDEF" value="SAMPLE5"/>
<property name="SCLM_ANTXML" value="BWBWEB"/>
<property name="SCLM_BLDMAP" value="YES"/>
<property name="JAVA_SOURCE" value="YES"/>
<property name="J2EE_HOME" value="${env.J2EE_HOME}"/>
<property name="JAVA_HOME" value="${env.JAVA_HOME}"/>
<property name="CLASSPATH_JARS" value="/var/SCLMDT/CLASSPATH"/>
<property name="ENCODING" value="IBM-1047"/>
<!-- WAR file name to be created by this build process -->
<property name="WAR_NAME" value="Sample5.war" />
<path id="build.class.path">
  <pathelement location="."/>
  <pathelement location="${J2EE_HOME}/lib/j2ee.jar"/>
  <pathelement location="${CLASSPATH_JARS}/jdom.jar"/>
</path>
<fileset dir="${CLASSPATH_JARS}" includes="**/*.jar, **/*.zip"/>
</ANTXML>
```

Figure 23. J2EE Build script WAR sample

```

<ANTXML>
<project name="J2EE EJB Project" default="EJBBuild" basedir=".">
<property name="env" environment="env" value="env"/>
<property name="SCLM_ARCHDEF" value="SAMPLE3"/>
<property name="SCLM_ANTXML" value="BWBEJBA"/>
<property name="SCLM_BLDMAP" value="NO"/>
<property name="J2EE_HOME" value="${env.J2EE_HOME}"/>
<property name="JAVA_HOME" value="${env.JAVA_HOME}"/>
<property name="CLASSPATH_JARS" value="/var/SCLMDT/CLASSPATH"/>
<property name="ENCODING" value="IBM-1047"/>
<property name="EJB_NAME" value="DateService.jar"/>
<path id="build.class.path">
  <pathelement location="."/>
  <pathelement location="${J2EE_HOME}/lib/j2ee.jar"/>
  <pathelement location="${CLASSPATH_JARS}/jdom.jar"/>
  <fileset dir="${CLASSPATH_JARS}" includes="**/*.jar, **/*.zip"/>
</path>
</ANTXML>

```

Figure 24. J2EE Build script EJB sample

```

<ANTXML>
<project name="J2EE EAR Project" default="j2ee-ear" basedir=".">
<property name="env" environment="env" value="env"/>
<property name="SCLM_ARCHDEF" value="SAMPLE6"/>
<property name="EAR_NAME" value="Sample6.ear"/>
<property name="SCLM_ANTXML" value="BWBEARA"/>
<property name="SCLM_BLDMAP" value="NO"/>
<property name="J2EE_HOME" value="${env.J2EE_HOME}"/>
<property name="JAVA_HOME" value="${env.JAVA_HOME}"/>
<property name="CLASSPATH_JARS" value="/var/SCLMDT/CLASSPATH"/>
<path id="build.class.path">
  <pathelement location="."/>
  <pathelement location="${J2EE_HOME}/lib/j2ee.jar"/>
  <pathelement location="${CLASSPATH_JARS}/jdom.jar"/>
  <fileset dir="${CLASSPATH_JARS}" includes="**/*.jar, **/*.zip"/>
</path>
<target name="common">
<echo message="BuildName: ${Ant.project.name}" />
<echo message="BuildHome: ${basedir}" />
<echo message="BuildFile: ${Ant.file}" />
<echo message="BuildJVM: ${Ant.java.version}" />
</target>
</ANTXML>

```

Figure 25. J2EE Build script EAR sample

JAVA/J2EE Ant XML build skeletons

This section lists sample Ant build skeletons which are provided in the SBWBSAMP library. These sample members can be copied into SCLM type J2EEBLD in the SCLM hierarchy to be referenced and used by the JAVA/J2EE build scripts. The JAVA/J2EE build scripts are property variable files that overlay the Ant XML skeleton files.

JAVA/J2EE Ant XML build skeletons

The supplied sample J2EE build skeletons for building a simple JAR, SQLJ project, EJB JAR, WAR, EAR or for deployment can generally be used, as is, without user customization. Be aware however that some J2EE projects might not fit the standard model and some customization of the supplied Ant XML skeletons may be required.

Note: JAVA/J2EE build scripts can be generated via the Developer toolkit client application. These build scripts use a referenced Ant XML skeleton (as below) and an ARCHDEF in the JAVA/J2EE build process.

A detailed description of build scripts, Ant skeletons and examples on JAVA/J2EE build processing is contained in the SCLM Developer Toolkit User Guide supplied with the client plug-in.

BWBJAVA Sample Ant XML JAVA build skeleton

This Ant skeleton is used by a Java build script to compile multiple java programs and optionally create a Java Archive (JAR) file which has a structure determined by a specified ARCHDEF.

BWBEJBA Sample Ant XML J2EE EJB build skeleton

This Ant skeleton is used by a J2EE build script to compile/build an EJB project which would usually create an EJB JAR which has a structure determined by a specified ARCHDEF.

BWBWEB Sample Ant XML J2EE WEB build skeleton

This Ant skeleton is used by a J2EE build script to compile/build a WEB project which would usually create a WEB Archive (WAR) file.

BWBEARA Sample Ant XML J2EE EAR assemble skeleton

This Ant skeleton is used by a J2EE build script as an assemble process in preparation for J2EE application deployment. The process produces Enterprise Archive (EAR) files which can be deployed on to a Web application server such as WebSphere application server.

BWBSQLB Sample Java/SQLJ build script.

This Ant Skeleton is used by a J2EE build script to compile/build a JAR project that uses SQLJ.

BWBSQLBE Sample EJB/SQLJ build script.

This Ant Skeleton is used by a J2EE build script to compile/build an EJB project that uses SQLJ.

Mapping J2EE projects to SCLM

IBM SCLM Developer Toolkit provides the capacity to manage, build, and deploy projects in SCLM. This section describes how to configure the SCLM project structure to support distributed application development such as JAVA/J2EE.

Many JAVA/J2EE projects result in the creation of an executable EAR file. This application is an assembly of projects, typically EJBs and Web applications and, within the IDE environments, these are generally developed as individual project structures that are linked to an EAR project.

This form of multiple-project structure does not map to SCLM directly. That is, an SCLM project cannot be linked to another SCLM project to provide some form of aggregated project structure. However, SCLM does provide a means to support this multiple project structure within a single SCLM project using types.

Mapping J2EE projects to SCLM

SCLM projects can be defined with multiple source types. Each type can hold a single IDE project. If we tried to store multiple Eclipse IDE projects in SCLM without some form of segregation then each of the project's .classpath and .project files would be overwritten as each project was added to SCLM. The use of different source types enables these files, and all others associated with that project, to be stored safely within SCLM.

Figure 26 shows how multiple types make it possible to support a multiple-project structure within a single SCLM project.

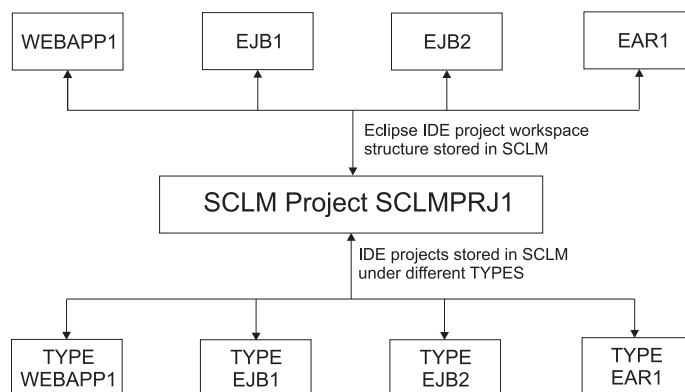


Figure 26. Multiple types

This mapping would result in the IDE projects being stored independently within SCLM using the type as the principal differentiator. For example, EJB1 is stored in the SCLM project SCLMPRJ1 under type EJB1. Using this structure, it is possible to map the IDE project structure to independent types within the SCLM project.

Notes:

1. It is not necessary to map a project name in the IDE to the SCLM type name; these names exist independently of each other.
2. Type names are restricted to eight characters; therefore an IDE project called 'ProjectOne' could not have the corresponding type name of 'ProjectOne'. You can use 'Proj1' instead.

It is therefore important that the SCLM project structure be planned to accommodate the mapping of different IDE-based projects into the single SCLM project structure. This is because, within large SCLM projects, it might be a non-trivial matter to add additional project types as this requires a change to the SCLM project definition, a rebuild of the SCLM project definition, and the allocation of data sets for the new types.

This structure is not restricted to J2EE-style projects but could also apply to any situation where multiple projects are being developed that provide some form of dependency upon each other.

Recommendations for mapping J2EE projects to SCLM

The following list gives recommendations for mapping J2EE projects (and others) to SCLM:

- Identify the J2EE project composition in terms of EJBs, Web applications, and so on, so that this can be used to plan the SCLM project structure.
- For each of the J2EE IDE project components, create a corresponding type in the SCLM project. It is useful to provide some form of meaningful naming convention to support this. Whilst it is possible to name the IDE projects independently of the SCLM type, some correlation will make administration easier.
- As project requirements can change, create additional type definitions to enable the smooth addition of other components such as additional EJBs. Additional services can be anticipated through the type structure.
- Mapping multiple IDE projects into a single SCLM project is supported by the type construct. It is also useful to apply some packaging structure that takes into account the type definition for that project.
- Java-style packaging conventions can also be defined at the project level so as to avoid the likelihood of source naming collisions.
- If the IDE is structured with multiple projects it is advisable to replicate this structure in SCLM using type.

The use of multiple SCLM types to store individual IDE projects also relates to the operation of the ARCHDEF structure for the building of these IDE projects.

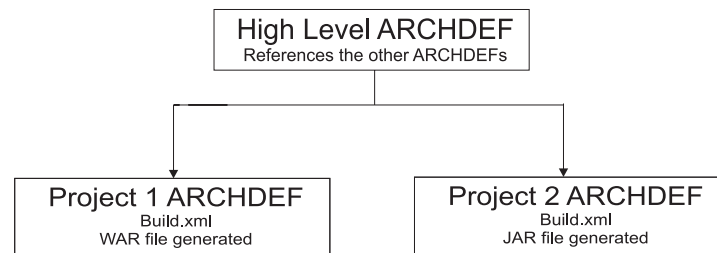


Figure 27. SCLM build hierarchy

The ARCHDEF file contains the list of files that make up a build. In a J2EE context a build can result in an EAR file being composed of a number of WAR and JAR files. This isolation of projects is similar to the type structure that defines the project in SCLM. By having a high-level ARCHDEF that references those ‘parts’ that make up the build, it is possible to have a structured build environment. This relates to the effective definition of project structure when defining the types in SCLM.

By defining the project in a structured manner this also enables:

- Migration of files from an SCLM project type or ARCHDEF to an IDE project without the need to know individual parts.
- The ARCHDEF structure based on type definition also enables project dependencies to be mapped more effectively. It is common for IDE projects to reference other IDE projects in the workspace. Use of the SCLM INCL keyword in ARCHDEFs supports this notion as other IDE projects, referenced by other ARCHDEFs, can be included by nesting the ARCHDEFs within higher level ARCHDEFs.

Recommendations for mapping J2EE projects (and others) to SCLM

When building applications with references or dependencies on other build objects such as JARs, other projects or other classes there are multiple approaches:

1. Include the reference to the JAR via the INCLD statement in the ARCHDEF. This will build the application with the library reference into the final build package.
2. Include the IDE project as a nested INCL SCLM project ARCHDEF
3. Include the dependent JAR in the CLASSPATH directory
 - a. Should the IDE project refer to a JAR but it is not expected to be part of the final build package then the library file can be copied to the system CLASSPATH using the Upload JARS service in Developer Toolkit. This will have the effect of making that service available from a different SCLM project. At build time the IDE project references to the JAR will be resolved. However at runtime, the JAR must be available in a PATH statement.
 - b. Reference a JAR that is in the same SCLM project through the use of the CLASSPATH_JARS_FILES property in the build script.

SCLM Developer Toolkit deployment

SCLM Developer Toolkit provides several deployment features. You can deploy Enterprise Archive files (EAR) into any WebSphere Application Server (WAS). In addition, any component built or controlled by the SCLM Developer Toolkit can be distributed using a customizable deploy script. Sample scripts are provided that can be used to copy an EAR to a remote host using the secure copy (SCP) and secure FTP (SFTP) commands.

At the core of deployment there are essentially two scripts. The first type of script, the one that is modified by the user, is the properties script. It contains a list of parameters for the deployment operation. The second is the action script that contains the steps required to run the deployment operation.

Deployment is initiated from the SCLM Developer Toolkit client plug-in and the type of deployment is chosen by pressing the relevant button on the deployment screen. Depending on what deployment action is chosen will have an effect on what is populated in the properties script. For most of the scripts there is a property named SCLM_ANTXML that contains the member name of the corresponding action script. Developer Toolkit takes the generated properties script and overlays it on the action script, before invoking the resultant action script.

Below is a list of sample Ant deployment action scripts which are provided in the SBWBSAMP library. These sample members can be copied into SCLM type J2EEBLD in the SCLM hierarchy to be referenced and used by the generated properties scripts. The generated properties scripts are property variable files that overlay the Ant XML deployment action scripts referenced below. These scripts must be stored with a text type language such as TEXT or J2EEPART.

Member Name	Description
BWBDEPLA	WAS EAR Deployment.
BWBRDEPL	Remote WAS EAR Deployment.
BWBSCOPY	Secure copy deployment. Copies a build object from one host to another using SCP.

BWBSFTP

Secure FTP deployment. Copies a build object from one host to another using SFTP.

In order for these build scripts to be usable from multiple groups, the administrator must build and promote the scripts to the highest group level available in the project.

There is slightly different processing depending on the types of scripts being generated.

WebSphere Application Server (WAS) deployment

For WebSphere Application Server (WAS) deployment the SCLM_ANTXML property does not point to an Ant action script, but references a JACL action script instead. Alternatively, you can use the wsadmin tool that is shipped with WAS on z/OS. The wsadmin tool requires a JACL script to guide the deployment process. If using this deploy method then the JACL script must be installed under UNIX Systems Services before the deployment process can be invoked. The JACL script must be copied into a z/OS UNIX System Services file system directory on z/OS. It is recommended to store the JACL script as `/etc/SCLMDT/CONFIG/scripts/deploy.jacl` (where `/etc/SCLMDT` is the default etc directory for SCLM Developer Toolkit). The wsadmin tool expects the JACL script to be in ASCII format.

To copy the sample JACL script BWBJACL (that is stored in the SBWBSAMP sample library) into a z/OS UNIX System Services file system directory in ASCII format, follow these steps:

- Review the instructions contained in the sample job BWBJACLJ and customize accordingly. BWBJACLJ can be found in the SCLM Developer Toolkit SBWBSAMP library. Additional JACL examples can be found in the WebSphere Application Server (WAS) documentation.
- Submit the job BWBJACLJ. This will convert into ASCII sample BWBJACL and copy into the customized directory as file name `deploy.jacl`.

The client plug-in must be able to communicate with the directory locations where the wsadmin tool (`wsadmin.sh`) and the JACL script are installed under Unix System Services. Both locations can be configured in the preference page under **Team -> SCLM Preferences -> Build Script Options**. The client plug-in is used to generate a deployment script which can then be built against. (The deployment process is triggered by a deploy function request against the deployment script which is stored in SCLM type J2EEBLD).

The sample action scripts that need to be stored in SCLM type J2EEBLD for WAS deployment or remote WAS deployment are BWBDEPLA and BWBRDEPL.

SCLM to Unix System Services deployment

SCLM Developer Toolkit provides a means to deploy any files that are stored in the SCLM repository to the z/OS UNIX System Services File System on the same LPAR. This provides a simple means to deploy an object built by SCLM into an environment where it can be either executed or even deployed to a remote host using the Secure Deployment described below.

SCLM to Unix System Services deployment

There is no sample action script for this action. Select the members from SCLM and use the **Include SCLM members** button to generate the required properties script. This copies the files from the selected SCLM location to a directory specified on the z/OS UNIX System Services File System. This directory must previously exist or an error will occur.

Secure deployment

This options provide a means to copy deployable objects to a remote host by utilizing the secure copy (SCP) and secure FTP (SFTP) commands. By using a combination of the Secure deploy properties script and the Include SCLM members, the required files can be selected from the SCLM hierarchy, copied to a location in the z/OS UNIX System Services File System, and then copied to the destination machine from that z/OS UNIX System Services File System location utilizing the secure copy (SCP) and secure FTP (SFTP) commands.

The sample action scripts that need to be stored in SCLM type J2EEBLD for secure deployment are BWBSCOPY and BWBSFTP.

In order for Developer Toolkit to use SFTP (Secure file transfer protocol) or SCP (Secure copy protocol) within ANT deploy scripts the IBM Ported Tools for z/OS (For z/OS V1.4 and above - Program number 5655-M23) product is required to be installed as a pre-requisite.

IBM Ported Tools for z/OS is a non-priced program product designed to deliver tools and applications for the z/OS platform. These applications have been modified to operate within the z/OS environment. IBM Ported Tools for z/OS is only available if you have a license to z/OS; it is supported on z/OS 1.4 and above.

It provides:

- scp for copying files between networks. It is an alternative to rcp.
- sftp for file transfers over an encrypted ssh transport. It is an interactive file transfer program similar to ftp.

Public key authentication

Public key authentication provides an alternative to interactive logon that can be automated as part of Developer Toolkit's secure deployment operation.

In order for public key authentication to work as desired, you can either use a surrogate User ID for deployment or configure each user for whom you wish to provide deployment capabilities.

For instructions on how to set up automated key-based authentication using ssh-agent and ssh-add, see the IBM Ported Tools for z/OS User's Guide. For information about using SCLM Developer Toolkit surrogate user ID, see Chapter 4, "SCLM security," on page 73.

Other deployment options

It is also possible to create your own Ant scripts to perform deployment in a number of different ways. In your scripts, by using the Ant <exec> tag you can invoke any program that is available in the z/OS UNIX System Services File System. Using this method the build scripts can call other programs, such as FTP, to perform deployment. For more information of creating Ant scripts see the online Ant documentation at <http://ant.apache.org/>.

ASCII or EBCDIC storage options

Source files transferred from the SCLM Developer Toolkit plug-in can be stored in SCLM as either ASCII or EBCDIC.

Generally all source in SCLM is stored in EBCDIC to be viewed and edited directly from ISPF/SCLM on z/OS. If you do not want to browse or edit code directly from the host, you might want to store code directly (that is, as binary transferred) where source will be stored in SCLM using the original client's ASCII/UNICODE codepage. This does have some performance benefits for large projects being stored and imported from SCLM and for JAVA/J2EE builds as an ASCII to EBCDIC translation will not be performed.

SCLM Developer Toolkit determines if a file is binary transferred or if an ASCII to EBCDIC conversion takes place by checking the SCLM language associated with each file/member. Then SCLM Developer Toolkit checks to see if that SCLM Language has an entry in the TRANSLATE.conf file with a TRANLANG keyword.

ASCII/EBCDIC language translators

SCLM Language Translator	Description
JAVA	Java source members stored as EBCDIC. Created by using sample BWBTRAN1.
SQLJ	SQLJ members stored as EBCDIC. Created by using sample (BWBTRANS)
JAVABIN	Java source members stored as ASCII. Created by using sample BWBTRAN1.
J2EEPART	Any J2EE files where no parsing is required and stored as EBCDIC. Created by using sample BWBTRAN1.
J2EEBIN	Any J2EE files where no parsing is required and stored as binary or ASCII files. Created by using sample BWBTRAN1.
SQLJ	SQLJ source members stored as EBCDIC. Created by using sample BWBTRANS.
SQLJBIN	SQLJ source members stored as ASCII. Created by using sample BWBTRANS.
TEXT	Default TEXT translator where no parsing is required and stored as EBCDIC. Created by using sample BWBTRAN1.
BINARY	Default binary language translator where no parsing required. Created by using sample BWBTRAN1.

Figure 28. SCLM Language Translators and ASCII/EBCDIC

Default usage is assumed to be ASCII/EBCDIC translation. This means that files browsed and edited in the Eclipse Plug-in can also be browsed and edited directly on host from ISPF/SCLM.

ASCII usage (binary transferred) is recommended for project migration/import and build performance, as files require no translation. This is only suitable if editing in ISPF/SCLM is not required.

ASCII/EBCDIC language translators

Depending on the SCLM Language Translator used, source can be built in either ASCII or EBCDIC.

For cross platform usability, all deployable files such as JAR, WAR and EAR are built such that all of the contained objects are of type ASCII, regardless of whether any of the source is stored as EBCDIC.

JAVA/J2EE build note: If Java source is ASCII stored then the Build script must specify the ASCII codepage using the ENCODING property variable to correctly compile the Java source.

For example:

```
<property name="ENCODING" value="ISO8859-1"/>
```

The Ant script called will use the Javac command with the ENCODING=ISO8859-1 to compile the ASCII source. The default ENCODING codepage is the EBCDIC codepage IBM-1047.

\$GLOBAL member

As part of the JAVA/J2EE build process some additional information is required in order to successfully perform the builds. As the builds are performed in z/OS UNIX System Services, information such as the Java product location, Ant product location and the location of the SCLM Developer Toolkit configuration files and workarea are required.

Additionally it might be required to use different versions of Ant or Java for different SCLM development groups, so to this end the \$GLOBAL member can be group specific. The environment variables set in \$GLOBAL can be overwritten by specific build script variable settings.

Note: The JAVA_HOME specified in ANT customization will override any JAVA_BIN specified in \$GLOBAL for Java/J2EE project compiles.

A sample member BWBGL0B is provided in the SBWBSAMP library. This sample member needs to be copied into SCLM type J2EEBLD in the SCLM hierarchy as member \$GLOBAL and saved with a valid non-parsing language, such as TEXT (as provided in language translator FLM@TEXT in the SISPMACS library).

The \$GLOBAL member currently makes available the following information to the JAVA/J2EE build processes:

Table 9. \$GLOBAL variables

Variable	Description
ANT_BIN	z/OS UNIX System Services file system directory path of Ant runtime Example: <property name="ANT_BIN" value="/u/antdirectory/Ant/apache-Ant-1.6.0/bin/Ant"/>
JAVA_BIN	z/OS UNIX System Services file system directory path of Java compile/runtime Example: <property name="JAVA_BIN" value="/u/javadirectory/IBM/J1.4/bin"/>

Table 9. \$GLOBAL variables (continued)

Variable	Description
CGI_DTWORK	The location of the SCLM Developer Toolkit WORKAREA directory Example: <code><property name="CGI_DTWORK" value="/var/SCLMDT"/></code>
CGI_DTCONF	The location of the SCLM Developer Toolkit CONFIG directory Example: <code><property name="CGI_DTCONF" value="/etc/SCLMDT"/></code>
CLASSPATH_JARS	z/OS UNIX System Services file system classpath directory used for JAVA compiles. All jars located in this directory will be used in the classpath. Example: <code><property name="CLASSPATH_JARS" value="/var/SCLMDT/CLASSPATH"/></code>
TRANTABLE	VSAM file containing the long/short name translations Example: <code><property name="TRANTABLE" value="BWB.LSTRANS.FILE"/></code>
DEBUG_MODE	Set to "on" if you want Developer Toolkit to not remove Java/J2EE build files from the z/OS UNIX System Services file system. This is useful if you want to see the structure of the built outputs in the USS file system for debugging purposes.

If these variables are to be set for the all group levels in the SCLM project then it is good practice to create a single \$GLOBAL member at the highest level in the hierarchy. When the JAVA/J2EE build translator runs it will look up the hierarchy from the group level performing the build and use the first \$GLOBAL it finds in the J2EEBLD type.

Note: The \$GLOBAL member must be stored as a valid saved SCLM member so this hierarchy lookup can be performed.

If different settings are required, at different development groups for example, then a \$GLOBAL member can be created in each of the development groups.

SITE and project-specific options

A facility has been provided to allow certain settings to be made at a SITE installation level or at a specific SCLM project level. The options that can currently be configured are:

- Mandatory Change Code entry
- Deactivation of foreground Builds and Promotes
- Specification of package approval system. Currently IBM Breeze for SCLM is the supported approval system.
- Definition of Batch Build, Promote and Migrate job cards
- Override settings in the TRANSLATE.conf configuration file
- Project list filter restriction
- Define default settings for BiDirectional languages

All or none of these options can be set. If they are not set they will be defaulted in the programs. Some of these options can be set in the SITE file while others can be set at an SCLM project-specific level. Alternatively there can be no SITE specific file

SITE and project-specific options

and options can be set at an SCLM project level only. For job cards you can override the job card information by using your own specified job card entered through the IDE.

This facility is activated by creating certain files in the z/OS UNIX System Services file system under the `/etc/SCLMDT/CONFIG/PROJECT` directory, or wherever you created the `/CONFIG/PROJECT` directory at installation time. This directory is created at project initialization time by running job BWBINST1.

If you want to set SITE specific values then you need to create a file called `SITE.conf` in the `/PROJECT` directory. A sample SITE config file is provided in the `SBWBSAMP` library in member `BWBSITE`. Copy this member to a file named `SITE.conf` in this directory and tailor the values accordingly. The following figure shows the sample SITE configuration file.

SITE and project-specific options

```
*****
*
* SCLM Developer Toolkit Site Specific option
*
*****
*
* Below are a number of site specific options used to
* determine the behavior of the Eclipse front-end.
* These can be overridden by creating a project specific
* options file for the SCLM project that overrides some
* or all of these options.
*
* SCM Approver processing applies to this project?
BUILDAPPROVER=NONE
PROMOTEAPPROVER=NONE
*
* Change Code entry on check-in is mandatory?
CCODE=N
*
*
* To allow promotion by architecture definition only,
* set the value of PROMOTEONLYFROMARCHDEF to Y
PROMOTEONLYFROMARCHDEF=N
*
* Foreground or On-line builds/promotes allowed for this project?
FOREGROUNDBUILD=Y
FOREGROUNDPROMOTE=Y
*
* Batch Build default jobcard
BATCHBUILD1=//SCLMBILD JOB (#ACCT),'SCLM BUILD',CLASS=A,MSGCLASS=X,
BATCHBUILD2=//          NOTIFY=,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=//*
*
* Batch Migrate default jobcard
BATCHMIGRATE1=//SCLMMIGR JOB (#ACCT),'SCLM MIGRATE',CLASS=A,MSGCLASS=X,
BATCHMIGRATE2=//          NOTIFY=&SYSUID,REGION=128M
BATCHMIGRATE3=//*
BATCHMIGRATE4=//*
*
* Batch Deployment default jobcard
BATCHDEPLOY1=//SCLMDPLY JOB (#ACCT),'SCLM DEPLOY',CLASS=A,MSGCLASS=X,
BATCHDEPLOY2=//          NOTIFY=&SYSUID,REGION=128M
BATCHDEPLOY3=//*
BATCHDEPLOY4=//*
*
* BUILD Security flag for SAF/RACF security call and possible Surrogate ID switch
BUILDSECURITY=N
*
* Project list flag if set to N will stop users selecting * as project filter
* This may avoid long user catalog searches for all SCLM projects
*
PROJECTLISTALL=Y
```

Figure 29. Sample SITE specific SCLM project setting

It is also possible to have project-specific configuration settings that are used to configure a single SCLM project. These will override the SITE-specific values if a SITE.conf exists. If you want to set project-specific values then you need to create a file called project.conf in the /PROJECT directory, where project is the SCLM project name. A sample project config file is provided in the SBWBSAMP library in

member BWBPROJ. Copy this member to a file named project.conf in the /PROJECT directory and tailor the values accordingly. The following figure shows the sample Project configuration file.

```

*
* ----- PROJECT SPECIFIC OPTIONS -----
*
* Below are a number of project specific options used to
* determine the behavior 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,NOTIFY=&SYSUID,
BATCHBUILD2=// MSGLEVEL=(1,1)
BATCHBUILD3=//*
BATCHBUILD4=//*
*
* Batch Promote default jobcard
BATCHPROMOTE1=//SCLMPROM JOB (#ACCT),'SCLM PROMOTE',CLASS=A,MSGCLASS=X,
BATCHPROMOTE2=// MSGLEVEL=(1,1),NOTIFY=&SYSUID
BATCHPROMOTE3=//*
BATCHPROMOTE4=//*

```

Figure 30. Sample PROJECT specific SCLM project setting

Options Definition

All of the options are optional. They are set to the default values by the product. If any of the options are specified in the SITE.conf or the project.conf then they will be used.

Table 10. SITE/Project options

BUILDAPPROVER=approval product/ <u>NONE</u>	Specify the name of the approval product used for the build process. Currently the only supported product is Breeze for SCLM. Default is NONE.
PROMOTEAPPROVER=approval product/ <u>NONE</u>	Specify the name of the approval product used for the promote process. Currently the only supported product is Breeze for SCLM. If the PROMOTEAPPROVER is set to BREEZE then the Breeze specific fields will be displayed during a promote. Default is NONE.
CCODE= <u>N</u> /Y	Specify Y to make change code entry on check-in a mandatory field. Default is N such that Change Code entry is not mandatory.

Options Definition

Table 10. SITE/Project options (continued)

FOREGROUNDBUILD=Y/N	Specify N to restrict foreground builds. Default is Y such that foreground builds are allowed.
FOREGROUNDPROMOTE=Y/N	Specify N to restrict foreground promotes. Default is Y such that foreground promotes are allowed.
BATCHBUILD1=Job card 1 BATCHBUILD2=Job Card 2 BATCHBUILD3=Job Card 3 BATCHBUILD4=Job Card 4	Set a default batch job card for the build process. Different projects can use different account codes or Job class so the option of specifying project specific job cards allows for this scenario.
BATCHPROMOTE1=Job card 1 BATCHPROMOTE2=Job card 2 BATCHPROMOTE3=Job card 3 BATCHPROMOTE4=Job card 4	Set a default batch job for the Promote process. Different projects can use different account codes or Job class so the option of specifying project specific job cards allows for this scenario.
BATCHMIGRATE1=Job card 1 BATCHMIGRATE2=Job card 2 BATCHMIGRATE3=Job card 3 BATCHMIGRATE4=Job card 4	Set a default batch job for the Migrate process. Different projects can use different account codes or Job class so the option of specifying project specific job cards allows for this scenario.
BUILDSECURITY=Y/N	Specify Y to invoke SAF/RACF security call and possible surrogate ID switch. For more information about this, see Chapter 4, "SCLM security," on page 73.
PROMOTECURITY=Y/N	Specify Y to invoke SAF/RACF security call and possible surrogate ID switch. For more information about this, see Chapter 4, "SCLM security," on page 73.
DEPLOYSECURITY=Y/N	Specify Y to invoke SAF/RACF security call and possible surrogate ID switch. For more information about this, see Chapter 4, "SCLM security," on page 73.
ASCII=ASCII codepage	Specify the ASCII codepage to override the ASCII codepage specified in the TRANSLATE.conf file. For example: ASCII=UTF-8
EBCDIC=EBCDIC codepage	Specify the EBCDIC codepage to override the EBCDIC codepage specified in the TRANSLATE.conf file. For example: EBCDIC=IBM-420
TRANLANG=SCLM Language	Specify a TRANLANG parameter to be added to the list of TRANLANG parameters specified in the TRANSLATE.conf. For example: TRANLANG=DOC

Table 10. SITE/Project options (continued)

NOTRANLANG=SCLM Language	Use the NOTRANLANG keyword to remove an already specified TRANLANG from the list allowable for this SCLM project as specified in the TRANSLATE.conf. For example: NOTRANLANG=JAVA
LONGLANG=SCLM Language	Specify a LONGLANG parameter to be added to the list of LONGLANG parameters specified in the TRANSLATE.conf. For example: LONGLANG=DOC
NOLONGLANG=SCLM Language	Use the NOLONGLANG keyword to remove an already specified LONGLANG from the list allowable for this SCLM project as specified in the TRANSLATE.conf. For example: NOLONGLANG=COBOL
BIDIPROP=LANG=SCLM Language/* TextOrient=LTR/RTL TextType=Visual/ Logical SymetricSwap=On/Off NumericSwap=On/Off	Use the BIDIPROP keyword to set BiDirectional language defaults to SCLM languages. The LANG= can be set to either all SCLM languages or to specific SCLM languages. BiDirectional support is only supported under Rational Developer for System z. For more information see the "Rational Developer for System z Host Configuration Guide".
PROJECTLISTALL=Y	Project list flag if set to N will stop users selecting * as project filter. This may avoid long user catalog searches for all SCLM projects.

Example of using combinations of the TRANSLATE.conf overrides

The TRANSLATE.conf file sets up default settings for codepage support and default SCLM language support to be applied across the installation of SCLM Developer Toolkit. Below is a sample list of defaults that can be added to or modified:

```

CODEPAGE ASCII = ISO8859-1
CODEPAGE EBCDIC = IBM-1047
*
TRANLANG JAVABIN
TRANLANG J2EEBIN
TRANLANG J2EEOBJ
TRANLANG TEXTBIN
TRANLANG BINARY
TRANLANG DOC
TRANLANG XLS
*
LONGLANG JAVA
LONGLANG SQLJ
LONGLANG DOC
LONGLANG XLS

```

Example of using combinations of the TRANSLATE.conf overrides

```
|          LONGLANG J2EEPART  
|          LONGLANG JAVABIN  
|          LONGLANG J2EEBIN  
|          LONGLANG J2EEOBJ
```

Example of using combinations of the TRANSLATE.conf overrides

It is then possible for different SCLM projects, that are storing different types of data, maybe in different national languages, to override these default settings. So the project configuration file for SCLM project SCLMPRJ1 can have the following TRANSLATE.conf override settings:

```
* Arabic Codepage overrides
*
ASCII=UTF-8
EBCDIC=IBM-420
*
* Project specific TRANLANG and LONGLANG entries
*
TRANLANG=DOC
LONGLANG=DOC
```

This sets codepages for source translations to the Arabic codepage pair. Additionally an SCLM Language of DOC will be added to the defaults from the TRANSLATE.conf.

SCLM Project SCLMPRJ2 might have some different TRANSLATE.conf settings:

```
* Hebrew Codepage overrides
*
ASCII=UTF-8
EBCDIC=IBM-424
*
* Project specific TRANLANG and LONGLANG entries
*
TRANLANG=DOC
TRANLANG=XLS
NOTRANLANG=JAVABIN
NOTRANLANG=J2EEBIN
NOTRANLANG=J2EEOBJ
LONGLANG=DOC
LONGLANG=XLS
NOLONGLANG=COBOL
NOLONGLANG=J2EEPART
NOLONGLANG=JAVABIN
NOLONGLANG=J2EEBIN
NOLONGLANG=J2EEOBJ
```

This sets codepages for source translations to the Hebrew codepage pair. Additionally SCLM Languages of DOC and XLS are added to the defaults from the TRANSLATE.conf. In this case, however, the defaults set in TRANSLATE.conf are then removed. This is not really necessary, as having additional settings is not an issue, but it demonstrates how a project can be set up to only have the required SCLM languages for a specific SCLM project.

Example of using combinations of the BIDIPROP overrides

The BIDIPROP values specified in the SITE.conf file can be overridden by any of the BIDIPROP values specified in the SCLM project-specific project.conf files. For example the following is set in the SITE.conf:

```
*
* ----- SITE SPECIFIC BIDI OPTIONS -----
*
*
* BiDi Language default properties
*
BIDIPROP=LANG=*      TextOrient=LTR TextType=Visual SymetricSwap=Off NumericSwap=Off
```

Example of using combinations of the BIDIPROP overrides

This sets all SCLM languages to the specified settings. Now the following can be set in the ADMIN10.conf file:

```
*
* BiDi Language default properties
BIDIPROP=LANG=JAVA TextOrient=RTL TextType=Visual SymetricSwap=On NumericSwap=Off
BIDIPROP=LANG=COBOL TextOrient=RTL TextType=Logical SymetricSwap=Off NumericSwap=Off
```

These settings will override the settings in the SITE.conf for the JAVA and COBOL language definitions. All other languages will have the default settings specified in the SITE.conf.

Chapter 4. SCLM security

The SCLM Developer Toolkit offers optional security functionality if required for the Build, Promote, and Deploy functions.

- Security controlled by security flags set in the SCLM Developer Toolkit configuration files and by SAF/RACF rules.
- If the security flags are set for a function and the requesting user ID meets the SAF/RACF authority check then allow the SCLM Build, Promote, or Deploy to continue under the requesting userid or optionally to run under a surrogate user ID if defined in that SAF/RACF rule.

Build/Promote/Deploy security flag and process flow

You can set a Build, Promote, Deploy security flag in the SITE/PROJECT configuration files that specifies additional SAF security checking for that function and possible processing under a surrogate user ID.

- Read the SITE/Project configuration files to see whether the Build Security option is set on
BUILDSECURITY = Y
- Read the SITE/Project configuration files to see whether the Promote Security option is set on
PROMOTESecurity = Y
- Read the SITE/Project configuration files to see whether the DEPLOY Security option is set on
DEPLOYSECURITY = Y

If the security flag is set:

- After reading the sub-parameters from the function request, the translator makes a call to the SCLM security module to check SAF/RACF user authority against a defined rule.
- If authorised, the SAF/RACF application data associated with the rule is read for a surrogate user ID (suid=xxxxxxx) and, if a surrogate user ID exists, then the effective user ID of the current task switches to the new surrogate user ID otherwise processing continues under the originating user ID.

You have a choice for data set-naming conventions from the Build, Promote panel.

- If you are not authorized then RC=8 is returned on the build, promote, or deploy with security error message. Build, promote, or deploy processing is cancelled.

Security rules and surrogate user ID

The SCLM Build, Promote, and Deploy function security is to be defined using the SAF/RACROUTE security interface. The following security products are known to support this interface: RACF, ACF2, and TOP SECRET.

The examples that follow are modelled on RACF using the definition and permission commands RDEFINE and PERMIT. Other security products supporting SAF will have their own commands for equivalent definitions. Refer to the appropriate security product documentation for these definitions.

Security rules and surrogate user ID

- Use the RDEFINE command to define to RACF all resources belonging to classes specified in the class descriptor table. The RDEFINE command adds a profile for the resource to the RACF database in order to control access to the resource.
- The PERMIT command is used to maintain the lists of users and groups authorized to access a particular resource. The standard access list includes user IDs and group names authorized to access the resource and the level of access granted to each.
- Define function profiles for SCLM functions against the XFACILIT class.
- The security administrator defines the required RACF profiles using the RDEFINE command, and allows you to access with the PERMIT command.

Build rule format

Below is the format for the SCLM profile definition for Builds.

SCLM.BUILD.project.projdef.group.type.member

Field sub-parameter details:

- SCLM: Indicates an SCLM function profile
- BUILD: Indicates the BUILD function from SCLM
- Project: The SCLM project or * for all projects
- Projdef: The alternate project definition (default Project) or * for all alternate projects
- Group: The SCLM group to build at or * for all groups
- Type: The SCLM type or * for all types
- Member: The SCLM member to build or * for all members

Promote rule format

Below is the format for the SCLM profile definition for Promotes.

SCLM.PROMOTE.project.projdef.group.type.member

Field sub-parameter details:

- SCLM: Indicates an SCLM function profile
- PROMOTE: Indicates the PROMOTE function from SCLM
- Project: The SCLM project or * for all projects
- Projdef: The alternate project definition (default Project) or * for all alternate projects
- Group: The SCLM group to build at or * for all groups
- Type: The SCLM type or * for all types
- Member: The SCLM member to promote or * for all members

Deploy rule format

Below is the format for the SCLM profile definition for deployment:

SCLM.DEPLOY.server.application.node.cell.project.projdef.group.type

Field sub-parameter details:

- SCLM: Indicates an SCLM function profile
- DEPLOY: Indicates the DEPLOY function from SCLM
- Server: The target deployment server – SERVER_NAME in deploy Ant script or * for all servers
- Application: The target WAS application name – APPLICATION_NAME in Ant script or * for all applications
- Node: The target WAS node name – NODE_NAME in Ant script or * for all nodes

- Cell: The target WAS cell name – CELL_NAME in deploy Ant script or * for all cells
- Project: The SCLM project containing the EAR to be deployed or * for all projects
- Projdef: The alternate project definition (default Project) or * for all alternate projects
- Group: the SCLM group the deployable EAR is stored in or * for all groups
- Type: The SCLM type the deployable EAR is stored in or * for all types

SAF/RACF BUILD, PROMOTE, DEPLOY, and PROFILE rules

Below is an example for a Build profile rule using class XFACILIT which would be defined by the security administrator. The same sample can be used for a PROMOTE rule by replacing the SCLM.BUILD.* rule with the SCLM.PROMOTE.* rule.

Note: As with standard RACF rules instead of specific parameters generic “*” fields can be used. If you match the appropriate SAF/RACF rule, you will be allowed BUILD access if you have READ access on the rule. The default access for the rule will be NONE → UACC(NONE). Your access rule is defined by the PERMIT command.

The following rule secures all members for build in project=TESTPROJ at group level prod. Also a surrogate user ID is stored in the rule under application data - APPLDATA('SUID=xxxxx').

```
RDEFINE XFACILIT SCLM.BUILD.TESTPROJ.TESTPROJ.PROD.*.* UACC(NONE) APPLDATA('SUID=PMEANEY')
```

This example defines an SCLM build profile where:

- Project = TESTPROJ
- Alternate project definition = TESTPROJ
- SCLM group = PROD
- SCLM type = all types
- Member = all members

Note: A surrogate user ID of PMEANEY is stored in the application data.

The following example shows security permissions defined for individual users and user groups for the above example. This is also defined by the security administrator.

```
PERMIT SCLM.BUILD.TESTPROJ.*.*.*.* CLASS(XFACILIT) ID(HOGES) ACCESS(READ)
```

Through the use of wildcard characters, the PERMIT matches the original RDEFINE profile and permits user HOGES to build any member from project TESTPROJ and group PROD. Since there is a surrogate user ID stored in the 'APPLICATION DATA' of the matching rule then the BUILD is processed under that surrogate user ID (in this case PMEANEY).

Below is an example for a deployment profile using class XFACILIT which would be defined by the security administrator:

```
RDEFINE
XFACILIT SCLM.DEPLOY.servery.testapp.node1.cell11.TESTPROJ.TESTPROJ.PROD.J2EEDEP
UACC(NONE)
```

SAF/RACF BUILD, PROMOTE, DEPLOY, and PROFILE rules

This defines an SCLM deploy profile for WAS:

```
server name = servery
application name = testapp
node name = node1
Cell name = cell1
```

SCLM project details:

```
Project = TESTPROJ
Alternate project definition = TESTPROJ
Scm group = PROD
Scm type = J2EEDEP
```

The following two examples show different security permissions defined for individual users and user groups for the above example. This would also be defined by the security administrator. UACC authority for the profiles would be ACCESS(NONE):

```
PERMIT SCLM.DEPLOY.servery.testapp.*.*.*.*.* CLASS(XFACILIT) ID(HOGES) ACCESS(READ)
```

Through the use of wildcard characters, the PERMIT matches the original RDEFINE profile and permits user HOGES to deploy on WAS server=servery, application=testapp, node=node1, cell=cell1, SCLM project=TESTPROJ, projdef=TESTPROJ, group=PROD, and type=J2EEDEP.

```
PERMIT SCLM.DEPLOY.*.*.*.*.*.*.* CLASS(XFACILIT) ID(J2EEGRP) ACCESS(read)
```

This matches the original RDEFINE profile and permits any user who belongs to the RACF group J2EEGRP to deploy on the above server and from the same SCLM project details.

Chapter 5. CRON-initiated Builds and Promotes

Although most Builds and Promotes are initiated via the Developer Toolkit client, there is a provision to set up Build and Promote configuration files within the z/OS UNIX System Services file system and to initiate these builds or promotes via the CRON (time) service within UNIX System Services. Using this method, the SCLM Developer Toolkit Client is not required, as the relevant Build and Promote parameters are read from a z/OS UNIX System Services file system configuration file and passed to the Developer Toolkit Host component for SCLM processing.

Below is a description of the SCLM Developer Toolkit samples that provide CRON-initiated Builds and Promotes. These samples are available in the installed Developer Toolkit SBWBSAMP data set.

Sample member	Description
---------------	-------------

BWBCRON1	This REXX sample calls the SCLM Developer Toolkit host interface and passes the function parameters. Output from the function process by default is displayed to STDOUT but might be re-directed to a z/OS UNIX System Services file system file or log.
-----------------	--

This sample can be copied into the z/OS UNIX System Services file system into a directory path of your choice to run. The sample will need to be customized as detailed within the sample.

This REXX sample must be run in conjunction with input from sample BWBCRONB for a build or sample BWBCRONP for a promote.

BWBCRONB	This REXX sample sets up the Build parameter input string which is passed to module BWBCRON1.
-----------------	---

The sample requires user customization to update all required build parameters.

This sample must be copied into a user determined z/OS UNIX System Services file system directory (optionally renamed) to be run with sample BWBCRON1.

BWBCRONP	This REXX sample sets up the Promote parameter input string which will be passed to module BWBCRON1.
-----------------	--

The sample requires user customization to update all required promote parameters.

This sample must be copied into a user determined z/OS UNIX System Services file system directory (optionally renamed) to be run with sample BWBCRON1.

Figure 31. Sample CRON members

STEPLIB and PATH requirements

The PATH and STEPLIB variables in either the system wide profile (/etc/profile) or user profile (/u/userid/.profile) will need to be set to locate the CRON jobs (\$PATH) and locate the SCLM Developer Toolkit modules (\$STEPLIB) if the SCLM Developer Toolkit data set is not in the LINKLIST.

STEPLIB and PATH requirements

Example:

The samples BWBCRON1 and BWBCRONB are copied to a test directory /var/SCLMDT/CRONJOBS. The following z/OS UNIX System Services file system PATH and STEPLIB variables are set in /etc/profile:

```
PATH=/var/SCLMDT/CRONJOBS:$PATH
STEPLIB=BWB.SBWBLOAD:$STEPLIB
```

CRON Build job execution

After the CRON jobs are added to the PATH variable they can be run by piping the output from the parameter_exec into the processing_exec. The output can then be directed to an output log file.

Syntax

```
parameter_exec | processing_exec > output.log
```

The “|” is the z/OS UNIX System Services pipe symbol.

Invocation Example:

Using the sample names as provided the CRON build exec can be invoked as follows:

```
BWBCRONB | BWBCRON1 >bwbcronb.log
```

Additionally this sample Build execution can be added to a CRONTAB file to run at 7.30pm Monday- Friday:

```
30 19 * * 1-5 BWBCRONB|BWBCRON1 >bwbcronb.log ;
```

For further information about the CRON services available and the CRONTAB format see the following manuals:

- z/OS UNIX System Services Commands
- z/OS UNIX System Services Planning

Alternatively use the online manual help (man) under z/OS UNIX System Services:

- man cron
- man crontab
- man at

CRON Build job samples

Below are the BWBCRON1 and BWBCRONB job samples as provided in the SBWBSAMP library.

```

/* REXX */
/* Customize STEPLIB, CGI_DTCONF and CGI_DTWORk BELOW */
/*
The STEPLIB should reflect the install load library for SCLM Developer toolkit.
If this dataset resides in the LINKLIST then set STEPLIB to '' .
*/
STEPLIB= 'BWB.SWBLOAD'
/*
The Environment variable CGI_DTCONF determines the HOME
directory path where the configuration files reside for SCLM Developer Toolkit.
This was determined by the install directory specified in install job BWBINST1.
By default /etc/SCLMDT .
*/
CGI_DTCONF = '/etc/SCLMDT'
CGI_DTWORk = '/var/SCLMDT'
/* */
/* SAMPLE USEAGE */
/*
COMMAND : BWBCRONB|BWBCRON1 >BWBCRONB.log (passes
build parameter list to BWBCRON1 & outputs to BWBCRONB.log)
*/

/* DO NOT ALTER BELOW */
CALL ENVIRONMENT 'STEPLIB',STEPLIB
CALL ENVIRONMENT 'CGI_DTCONF',CGI_DTCONF
CALL ENVIRONMENT 'CGI_DTWORk',CGI_DTWORk CALL

BWBINT

EXIT

```

Figure 32. Sample CRON Build Exec

```

/* REXX */
/* SAMPLE BUILD PARAMETER FILE USED FOR CRON INITIATED BUILDS */
/* Update Build parameters below */
/* if parameter required as Blank then set as '' */
FUNCTION = 'BUILD'
PROJECT  = 'PROJ1' /* SCLM Project          */
PROJDEF  = ''      /* Alt proj definition    */
TYPE     = 'SOURCE' /* SCLM Type             */
MEMBER   = 'TESTMEM' /* SCLM Member name      */
GROUP    = 'DEV1'  /* SCLM Group            */
GROUPBLD = ''      /* Build at Group         */
REPDGRP  = 'DEV1'  /* Users Development group */
BLDREPT  = 'Y'     /* Generate Build report  */
BLDLIST  = 'Y'     /* Generate List on error */
BLDMSG   = 'Y'     /* Generate Build Messages */
BLDScope = 'N'     /* Build Scope E/L/N/S    */
BLDMODE  = 'C'     /* Build Mode C/F/R/U     */
BLDMSGDS = ''      /* Message data set       */
BLDRPTDS = ''      /* Report data set        */
BLDLSTDS = ''      /* list data set          */
BLDEXTDS = ''      /* Exit data set          */
SUBMIT   = 'BATCH' /* Online or Batch        */
/* DO NOT ALTER PARM BUILD VARIABLE BELOW */
PARM1 = 'SCLMFUNC='FUNCTION'&PROJECT='PROJECT'&PROJDEF='PROJDEF'|,
        '&TYPE='TYPE'&MEMBER='MEMBER'&GROUP='GROUP'&GROUPBLD='GROUPBLD'|,
        '&REPDGRP='REPDGRP'&BLDREPT='BLDREPT'&BLDLIST='BLDLIST'|,
        '&BLDMSG='BLDMSG'&BLDScope='BLDScope'&BLDMODE='BLDMODE'|,
        '&BLDMSGDS='BLDMSGDS'&BLDRPTDS='BLDRPTDS'&BLDLSTDS='BLDLSTDS'|,
        '&BLDEXTDS='BLDEXTDS'&SUBMIT='SUBMIT
/* outputs parameter string as input to BWBCRON1 */
SAY PARM1

```

Figure 33. Sample Build parameter file

Appendix A. SCLM overview

IBM SCLM Developer Toolkit provides the means by which distributed applications written in Eclipse can be managed and built using SCLM, the IBM z/OS source code management system.

The language and tools used by distributed and mainframe users is as different as the environments they employ. By identifying and understanding key concepts of both environments then it is possible to successfully integrate these into a cohesive structure.

In terms of application structure Developer Toolkit is a series of Eclipse plug-ins with corresponding z/OS host code that enables both the use of HTTP and RSE transports. Operationally an Eclipse developer registers a workspace project with SCLM. Files in the project can be added to an SCLM project, checked in and out, and optionally built and deployed. All these services are driven via the Team Actions menu. From the SCLM administrators point of view, they can create projects, types, languages and associated built translators. Features such as change and authorization codes are dependent on requirements.

SCLM Concepts

From a Java/J2EE developers perspective, the following concepts help describe SCLM.

File naming

The z/OS file system only supports file name lengths of eight characters. The Developer Toolkit interface provides a translation service that enables normal Java long name conventions to be supported. There are files specific to SCLM that must comply with the naming restriction. These relate principally to the ARCHDEF structure described in “J2EE ARCHDEF” on page 46.

Type

Each file (known as a member in SCLM terminology) that is stored in an SCLM project is stored in a data set. The data set where the file is stored is identified by SCLM type. The type is part of the data set name, made up as far as SCLM is concerned by SCLM Project.Group.Type. with a language associated with it. There can be many types defined in an SCLM project. These types provide a means whereby two files with the same name can be stored in the same SCLM project. Each project can contain many types. By applying the use of type it is possible to store multiple Eclipse projects in the one SCLM project even though each IDE project potentially has a .project and .classpath file. If we do not segregate these files using type, then only one copy of these files exists in SCLM.

The SCLM administrator is responsible for the definition of the SCLM types. When you share a project with SCLM you need to know what type you are to use when storing objects in SCLM.

Language

When you add a file to SCLM you must store it with a certain language definition. Again the SCLM administrator is responsible for the definition of languages. This definition controls the behavior of SCLM as files are transferred to and from the host system. Using the language definition it is possible to define whether a certain file type is long-name translated, stored as a binary object, or translated into ASCII or EBCDIC (native z/OS encoding). For example a language definition of JAVABIN might relate to a long name translated binary object. Alternatively, WEBHTML can be defined as representing a long-name translated, text file to be stored in ASCII. The number of language definitions is defined per project. Understanding which language to use is necessary to ensure that the file is stored and can be retrieved correctly from SCLM. The language also defines how SCLM builds an object.

SCLM properties

Any file that is under SCLM control will have a number of properties associated with it. These properties are effectively the mapping mechanism between the IDE file and its corresponding SCLM properties. When service calls are made to SCLM, this data is read to formulate the appropriate service parameters. You can view these from the Properties menu when you highlight an SCLM-controlled member from Eclipse.

SCLM project structure

When you share a project with SCLM you also need to nominate what development group you belong to. SCLM project structures are hierarchical in nature. A typical hierarchy can be:

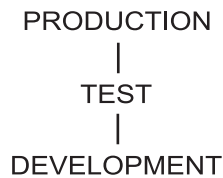


Figure 34. Multiple types

Code is initially stored at the DEVELOPMENT level. After it has been successfully built and tested it can be promoted up to TEST. Following successful testing it can then be promoted to PRODUCTION. This generally represents the developed product. If a defect is found in the Production level code then those files that must be edited to fix the defect are copied down to the Development level and the build process starts again. All the code that makes up the application is not copied down to the Development level. SCLM keeps track of the elements that make up the build and at what level they are stored.

Within the Development level there can be multiple groups. This provides a means of separating out code stored at the development level. SCLM also provides controls for determining the behavior of code stored in different development groups in terms of the ability to promote.

ARCHDEF

The structure of the IDE project is generally one composed of one or more IDE projects. By storing each of these IDE projects in a different SCLM type this structure is maintained. The ARCHDEF file then effectively defines the files that make up an IDE project. Each SCLM project can have multiple ARCHDEFs. It is possible for an ARCHDEF to reference other ARCHDEFs so that this multiple IDE project structure can be defined to the build process, the ARCHDEF being the principal means of defining a build list to SCLM. The closest analogy of this is that of a make process. The ARCHDEF lists the files that make up the build in addition to specifying a build script that will allow you to specify the location of external JARs or classes. For more information about this, see the User Manual section of the Online Help System.

JAVA/J2EE concepts

When an IDE project is created in the workspace, a project description file is automatically generated and stored under the name **.project**. This XML document contains descriptions of any 'builders' or 'natures' associated with the project. 'Builders' are incremental project builders that create some built state based on the project contents. As the contents of the project changes, this file will be updated. 'Natures' define and manage the association between a given project and a particular plug-in or feature.

The **.classpath** file is a file that describes the path which is used to find external jars and classes that are referenced by the source code in your IDE project. The equivalent function during a build through SCLM Developer is defined with the CLASSPATH_JARS directive in the Ant build scripts. This directive will describe the path on the z/OS host that is used to find external jars and classes that are referenced by the source code in your IDE project.

Both **.classpath** and **.project** are used to preserve your IDE project configuration so that it can be recreated in another workspace. For this reason it is recommended that both are checked into SCLM as part of the IDE project.

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, RAR or EAR files.

JAR files typically relate to standard Java applications or Enterprise Java Beans (EJB).

WAR files are created for Web applications. Typically these are composed of Java servlets, JSPs, and HTML files. WAR applications are often the front end of Web-based applications. These applications can 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.

RAR file development is not currently supported in Developer Toolkit.

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 container 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.

Within the Eclipse workspace the projects are effectively proximate, that is within the IDE environment they can effectively refer to other IDE projects readily in terms of packaging. Within SCLM, each of these IDE projects (for example, WAR, JAR and EAR projects) need to be mapped into a single SCLM project, with each project differentiated through the use of a different SCLM type. The reason for this is that there are common file names used in many IDE projects such as .project, .classpath, web.xml and application.xml, so use of separate types allows these same named parts to exist in the same SCLM project. For more information about mapping, see “Mapping J2EE projects to SCLM” on page 54.

From an SCLM perspective the development of the EAR application is best referenced through the use of a high-level ARCHDEF structure. Within SCLM the high-level ARCHDEFs, in many SCLM projects referred to as a *package*, are the apex of the ARCHDEF structure followed by the EAR application and lower level references (WAR and JAR files) that make up the EAR application. This structure enables the use of builds at both high and low level and also the use of full or conditional builds. The ARCHDEFs thus provide a means by which it is also possible to define the J2EE project elements within the SCLM project.

Appendix B. SQLJ Support

SQLJ is a language extension for Java. It is one of several technologies that allow Java programmers to include database communication in their programs. SQLJ provides a means to produce static, embedded SQL that generally out-performs dynamic equivalents such as JDBC.

SCLM Developer Toolkit ships with sample scripts allowing you to build SQLJ enabled Java programs using DB2.

It is hoped that after reading this chapter, you will understand the essentials of SQLJ, and how to apply this knowledge while using SCLM Developer Toolkit.

What is SQL?

SQL is an acronym for *Structured Query Language*. It is an open language, used to query, add to, remove from, and change, data in a Relational Database Management System (RDMS).

The first implementation of this language was in an early IBM database product in the 1970's : System R . Since then, SQL has grown, been standardized (by ANSI and ISO) and appeared in many flavours on many different database systems.

What is DB2?

DB2 is a popular database system, traditionally for the mainframe platform, that has since been extended onto many others. It is the defacto standard for relational database management systems on z/OS.

DB2 UDB Version 8 is the version that SCLM Developer Toolkit s build scripts are based on. References to DB2 in this chapter refer specifically to DB2 UDB Version 8.

What is JDBC?

JDBC stands for *Java Database Connectivity*. In Java development, this is a well known and commonly used technology for implementing database interaction. JDBC is a call-level API, meaning SQL statements are passed as strings to the API, which then takes care of executing them on the RDMS. Consequently, the value of these strings can be changed at runtime, making JDBC *dynamic*.

While JDBC programs will execute slower than their SQLJ equivalents, one advantage of this approach is a concept known as Write once, call anywhere. This means that since no interaction is required until runtime, a JDBC program is very portable and can be taken between different systems with minimum fuss.

What is SQLJ?

SQLJ is a language extension used for database transactions in Java applications. It produces static, embedded SQLJ. The term is made up of SQL - *Structured Query Language* and J which stands for *Java*.

SQLJ is *static* because the SQL statements that will be executed at runtime are known when the program is assembled. Contrast this to JDBC, where the queries that are executed can be changed at any time.

SQLJ is *embedded* because during binding, a serialized form of the programs SQL statements is given to the database. The database uses this serialized data to determine optimized access paths to the tables that are referenced within. In JDBC, the database has no way to determine which statements will be executed, until it receives them at runtime from the application. Therefore it must determine access paths at runtime. This incurs an overhead that is avoided by using SQLJ.

Comparing JDBC and SQLJ

This table is based on material found in section 5.2 the Redbook *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*, which is recommended reading.

	SQLJ (static)	JDBC (dynamic)
PERFORMANCE	Most of the time, static SQL is faster than dynamic SQL, because at runtime only the authorization for packages and plans must be checked prior to the execution of the program.	Dynamic SQL statements require the SQL statements to be parsed, table/view authorization to be checked, and the optimization path to be determined.
AUTHORIZATION	With SQLJ, the owner of the application grants EXECUTE authority on the plan or package, and the recipient of that GRANT must run the application as written.	With JDBC, the owner of the application grants privileges on all the underlying tables that are used by the application. The recipient of those privileges can do anything that is allowed by those privileges, for example, using them outside the application the authorizations were originally granted for. The application cannot control what the user can do.
DEBUGGING	SQLJ is not an API but a language extension. This means that the SQLJ tooling is aware of SQL statements in your program, and checks them for correct syntax and authorization during the program development process.	JDBC is a pure call-level API. This means that the Java compiler does not know anything about SQL statements at all they only appear as arguments to method calls. If one of your statements is in error, you will not catch that error until runtime when the database complains about it.

IMONITORING	With SQLJ, you get much better system monitoring and performance reporting. Static SQL packages give you the names of the programs that are running at any given point in time. This is extremely useful for studying CPU consumption by the various applications, locking issues (such as deadlock or time-out), etc.	Where in SQLJ you can determine the name of the program currently executing, with JDBC all transactions occur through the same program. This makes monitoring and locating problem areas more difficult.
VERBOSITY	As SQLJ statements are coded in purely SQL syntax, without the need to wrap them in a Java method, the programs themselves are easier to read, making them easier to maintain. Also, since some of the boilerplate code which has to be coded explicitly in JDBC is generated automatically in SQLJ, programs written in SQLJ tend to be shorter than equivalent JDBC programs.	With JDBC, all SQL statements must be wrapped in API calls that generally make for unclear and verbose code.

What is a Serialized Profile?

Code that is written in SQLJ is placed in a file with a .sqlj extension. In the first step of program preparation (that will be discussed in more detail later on), the .sqlj file is fed into the SQLJ translator.

The translator produces two types of output. The first is Java source code (.java). This source code is obviously the Java implementation of the code within the .sqlj file.

The second type of output is a serialized profile (.ser). This file contains all the SQL statements from the .sqlj file, in a serialized form. This profile must be available to the program at runtime, and it can also be used to bind to the RDMS.

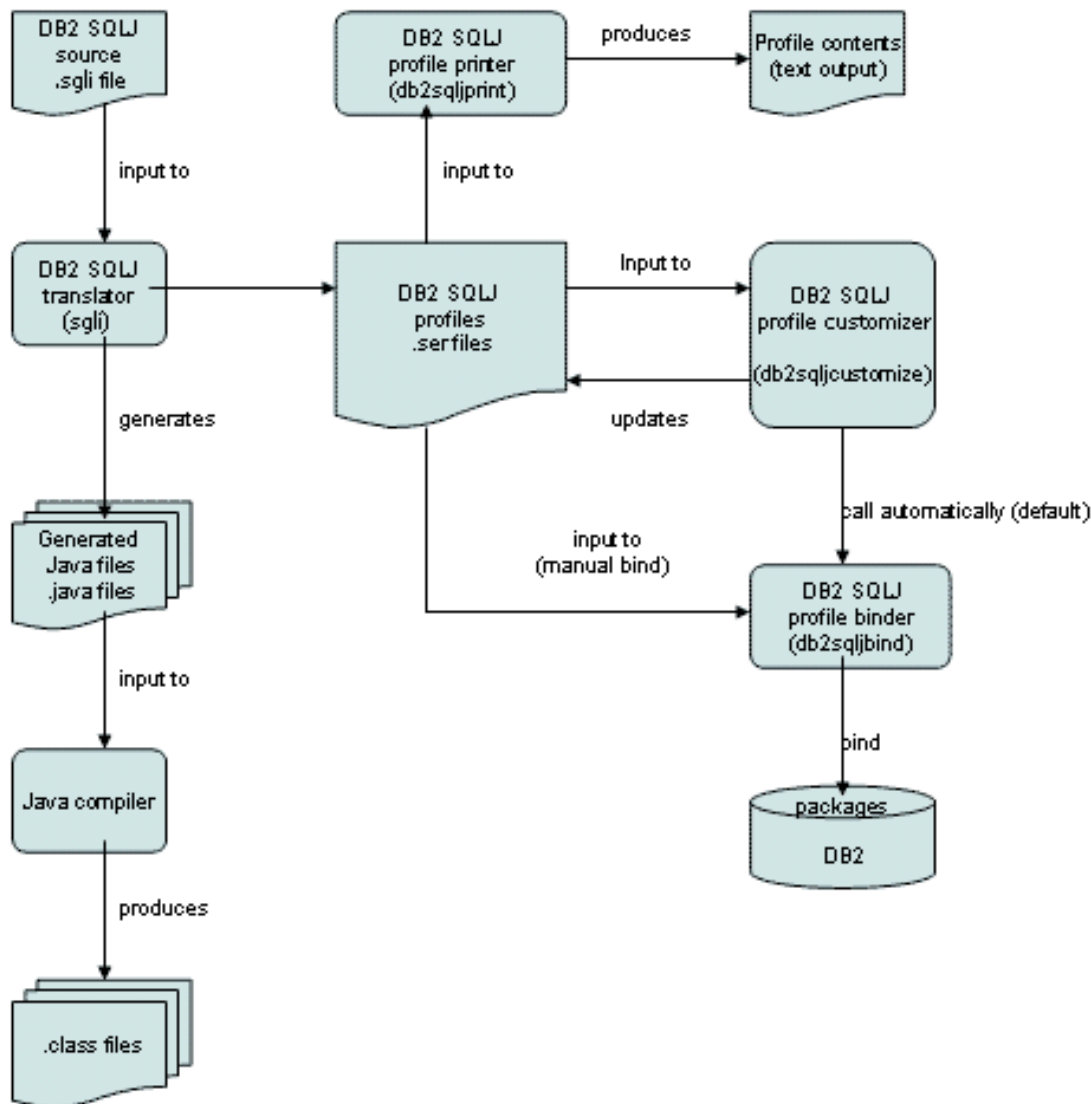
What is a DBRM?

DBRM stands for *Database Request Module*. This is the traditional DB2 serialized representation of the SQL statements in a program. For example, a program may be written in COBOL. This program will be pre-processed by DB2 to produce a DBRM that will be used to bind against a particular DB2 subsystem.

With SQLJ, the process is slightly different, and is referred to in DB2 UDB Version 8 terms as *compatibility mode*. The utility *db2sqljcustomize* can be provided with optional command line arguments that cause a DBRM to be generated. This DBRM can then be bound to DB2 using traditional means, for example, a REXX script called by a SCLM user exit.

SQLJ Program Preparation

Before we discuss how to use SCLM Developer Toolkit to build SQLJ programs, let us first examine the manual process. This process is for the DB2 implementation of SQLJ, and features 3 commands called *sqlj*, *db2sqljcustomize* and *db2bind*. Note that the bind step can optionally be performed in *db2sqljcustomize*, so *db2bind* is not always required.



Translation

The SQLJ translator (not to be confused with an *SCLM language translator*) takes SQLJ source files as input, and produces Java source code (.java files) and serialized profiles (.ser files).

The SQLJ language itself is not discussed in this book. Consult (<http://www.sql.org>) to find references on developing SQLJ code.

The number of serialized profiles generated per .sqlj file depends upon the number of *connection context* classes referenced within the SQLJ code. A serialized profile will be generated for each one.

Many SQLJ source files will only reference a single connection context class, and therefore generate a single serialized profile. The serialized profiles are named according to the order that they are referenced in the source file. The name takes the following format:

progrname_SJProfileX.ser

Where:

Progrname	The name of the program. This is determined by removing the .sqlj extension from the input source filename.
X	An integer representing the index of the current class. Indexing is zero based. The first connection context class referenced will produce profile 0; the second will produce profile 1, and so on.

Example:

Input: Customer.sqlj (references one connection context class)

Output: Customer.java
Customer_SJProfile0.ser

And optionally, if the argument `-compile=true` is supplied to sqlj:
Customer.class

Customization

Once the serialized profiles are generated, we customize them. The command for doing so in DB2 Version 8 is *db2sqljcustomize*; however, in previous versions it was *db2profsc*. Each invocation of the customizer should match up with an invocation with the SQLJ translator. In other words, if a single invocation of the translator generated 5 profiles, then those 5 profiles must be fed as input to a single invocation of the profile customizer. Another way to think of it is to associate each individual program name with one invocation of each of the utilities. Remember that the program name is the same as the input source filename with the .sqlj extension removed.

Customization adds DB2 specific information to the serialized profile that is used at runtime. Other options, such as automatic binding, can be configured via command line switches. If you are using a legacy version of DB2, or you are specifying the *gendbrm* and *dbrmdir* flags for *db2sqljcustomize*, a DBRM file will be generated. This file is used later to bind to the database. With the Universal driver from DB2 UDB 8+, you may forgo the generation of a DBRM, and instead bind using the serialized profiles generated by the SQLJ translator.

Binding

Binding is the last step in the SQLJ program preparation process. In DB2 version 8, the command used to bind is *db2sqljbind*, or you may bind automatically when running *db2sqljcustomize*. Binding is the step that builds an access path to DB2

tables for your serialized SQL statements. These statements are available either in the form of a DBRM or a serialized profile.

By default, four packages are created, one for each isolation level. You can either bind using the traditional method, wherein a DBRM is used, or the new Universal method, where serialized profiles are used instead.

SCLM DT types and translators

Before discussing the SCLM types and translators, an important distinction must be made between an *SCLM language translator*, or simply, *SCLM translator* and the SQLJ translator *sqlj* that ships as part of DB2.

In SCLM, any defined language is required to have a translator so it is known how to deal with that language. This is not the same as the SQLJ translator, *sqlj* that is a command line utility that takes a SQLJ source file and produces serialized profiles and Java source code.

Having made that distinction, we will discuss the *SCLM Types* and *SCLM Translators* associated with the SQLJ build process.

A SCLM translator for SQLJ is provided and should be assigned as the language type of all SQLJ source code stored in SCLM. This new translator requires additional SCLM Types to be defined. The SCLM translator for SQLJ is similar to the JAVA translator but contains additional IOTYPE definitions for SCLM output types SQLJSER and DBRMLIB. If you do not wish to generate DBRM files as part of the customization step then this DBRMLIB IOTYPE may be removed from the SQLJ language definition.

Within the project definition, an administrator must define and generate the new SCLM translator and the additional types.

SQLJSER	This is required to store the generated serialized profile files (.ser files) created/customized in the translation and customization steps. It is recommended to define this SCLM type dataset as recfm=VB, lrecl= 256.
DBRMLIB	A type required to contain the generated DBRM files created in the customization step. This type is only required for customers using generated DBRM files as part of their DB2 bind processing. This SCLM type dataset must be allocated as recfm=FB , lrecl= 80.

Tailoring the build process

In order to retain maximum flexibility, the SQLJ build process is highly customizable, to cater for different site configurations, and any combination of parameters that must be passed to *sqlj* and *db2sqljcustomize*.

This section describes the concepts behind SCLM Developer Toolkit s implementation of SQLJ. It is hoped that after reading it, you will be able to customize the build process to match the requirements of your site.

While doing SQLJ translation and profile customization, SCLM Developer Toolkit directly invokes the same Java classes used by the commands *sqlj* and *db2sqljcustomize*. Be aware that the arguments supplied to the SCLM DT translation and customization processes will be exactly the same. For an in-depth discussion of all the command line arguments for each command, please consult the DB2 Universal Database user guide.

Tailoring the Build Script

Assuming you have used the Add to SCLM wizard, the build script for your SQLJ program will be given the same member name as the Archdef. For example, if the Archdef for your sqlj project is:

```
SCLM10.DEV1.ARCHDEF(SQLJ01)
```

You will locate the build script at:

```
SCLM10.DEV1.J2EEBLD(SQLJ01)
```

In that build script will be a reference to the master build script. This can be found in the property. *The build script shipped with Developer Toolkit is BWBSQLB for JAR projects, and BWBSQLBE for EJB projects. You should not need to change this value.* Most of the configuration listed for the translation and customization steps goes on in this file.

sqlj properties

Each property listed in the table below appears in the BWBSQLB build script. The properties are in XML form as follows:

Configuring the script involves changing the value for any relevant properties.

NAME	VALUE	DESCRIPTION
sqlj.exec	/etc/SCLMDT/bwbsqlc.rex	Specifies the location of the sqlj & db2sqljcustomize exec routine bwbsqlc.rex. (Sample BWBSQLC) By default this sample should be located in a SCLM DT install directory.
sqlj.class	sqlj.tools.Sqlj	Specify the sqlj class name. This is the name of the class invoked by the <i>sqlj</i> utility. It is very unlikely you will need to change this value.
sqlj.bin	/db2path/bin	Specify the db2 sqlj bin directory location where the sqlj script resides
sqlj.cp	/db2path/jcc/classes/sqlj.zip	Specify the location of sqlj.zip for inclusion on the classpath.
sqlj.arg	-compile=false status linemap=NO db2optimize	Specify global property arguments below for sqlj processing.

db2sqljcustomize properties

Each property listed in the table below appears in the BWBSQLB build script. The properties are in XML form as follows:

```
<property name= NAME value= VALUE />
```

Configuring the script involves changing the value for any relevant properties.

NAME	VALUE	DESCRIPTION
sqljdb2cust.class	com.ibm.db2.jcc.sqlj.Customizer	Specify the sqlj db2 customize class name. It is very unlikely that you should need to change this value.
db2sqljcust.cp	/db2path/jcc/classes/db2jcc.jar: ./SRC: /db2path/jcc/classes/ db2jcc_license_cisuz.jar	Classpath settings for the customize utility. Fully qualified pathnames must be supplied in XML.
db2sqljcust.arg	-automaticbind NO -onlinecheck YES -staticpositioned YES -bindoptions â ISOLATION(CS)â -genDBRM	General arguments to supply to the customization utility.
db2sqljcust.propfile	user.properties	Temporary property file name to be passed to a user property determination script for dynamic property values. Leave as default.
db2sqljcust.userpgm	NONE if you wish to bypass the script. Otherwise, specify the fully qualified path and file name of user script.	This script will be run immediately before the customization utility. It dynamically updates a property file that is used as input to the customization utility.

custom user script

The SQLJ build script provided by SCLM Developer Toolkit is designed to work in DB2 UDB v8 compatibility mode. This mode supports the DB2 concept of DBRM's, rather than binding via the serialized profiles. In order to use the serialized profiles, changes must be made to BWBSQLB. This is discussed in the subtopic *Using serialized profiles for binding*.

Binding methodology aside, in order to match the build process to your site, you will probably need to customize the arguments to *sqlj* and *db2sqljcustomize* to match your database environment, isolation policy, and other factors. You may even want to put your own scripts in to determine dynamic properties for these arguments: for example you may wish to intelligently create a package name related to the input file name.

SCLM Developer Toolkit allows you to do this by specifying your own customization script. Everything in the ANT XML build process works on the concept of "properties", XML *Property* elements specifying a name/value pair. For example, in the *db2sqljcustomize* step in build script BWBSQLB, the global command line arguments to be supplied to *db2sqljcustomize* are defined in a property element with the name *db2sqljcust.arg* and a default value of

-automaticbind NO -onlinecheck YES -staticpositioned YES -bindoptions "ISOLATION(CS)" genDBRM lang=EN-AU

If you want to change the arguments supplied, you can both edit the build script to change the value of the property, which would change the settings globally, or hook your own customization script into the process.

To hook in your own custom property script, place the name of your script in `db2sqljcust.userpgm`, and the name of the property file you wish to write to in `db2sqljcust.propfile`.

The script specified in `db2sqljcust.userpgm` will be run immediately before the `db2sqljcustomize` process. Your script will dynamically update the property file specified in `db2sqljcust.userpgm`. This property file will be used as input to the `db2sqljcustomize` process, as the build process concatenates both properties in the dynamically updated property file, and properties already defined in the build script.

The script specified in `db2sqljcust.userpgm` will be supplied the following arguments when it is executed.

Argument	Description
Basedir	Base directory (workspace directory)
Propfile	The name of the property file to create and update. <i>Note: The property file being created needs to be basedir/'/propfile</i>
Sqljf	A list of file names, representing the serialized profiles (.ser) to be processed by <code>db2sqljcustomize</code> .

The properties should be set in the file in the following format, with one property declaration per line:

```
argument=value
e.g.
singlepkgname= pkgname
```

For example:

```
pkgversion=1
url=jdbc:db2://site1.com:80/MVS01
qualifier=DBT
singlepkgname= SQLJ986
```

The custom routine is called once per file. Finally, the argument properties are used for building up the required argument string for the `db2sqljcustomize` call. For example:

```
db2sqljcustomize -automaticbind NO -collection ${db2.collid}
-url ${db2.url} -user ${db2.user} -password ???????? -onlinecheck YES
-qualifier ${db2.qual} -staticpositioned YES -pkgversion ${db2.packversion}
-bindoptions "ISOLATION (CS)"
-genDBRM -DBRMDir DBRMLIB
-singlepkgname ${db2.pack}
```

Binding [DBRM]

Traditional DB2 uses a *Database Request Module* or DBRM for this purpose. The DBRM is generated by the `db2sqljcustomize` command when the flag `gendbrm` is

provided. Without this flag, the command will assume you wish to bind via serialized profiles, and will not generate a DBRM.

If you provide this parameter, SCLM Developer Toolkit will pick up the generated DBRM s, and store them in SCLM for future use. One advantage of using this technique is that you can easily perform a DB2 bind in a SCLM user exit, such as the build/copy exit.

Since the build/copy user exit is automatically provided with a list of updated objects, you can selectively rebind only the modules that have changed, avoiding inefficiency through redundant binding.

To configure binding for DBRMs there are four steps.

1. Set the appropriate arguments for *sqlj*.

```
<!-- specify global property arguments below for sqlj processing -->
<property name="sqlj.arg"
value="-compile=false -status -linemap=no"/>
```

Argument	Description
compile=false	Setting this option to false prevents the sqlj translator from automatically compiling the Java source it produces. SCLM Developer Toolkit uses the generated source in its own build process, so it is recommended you always set this option to false.
linemap=no	Specifies whether line numbers in Java exceptions match line numbers in the SQLJ source files (the .sqlj file) or line numbers in the Java source file that is generated by the SQLJ translator files (the .java file). This requires a .class file, so must be set to <i>no</i> when used in conjunction with <i>compile=false</i> .
status	Prints immediate status display of SQLJ processing.

2. Set the appropriate arguments for *db2sqljcustomize*, including *gendbrm*.

```
<property name="db2sqljcust.arg"
Value='-automaticbind NO -onlinecheck YES
-bindoptions "ISOLATION(CS)" -gendbrm' />
```

Argument	Description
automaticbind no	When set to no , the customizer will not perform a bind when customization is complete.
onlinecheck yes	Perform online checking on the system specified by the <i>url</i> parameter. Defaults to yes if <i>url</i> is supplied, and no otherwise.
Bindoptions ISOLATION(CS)	Instructs the binder to create a single package (cursor stability). Used in conjunction with <i>singlepkgname</i> (set dynamically).
gendbrm	Instructs the customizer to generate DBRM files.

3. Configure the user script.

Set the location of your user program in BWBSQLB. This tells the build process where to find the rex script used to calculate dynamic properties.

The big property we want to configure dynamically is *singlepkgname*. This is the name of the package to bind to, and each program is going to have its own unique package name, which in this simple example, is the first 8 letters of the program name.

4. Write a build exit to bind the DBRM. The build copy exit is recommended.
Since we are using *singlepkgname* in the customization step, the name of the package will be the same as the name of the DBRM.

Binding [Serialized Profile]

The new and recommended approach for binding SQLJ programs is to use the serialized profiles (.ser files) to bind. This was inevitable since the serialized profile performs the same function as the DBRM: providing a serialized image of the statements within the program.

With some small modifications to the build script BWBSQLB, you can configure SCLM Developer Toolkit to use this method instead. It is simply a matter of changing the arguments provided to *db2sqljcustomize* to remove the *gendbrm* command line switch, and change *automaticbind* to YES.

To configure binding for serialized profiles, there are just three steps:

1. Set the appropriate arguments for *sqlj*.

There are no command line arguments to the sqlj translator that are unique to serialized profile binding. However, the arguments set for this particular example are shown.

```
<!-- specify global property arguments below for sqlj processing -->
<property name="sqlj.arg"
value="-compile=false -status -linemap=no"/>
```

Argument	Description
compile=false	Setting this option to false prevents the sqlj translator from automatically compiling the Java source it produces. SCLM Developer Toolkit uses the generated source in its own build process, so it is recommended you always set this option to false.
linemap=no	Specifies whether line numbers in Java exceptions match line numbers in the SQLJ source files (the .sqlj file) or line numbers in the Java source file that is generated by the SQLJ translator files (the .java file). This requires a .class file, so must be set to <i>no</i> when used in conjunction with <i>compile=false</i> .
status	Prints immediate status display of SQLJ processing.

2. Set the appropriate arguments for *db2sqljcustomize*.

```
<property name="db2sqljcust.arg"
Value='-automaticbind YES -onlinecheck YES' />
```

Argument	Description
automaticbind yes	When set to yes , the customizer will also perform a bind when customization is complete. When set to no the bind must be performed separately with the command <i>db2bind</i> .
onlinecheck yes	Perform online checking on the system specified by the <i>url</i> parameter. Defaults to yes if <i>url</i> is supplied, and no otherwise.

3. Configure the user script.

Set the location of your user program in BWBSQLB. This tells the build process where the find the rex script used to calculate dynamic properties.

```
<property name="db2sqljcust.userpgm" value="/u/dba/sqljcust.rex"/>
```

Appendix C. Long/short name translation table

Currently core SCLM does not support the use of code storage with file (member) names greater than eight characters.

Code such as Java and other PC client code inherently have much greater name lengths and even incorporate path information (packaging) as part of the name. This causes the need for code with named parts greater than eight characters to go through a long/short name conversion utility to enable these parts to be stored within SCLM with an eight character (or less) name length.

A long name to short name translation table stores the matching long names (real name) against the short names (as stored in SCLM). These tables are controlled and accessed by SCLM and is saved in a VSAM data set. This functionality has been introduced into SCLM with the PTF that addresses APAR OA11426 for z/OS V1.4 and later. For z/OS V1.8 and later, this PTF is not required.

The conversion algorithm follows these steps:

1. The translate prefix consists of the first two characters (uppercase) of the long program/file name (that is, the last file name after "/" character in multi-packaging format). If the first two characters are not valid as a prefix for a host member name (because they contain invalid special characters) then the prefix is "XX". Special cases, such as a single character alphabetic name (/u/test/A or /u/test/A.java), are also assigned the prefix of "XX".
2. The last six characters are numerically assigned the next numeric sequential number available in the translate table.

For example:

Long name	Short name in SCLM PDS or PDSE
com/ibm/workbench/testprogram.java	TE000001
source/plugins/Phantom/.classpath	XX000001

Technical summary of the SCLM Translate program

SCLM program FLMLSTRN was created to read and update the VSAM translation table. SCLM Developer Toolkit uses this program to read and update correlating long names and short names.

The VSAM file used to store the translation table is a variable length KSDS with an alternative index and path defined. A sample job is provided in SCLM to allocate this VSAM file.

The internal structure of the VSAM cluster is:

```
1 ls_record,  
  3 ls_short_name  char(08),  
  3 ls_lngname_key char(50),  
  3 ls_long_name   char(1024);
```

Sample JCL to allocate the Long/Short name translation VSAM file can be seen in "Step 6: Configure long/short name table VSAM file" on page 21.

Technical summary of the SCLM Translate program

Note: The following technical information about SCLM Translate table function calls is supplied as information only and is not required to enable any SCLM Developer Toolkit functionality.

The program FLMLSTRN is invoked with the ISPF SELECT service with one of the parameters listed in Table 11.

Syntax

```
"SELECT PGM(FLMLSTRN) PARM(keyword)"
```

Invocation example:

```
"SELECT PGM(FLMLSTRN) PARM(TRANSLATE)"
```

Table 11. Long/Short name translation parameters

Keyword Record	Processing	Description
FINDLONG	Single	Find a long name for a given short name
FINDSHORT	Single	Find a short name for a given long name
TRANSLATE	Single	Find a short name if it exists or allocate a new short name if it does not exist
MIGRATE	Multiple	Search for multiple long names
IMPORT	Multiple	Search for multiple short names

Single long/short name record processing

FINDLONG Processing

- The VSAM cluster allocated to DD LSTRANS is opened in read mode.
- The short name is retrieved from the ISPF variable FLMLSSHR and this short name is used to read the VSAM file.
- If the record is not found a message is returned via the ISPF variable FLMLSERR stating that the long name was not found.
- If the long name was found it is returned in the ISPF variable FLMLSLNG.
- The VSAM cluster is closed.

FINDSHORT Processing

- The VSAM Path allocated to DD LSTRNPTH is opened in read mode.
- The long name is retrieved from the ISPF variable FLMLSLNG.
- The last 50 bytes of the long name is used to read the path.
- If a record is not returned a message is returned via the ISPF variable FLMLSERR stating that the short name was not found.
- If a record is returned the long name in the VSAM record is checked against the long name in the ISPF variable FLMLSLNG.
- If it doesn't match the VSAM records are read and compared until the ls_lngname_key doesn't match or the long name is found.

Note: The ls_lngname_key allows duplicates as it is possible to have a VSAM record with the same ls_lngname_key but different long name.

- If the short name was found it is returned in the ISPF variable FLMLSSHR.
- The VSAM path is closed.

TRANSLATE Processing

The processing is the same as for FINDSHORT.

- If the short name is found no further processing is performed.
- If the short name is not found the VSAM cluster allocated to DD LSTRANS is opened in update mode.
- The file name is determined by finding the last '/' or '\' in the long name.
- The first 2 bytes of the file name are used to look up the VSAM file prefix record which contains a number.
- The file prefix and number will be used to generate the short name (for example, PR000123).
- The short name generated (PR000123) is used to check VSAM file to determine if the short name is being used.
- If it is the prefix number is incremented and short name again checked.
- This process continues until we find a short name that is not being used.
- The prefix record is updated and then the new translate record is added.
- The short name is returned in the ISPF variable FLMLSSHR.
- The VSAM cluster is closed.

Multiple long/short name record processing

MIGRATE and IMPORT are functions that were introduced to improve performance with large numbers of long names being translated (MIGRATE) or large numbers of short names being searched for (IMPORT).

Both functions, "MIGRATE" and "IMPORT", read a variable blocked sequential file with LRECL=1036 which is allocated as DD LSTRNPRC.

Before invocation this file will contain the short names or long names depending on the function called and in the correct format and column.

After invocation LSTRNPRC will contain both the short name and correlating long name.

The format of the file is:

```
1 pr_record,
  3 pr_short_name  char(08),
  3 pr_long_name   char(1024);
```

IMPORT processing

- The VSAM cluster allocated to DD LSTRANS is opened in read mode and the processing file allocated to DD LSTRNPRC is opened for update.
- For each of the records on the processing file, the short name is used to read the VSAM translation file. If a record is found the processing file is updated with the long name.
- The VSAM cluster/processing files are closed.

MIGRATE processing

- The VSAM cluster allocated to DD LSTRANS is opened in read mode and the processing file allocated to DD LSTRNPRC is opened for update.
- For each of the records on the processing file, the long name is used to read the VSAM file. If a record is found the processing file record is updated with its corresponding short name otherwise LSTRANS is opened in update mode to add new long name/short name entries and the new short name generated is written back to the LSTRNPRC file.
- The VSAM cluster/processing files are closed.

Here is some sample REXX to invoke the long/short name translation process:

```
/* REXX *****/
/* Sample to translate long name to a short name */
/*****/
Address TSO
"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"
/* Create short name for long name com/ibm/phantom.txt */
FLMLSLNG = "com/ibm/phantom.txt"
Address ISPEXEC "VPUT (FLMLSLNG) PROFILE"
Address ISPEXEC "SELECT PGM(FLMLSTRN) PARM(TRANSLATE)"
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"
    Say " Shortname = " FLMLSSHR
    Say " Longname = " FLMLSLNG
End
Address TSO
"FREE FI(LSTRANS)"
"FREE FI(LSTRNPTH)"
```

Figure 35. Sample REXX for Translate module invocation

Bibliography

Publications referred to in this document:

HTTP Server Planning, Installing and Using,
SC31-8690

*P/390, R/390, S/390 Integrated Server: OS/390
New User's Cookbook*, SG24-4757

*z/OS ISPF Software Configuration and Library
Manager Project Manager's and Developer's
Guide*, SC34-4817

*z/OS ISPF Software Configuration and Library
Manager Reference*, SC34-4818

z/OS Information Roadmap, SA22-7500

z/OS UNIX System Services Command Reference,
GA22-7802

z/OS UNIX System Services Messages and Codes,
GA22-7807

z/OS UNIX System Services Planning,
GA22-7800

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 about 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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one), and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie New York 12601-5400
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

Notices

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 can 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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

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

AIX
CICS
DB2
Domino
eServer
IBM
Intel
Library Reader
Lotus
MVS
OS/390
Passport Advantage
RACF
Redbooks
S/390
WebSphere
z/OS
zSeries

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

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

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

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

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

Glossary

A

accounting record. An SCLM control data record containing statistical, historical, and dependency information for a member under SCLM control.

alternate project definition. A project definition that provides a version of the project environment which differs from the default project definition.

architecture definition. A means of organizing components of an application into conceptual units. It is SCLM's method of defining an application's configuration. It describes how the components of an application fit together and how they are used to drive both the build and promote functions. Architecture definitions are used to group components into applications, sub-applications, and load modules.

architecture member. Defines an individual software component, which may be a collection of other architecture members, by specifying its relationship to other software components of an application.

authcode. An identifier used by SCLM to control authority to update and promote members within a hierarchy. These codes can be used to allow concurrent development without the risk of module collisions (overlaid changes).

authorization code. See **authcode**.

B

bidirectional (bi-di). Pertaining to scripts such as Arabic and Hebrew that generally run from right to left, except for numbers, which run from left to right.

bidirectional attribute. Text type, text orientation, numeric swapping, and symmetric swapping.

build. The process of transforming inputs into outputs through the invocation of translators specified in the language definition. Compilers, preprocessors, and linkage editors are examples of translators that might be invoked at build time.

build map. Internal data record containing a complete analysis of the database at the time of the build; it includes the names of all referenced members and the last change date and version number of each member.

C

change code. An 8-character identifier used to indicate the reason for an update or modification to a member controlled by SCLM.

component. Any input or output member associated with an application, which together make up all or a member of the application.

container. In J2EE, an entity that provides life-cycle management, security, deployment, and runtime services to components. Each type of container (EJB, Web, JSP, servlet, applet, and application client) also provides component-specific services.

D

development group. Groups that are in the lowest level of the hierarchy. These groups represent end-nodes with no other lower groups promoting into them.

E

EAR. See **Enterprise Archive**.

EJB. See **Enterprise Java Beans**.

Enterprise Archive. A specialized type of JAR file, defined by the J2EE standard, used to deploy J2EE applications to J2EE application servers. An EAR file contains EJB components, a deployment descriptor, and Web archive (WAR) files for individual Web applications.

Enterprise JavaBeans. JavaBeans are reusable objects, like subroutines. EJBs take this a step further and are designed to be platform independent.

G

group. A set of project data sets with the same middle-level qualifier in the SCLM logical naming convention.

H

hierarchy. The organization of groups in a ranked order, where each group is subordinate to the one above it.

I

IDE project. A project developed in the Eclipse IDE.

J

J2EE. Java 2 Platform, Enterprise Edition. It defines the standard for developing component-based multi-tier enterprise applications.

J2EE project. A project developed in the Eclipse IDE specifically related to Java/J2EE.

JAR. See **Java Archive**.

Java Enterprise. A file format for storing information for or about Java programs.

L

language definition. Specifies the set of translators to be run for SCLM functions PARSE, VERIFY, BUILD, COPY, and PURGE. A language definition is composed of one FLMLANGL macro followed by an FLMTRNSL macro for each translator to be run for members of SCLM libraries whose language attribute matches the value of the LANG keyword in the FLMLANGL macro.

level. A given tier of the hierarchy, made up of groups, of equivalent rank.

library. In z/OS, a partitioned data set.

lock. When a user locks a member, only that user can change it. All other users are unable to change that member until the member is promoted or unlocked. When a member is locked, an authorization code is specified. If two users need to change a part, two different authorization codes can be used.

M

member. The discreet element of an SCLM database, representing a single data type of a software component.

migrate. Registering software components in SCLM: this includes identifying the component language, and possibly the change code and authorization code.

migration. The process of introducing members into SCLM control. Migration locks the member, parses it according to the requested language, and stores the information in the accounting database. The migration utility can be used to enter a large number of members into a project's database, such as during conversion to SCLM.

P

perspective. A group of views that show various aspects of the resources in the workbench. The workbench user can switch perspectives, depending on the task at hand, and customize the layout of views and editors within the perspective.

project. A collection of libraries representing an integrated SCLM database, under a single high-level qualifier.

project administrator. The person who maintains an SCLM project.

project definition. Defines the SCLM library structure, project control information, and language definitions. A project definition is a load module used by SCLM at run time. The source code for a project definition is composed of macros.

project definition data. Project definitions and language definitions which are used to create and control an SCLM project.

promote. The process of moving an application or its components from one level in the project hierarchy to the next. Promotion out of a development group removes the lock on editable members that were successfully promoted.

S

SCLM administrator. The person who maintains an SCLM project.

scope. The set of members (including architecture definitions) that will be processed (for example verified, copied, compiled, or purged) by build or promote.

service. An SCLM function available via a command or programming interface.

T

translator. A load module, CLIST, or REXX program that receives control from SCLM for execution. The name of the translator is specified as the value of the COMPILE keyword for the FLMTRNSL macro. Examples of translators are compilers, assemblers, linkage editors, text processors, DB2® preprocessors, CICS® preprocessors, utilities, and customer tools.

type. The third qualifier of the SCLM naming convention for project-partitioned data sets. Typically identifies the kind of data maintained for a project hierarchy. Examples of types are SOURCE, OBJECT, and LOAD.

U

unlock. To make a member (formerly locked out) available for updating (usually associated with promote).

unlock service. Removes the restriction (unlocks) on a member to a development group.

V

version. A copy of a member as it existed at a previous point in time.

versioning. A function that enables the retrieval of a version of a member. This is useful for backing out changes.

W

WAR. See **Web Archive**.

Web Archive. A file format for storing information for or about Web type applications.

Index

Special characters

- \$GLOBAL member
 - CLASSPATH_JARS parameter 42
 - creating different settings 63
 - providing information to JAVA/J2EE
 - build processes 62
 - set at highest level for all groups 63

A

- Ant
 - customizing 23
 - installing 23
 - testing initialization 25
 - translating ASCII to EBCDIC 23
 - Web address 23
- ANT_BIN global variable
 - z/OS UNIX System Services file
 - system directory path 62
- ARCHDEF 83
- ARCHDEF SCLM type 44
- ASCII
 - See* ASCII/EBCDIC translation
- ASCII codepage 68
- ASCII storage options 61
- ASCII/EBCDIC conversion
 - See* ASCII/EBCDIC translation
- ASCII/EBCDIC translation
 - See also* IBM-1047
 - See also* ISO8859-1
 - code pages in use 9
 - converting Ant text files and scripts 23
 - files stored in ASCII or EBCDIC 61
 - language translators 61
 - non-standard 9
 - non-standard codepage
 - translation 15
 - source or components requiring 42
 - translation of part 13
 - translation to host 10
- authentication
 - public key 60

B

- BATCHBUILDn option 68
- BATCHMIGRATEn option 68
- BATCHPROMOTE option 68
- BIDIPROP Language 69
- BIDIPROP overrides 71
- BINARY language translator 61
- build rule format 74
- build skeletons 53
- Build/Promote/Deploy
 - process flow 73
 - security flag 73
- BUILDAPPROVER option 67
- Builds
 - configuration files 77

- Builds (*continued*)
 - CRON-initiated 77
- BUILDSECURITY option 68
- BWBCPANT
 - Ant install member 23
- BWBCRON1 sample member 77
- BWBCRONB sample member 77
- BWBCRONP sample member 77
- BWBGLOB sample 62
- BWBHTTPC member 6
- BWBHTTPE member 6
- BWBINST1 install JCL
 - tasks performed by 6
- BWBINST1 sample configuration file 14
- BWBPROJ member
 - holding sample Project config file 67
- BWBSITE sample 65
- BWBTRAN1 language translator 39
- BWBTRAN2 language translator 39
- BWBTRAN3 language translator 39
- BWBTRANJ language translator 39
- BWBTRANT
 - sample translation script 23

C

- CCODE option 67
- CGI_DTWORK global variable
 - install home directory 63
- classpath dependencies 42
- CLASSPATH_JARS global variable
 - z/OS UNIX System Services file
 - system classpath directory 63
- CODEPAGE keyword 9
- codepages
 - See* ASCII/EBCDIC conversion
- concepts
 - ARCHDEF 83
 - file naming 81
 - Language 82
 - project structure 82
 - properties 82
 - Type 81
- CONFIG directory, creating 6
- CONFIG/PROJECT directory
 - See* PROJECT directory
- configuration considerations 4
- configuration files
 - customizing 8
 - HTTP server 14
 - Project 67
 - Promotes 77
 - TRANSLATE 13
- conversion
 - See* ASCII/EBCDIC translation
- creating project.conf 66
- CRON
 - Build job execution 78
 - Build job samples 78
 - CRON-initiated Builds 77
 - CRON-initiated Promotes 77

- CRONTAB file 78
- customization
 - Ant 23
 - checking 25, 30
 - ISPF configuration file 8
 - ISPF.conf 8

D

- DEBUG_MODE global variable
 - VSAM file 63
- dependencies
 - classpath 42
- deploy rule format 74
- deployment
 - options 60
 - SCLM Developer Toolkit 58
 - SCLM to Unix System Services 59
 - secure 60
 - WebSphere Application Server (WAS) 59
- deployment options 60
- DEPLOYSECURITY option 68
- directives
 - Exec
 - in httpd.conf file 14
 - Pass
 - in httpd.conf file 14
 - PASS
 - check if connection has failed 27
- directory 64
 - CONFIG 6
 - LOGS 6
 - PROJECT 6
 - WORKAREA 6

E

- EBCDIC
 - See* ASCII/EBCDIC translation
- EBCDIC codepage 68
- EBCDIC storage options 61
- Eclipse-based client
 - installing 33
 - prerequisites 33
- Exec directives, in httpd.conf file 14

F

- file naming 81
- file, CRONTAB 78
- FINDLONG processing 98
- FINDSHORT processing 98
- FLMLSTRN, SCLM translate
 - program 97
- FOREGROUNDBUILD option 68
- FOREGROUNDPROMOTE option 68
- function
 - IMPORT 99
 - MIGRATE 99

G

global variables
ANT_BIN 62
CGI_DTWORK 63
CLASSPATH_JARS 63
DEBUG_MODE 63
JAVA_BIN 62
TRANTABLE 63

H

Hierarchical File System
 See z/OS UNIX System Services file system
HTTP installation 25
HTTP server
 check if connection has failed 27
 configuring 13
 customizing existing server 16
 default port configuration 14
 environment file
 customization 15
 location 14
JCL/STARTED TASK
 customization 15
logon prompt 25
owning user ID 16
port 14
sample configuration files 6, 14
starting 16
testing connection to 29
trace 16
httpd.conf
 copied into CONFIG directory 6
 customizing 14
 sample HTTP configuration file 14
httpd.env
 copied into CONFIG directory 6
 customizing 15

I

IBM z/OS HTTP server
 See HTTP server
IBM-1047
 default EBCDIC codepage 9
IMPORT function
 improve long/short name record processing 99
IMPORT processing 99
installation
 Ant 23
 checking 25, 30
 overview xiii
installation directory
 recommended for configuration files 6
ISO8859-1
 default ASCII codepage 9
ISPF configuration file
 customizing 8
ISPF.conf
 customizing 8
 location 8

IVP

running to check installation and customization 25, 30

J

J2EE
 See also JAVA/J2EE
 sample language translator 39
 sample scripts 52
J2EE ARCHDEF
 samples 47
J2EE projects, mapping 54
J2EEANT SCLM language translator 43
J2EEBIN
 language translator 61
 SCLM language translator 42
J2EEBLD SCLM type 43
J2EEEAR SCLM type 45
J2EEJAR SCLM type 44
J2EELIST SCLM type 44
J2EEOBJ SCLM language translator 43
J2EEPART language translator 61
J2EEPART SCLM language translator 42
J2EEWAR SCLM type 44
Java
 See also JAVA/J2EE
 level required 4
JAVA language translator 61
JAVA SCLM language translator 42
JAVA_BIN
 z/OS UNIX System Services file system directory path 62
JAVA_HOME environment variable 23
JAVA/J2EE
 accessing language translator modules 39
 Ant installation and customization 23
 Ant XML build skeletons 53
 build 41
 build summary 40
 language translators 39
 sample project definition 39
 SCLM member formats 45
 SCLM types to support 43
 use of WORKAREA 6
JAVA/J2EE concepts 83
Java/J2EE SCLM type 45
JAVABIN language translator 61
JAVABIN SCLM language translator 42
JAVACLAS SCLM type 44
JAVALIST SCLM type 44

K

key authentication 60
keyword
 LANG 9, 11
 TRANLANG 10

L

Language 82
language translators 61
 BINARY 61

language translators (*continued*)

BWBTRAN1 39
BWBTRAN2 39
BWBTRAN3 39
BWBTRANJ 39
J2EEPART 61
JAVA/J2EE support 39
SCLM, J2EEOBJ 43
SCLM, J2EEPART 42
TEXT 61
license inquiry 103
LOGS directory
 creating 6
 read/write access 16
long name
 See long/short name
long/short filename
 See long/short name
long/short name
 configuring table VSAM file 21
 conversion utility 97
 multiple record processing 99
 single record processing 98
 translation parameters 98
 translation table 97
long/short name translation
 REXX sample 100
LONGLANG keyword 11
LONGLANG=SCLM Language 69

M

Mapping J2EE projects 54
member
 BWBHTTTPC 6
 BWBPROJ
 holding sample Project config file 67
MIGRATE function
 improve long/short name record processing 99
MIGRATE processing 100

N

naming, files 81
NOLONGLANG=SCLM Language 69
NOTRANLANG=SCLM Language 69

O

option
 BATCHBUILDn 68
 BATCHMIGRATEn 68
 BATCHPROMOTEn option 68
 BUILDAPPROVER 67
 BUILDSECURITY 68
 CCODE 67
 FOREGROUNDBUILD 68
 FOREGROUNDPROMOTE 68
options
 ASCII storage 61
 definition 67
 EBCDIC storage 61
 in sample project specific file 67
 in sample SITE specific file 65

- options (*continued*)
 - project 63
 - PROMOTESECURITY 68
 - SITE 63
 - that can be configured 63
- overrides, TRANSLATE.conf 69

P

- PASS directive, check if connection has failed 27
- Pass directives, in httpd.conf file 14
- PATH, requirements 77
- port
 - See* port number
- port number
 - changing 14
 - default HTTP server 14
 - in httpd.conf file 14
 - used in location URL 25
 - used to check connection to HTTP server 29
- process flow
 - Build/Promote/Deploy 73
- processing
 - FINDLONG 98
 - FINDSHORT 98
 - IMPORT 99
 - MIGRATE 100
 - TRANSLATE 99
- PROJECT 64
- Project config file, sample 67
- PROJECT directory
 - creating 6
 - storage area for options 64
- project structure 82
- project.conf, creating 66
- promote rule format 74
- PROMOTEAPPROVER option 67
- Promotes
 - configuration files 77
 - CRON-initiated 77
- PROMOTESECURITY option 68
- properties 82
- public key authentication 60

R

- RACF
 - considerations 15
 - create OMVS segment for user ID 15
 - individual user requirements 4
- Recommendations for mapping J2EE projects (and others) to SCLM 57
- Remote Systems Explorer,
 - configuring 18
- requirements
 - RACF 4
 - STEPLIB 77
- requirements, PATH 77
- Resource Access Control Facility
 - See* RACF
- REXX
 - CRON build exec sample 78
 - host interface call sample 77

- REXX (*continued*)
 - long/short name translation
 - sample 100
 - required software 16
 - set up Build parameter input string
 - sample 77
 - set up Promote parameter input string
 - sample 77
- RSE installation, checking 30
- rule format
 - build 74
 - deploy 74
 - promote 74
- rules
 - SAF/RACF
 - BUILD 75
 - DEPLOY 75
 - PROFILE 75
 - PROMOTE 75

S

- sample member
 - BWBCRON1 77
 - BWBCRONB 77
 - BWBCRONP 77
- samples
 - BWBGLOB 62
 - BWBSITE 65
 - CRON-initiated Builds and Promotes 77
- SBWBSAMP library
 - BWBGLOB sample 62
 - BWBPROJ sample 67
 - BWBSITE sample 65
 - contents 77
 - JAVA/J2EE Ant XML build skeletons 53
 - source of sample translators 39
- SCLM
 - concepts 81
 - overview 81
- SCLM administrator
 - SCLM customization 39
- SCLM concepts 81
- SCLM customization
 - by SCLM administrator 39
- SCLM Developer Toolkit
 - customizing 3
 - installing 3
- SCLM file formats
 - \$GLOBAL 45
 - J2EE Ant 49
 - J2EE ARCHDEF 46
- SCLM language definitions 42
- SCLM member formats 45
- SCLM overview 81
- SCLM security 73
- SCLM Translate program
 - technical summary 97
- SCLM type
 - ARCHDEF 44
 - J2EEBLD 43
 - J2EEEAR 45
 - J2EEJAR 44
 - J2EELIST 44
 - J2EEWAR 44

- SCLM types 43
- security flag 73
- security rules
 - surrogate user ID 73
- setup JCL, running 6
- short name
 - See* long/short name
- SITE.conf
 - creating 65
 - site configuration file 65
- skeletons, build 53
- SMP/E installation xiv
- software requirements 4
- STEPLIB requirements 77
- storage options
 - See* options
- surrogate user ID
 - security rules 73

T

- TEXT language translator 61
- TRANLANG keyword 10
- TRANLANG=SCLM Language 68
- TRANSLATE configuration file,
 - example 13
- TRANSLATE processing 99
- TRANSLATE.conf
 - contents 9
 - creating 6
 - customizing 9
 - location 8
 - overrides 69
- translation
 - code pages in use 9
 - converting Ant text files and scripts 23
 - files stored in ASCII or EBCDIC 61
 - language translators 61
 - long/short file name 97
 - non-standard codepage translation 15
 - source or components requiring 42
 - translation of part 13
 - translation to host 10
- TRANSTABLE global variable
 - VSAM file 63
- Type 81

U

- UNIX System Services
 - initiate builds and promotes via CRON 77
 - pipe symbol 78
- USS
 - See* UNIX System Services

W

- WORKAREA directory
 - creating 6
 - purpose 6
 - read/write access 6

Z

z/OS

See also UNIX System Services

HTTP server

See HTTP server

software requirements 4

UNIX System Services

JAVA_HOME environment

variable 23

UNIX System Services file system

set up Build and Promote

configuration files 77

variables 78

UNIX System Services file system

mount point

check if connection has failed 27

Readers' Comments — We'd Like to Hear from You

SCLM Developer Toolkit
Installation and Customization Guide
Version 3.1

Publication No. SC23-8504-01

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:

- Send your comments to the address on the reverse side of this form.

If you would like a response from IBM, please fill in the following information:

Name

Address

Company or Organization

Phone No.

E-mail address



Cut or Fold
Along Line

Fold and Tape

Please do not staple

Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Information Development
Department G71A / Bldg. 503
555 Bailey Avenue
Research Triangle Park, NC
U.S.A 27709-2195



Fold and Tape

Please do not staple

Fold and Tape

Cut or Fold
Along Line



Program Number: 5655-S72

Printed in USA

SC23-8504-01

