Rational. IBM Rational Developer for System z

Version 7.6

SCLM Developer Toolkit Administrator's Guide

SC23-9801-01

Rational. IBM Rational Developer for System z

IBM

Version 7.6

SCLM Developer Toolkit Administrator's Guide

SC23-9801-01

Note
Before using this document, read the general information under "Documentation notices for IBM Rational Developer for System z" on page 125.

First Edition (September 2009)

This edition applies to IBM Rational Developer for System z Version 7.6, (program number 5724-T07) and to all subsequent releases and modifications until otherwise indicated in new editions.

You can also order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. You can send your comments by mail to the following address:

IBM Corporation
Attn: Information Development Department 53NA
Building 501 P.O. Box 12195
Research Triangle Park NC 27709-2195
USA

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

The IBM Rational Developer for System z Web site is at

http://www-306.ibm.com/software/awdtools/rdz/

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

Contents

Tables

About this document

This document discusses the configuration of IBM® Software Configuration and Library Manager (SCLM) required for the SCLM Developer Toolkit function of IBM Rational Developer for System z.

From here on, the following names are used in this manual:

- *IBM Software Configuration and Library Manager* is called *SCLM*.
- *IBM Rational® Developer for System z®* is called *Developer for System z*.
- The *SCLM Developer Toolkit* function of *IBM Rational Developer for System z* is called *SCLM Developer Toolkit*, sometimes abbreviated to *Developer Toolkit* or *SCLMDT*.

Who should read this book

This document contains information for the administrator of SCLM projects that will be used with 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 be familiar with z/OS UNIX System Services, REXX™ script, the Java Compiler, and SCLM project and language definitions.

Chapter 1. Product installation

This publication does not cover the implementation and loading of the SCLM product, which is shipped with the z/OS operating system. Neither does it cover the installation and configuration of SCLM Developer Toolkit itself, which is a function of Rational Developer for System z.

Refer to *ISPF Software Configuration and Library Manager Project Manager's and Developer's Guide* (SC34-4817) and *Rational Developer for System z Host Configuration Guide* (SC23-7658) for more information on these tasks.

To complete the customization and project definition tasks, the SCLM administrator needs to know several Developer for System z customizable values, as described in Table 1. Contact the z/OS system programmer responsible for installing and customizing Developer for System z for more information.

Table 1. SCLM administrator checklist

Description	Default value	Where to find the answer	Value
Developer for System z sample library	FEK.SFEKSAMP	SMP/E installation	
Developer for System z sample directory	/usr/lpp/rdz/samples	SMP/E installation	
Java bin directory	/usr/lpp/java/J5.0/bin	rsed.envvars - $JAVA_HOME/bin	
Ant bin directory	/usr/lpp/Ant/apache-ant-1.7.1/bin	rsed.envvars - $ANT_HOME/bin	
WORKAREA home directory	/var/rdz	rsed.envvars - $_CMDSERV_CONF_HOME	
SCLMDT project configuration home directory	/etc/rdz/sclmdt	rsed.envvars	
Long/short name translation VSAM	FEK.#CUST.LSTRANS.FILE	rsed.envvars - $_SCLMDT_TRANTABLE	

Chapter 2. SCLM customization for SCLM Developer Toolkit

This chapter looks at how the SCLM administrator can customize SCLM for use by SCLM Developer Toolkit.

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 through 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.

The following sample shows the JAR code:

```
*
* 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 1. Sample Jar application (JAR) ARCHDEF

The following example 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="BWBJAVAA"/> (A)
<property name="SCLM_BLDMAP" 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 2. 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, and so on, 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, and so on, 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.

Language translators for JAVA/J2EE support

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

Table 2. SCLM Developer Toolkit translators

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

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 or compiled in the Project Definition.

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

SCLM language definitions

The sample translators define the following languages:

J2EEANT

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 through the SINC keyword in the ARCHDEF.

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.

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.

J2EEOBJ

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.

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, or definition tables). Optionally language definition of TEXT can be used.

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 code page 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. Java compile output is stored in type JAVALIST.

  Any classpath dependencies can be satisfied by storing dependent JARs in the classpath directory specified in $GLOBAL member parameter `CLASSPATH_JARS`. For more information see "$GLOBAL" on page 22.
- 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.

JAVABIN
Language type that is similar to Java and used when storing Java source as ASCII in SCLM.

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.

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

SCLM data sets 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.

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

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

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

SCLM type	RECFM	LRECL
ARCHDEF	FB	80
J2EEBLD	FB	256
JAVALIST	VB	255
J2EELIST	VB	255
DBRMLIB	VB	256
JAVACLAS	VB	256
J2EEEAR	VB	256
J2EEJAR	VB	256
J2EEWAR	VB	256
SQLJSER	VB	256
Additional source dataset types for Java/J2EE	VB	1024

ARCHDEF

The ARCHDEF type 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.

J2EEBLD

The J2EEBLD type is required for Java and J2EE build and deploy processes, and contains the following:

- J2EEBLD build scripts used to drive the Ant build and deploy process.
- Java and J2EE ANTXML scripts to be invoked for builds and deploys.
- $GLOBAL, which specifies the default properties for the SCLM project for JAVA/J2EE build processing. See "$GLOBAL format" on page 9 and "$GLOBAL" on page 22 for more information.

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 15.

JAVALIST

The JAVALIST type is required for the Java build process and contains listing outputs from Java builds.

J2EELIST

The J2EELIST type is required for the J2EE build process and contains listing outputs from J2EE builds.

DBRMLIB

Technically a DB2® type.

DBRMLIB is required for SQLJ support and stores the database request modules.

JAVACLAS

The JAVACLAS type is required for both Java and J2EE build processes and contains output class files from builds associated with the JAVA, J2EEANT language definitions.

J2EEEAR

The J2EEEAR type is required for the J2EE build process and contains EAR output from builds associated with the J2EEANT language definition.

J2EEJAR

The J2EEJAR type is required for JAVA/J2EE builds and contains JAR output from builds associated with the J2EEANT language definition.

J2EEWAR

The J2EEWAR type is required for the J2EE build process and contains WAR output from builds associated with the J2EEANT language definition.

SQLJSER

The SQLJSER type is required for the J2EE/SQLJ build process and stores SQLJ serialized profiles.

<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 see "Mapping J2EE projects to SCLM" on page 16.

SCLM member formats

This section describes SCLM member formats.

$GLOBAL format

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" on page 22.

See the following code 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="_SCLMDT_CONF_HOME" value="/etc/rdz/sclmdt/CONFIG"/>
<property name="_SCLMDT_WORK_HOME" value="/var/rdz/WORKAREA"/>
<property name="CLASSPATH_JARS" value="/var/rdz/CLASSPATH"/>
```

Figure 3. $GLOBAL - SCLMDT environment variables

J2EE ARCHDEF format

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

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

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

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

The following example shows the JAR code:

```
*
* 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                         *
 INCLD V2000003 V2TEST    * V2InnerClass.java                        *
*
* Build script and generated outputs
*
 SINC   V2JARB1 J2EEBLD   * J2EE JAR Build script                    *
 OUT1   *       J2EEJAR   * V2TEST.jar                               *
 LIST   *       J2EELIST
```

Figure 4. Sample Jar application (JAR) ARCHDEF

The following example shows the WAR code:

```
*
* Initially generated on 5 Sep 2006 by SCLM DT V2
*
 LKED  J2EEOBJ            * 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.htpl                  *
 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 5. Sample Web application (WAR) ARCHDEF

The following example shows the EJB code:

```
LKED  J2EEOBJ
*
 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 6. Sample EJB Application (EJB) ARCHDEF

The following example shows the EAR code:

```
 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 7. Sample EAR Application (EAR) ARCHDEF

J2EE Ant Build Script format

The J2EE Ant Build Script format is of type J2EEBLD and language J2EEANT. It can be any name up to eight 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 scripts, variables are the same except for using EAR_NAME and JAR_NAME instead of WAR_NAME.

See the following example, which shows the sample build script:

```
<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 8. 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 3.

Table 3. 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. This will be set to the following values: • 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.

Table 3. Customer-defined variables (continued)

Variable	Description
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, as follows: `<property name="CLASSPATH_JARS_FILES" value="V2J4.jar,V2J3.jar" />`
ENCODING	Either ASCII or EBCDIC code page for JAVA This is the code page JAVA source is stored on the z/OS host. For example: • For ASCII JAVA standard code page should be ISO8859-1 • For EBCDIC JAVA standard code page 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.

  This directory can be updated with CLASSPATH files through 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, the listed JAR/class files are retrieved from SCLM or if not located in SCLM, the directory referenced by CLASSPATH_JARS is searched for retrieval. If this keyword is used, only those files listed are used in the classpath.

J2EE sample scripts

The following example shows the 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="BWBJAVAA"/>
<property name="SCLM_BLDMAP" value="YES"/>
<property name="JAR_FILE_NAME" value="V2TEST.jar"/>
<property name="CLASSPATH_JARS" value="/var/rdz/CLASSPATH"/>
<property name="ENCODING" value="IBM-1047"/>
</ANTXML>
```

Figure 9. J2EE Build script JAR sample

The following example shows the WAR script:

```
<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="BWBWEBA"/>
<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/rdz/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"/>
<fileset dir="${CLASSPATH_JARS}" includes="**/*.jar, **/*.zip"/>
</path>
</ANTXML>
```

Figure 10. J2EE Build script WAR sample

The following example shows the EJB script:

```
<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/rdz/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 11. J2EE Build script EJB sample

The following example shows the EAR script:

```
<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/rdz/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 12. J2EE Build script EAR sample

JAVA/J2EE Ant XML build skeletons

This section lists sample Ant build skeletons which are provided in the FEK.SFEKSAMV 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.

The supplied sample J2EE build skeletons for building a simple JAR, SQLJ project, EJB JAR, WAR, or 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 through the SCLM 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.

BWBJAVAA

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 or build an EJB project which would usually create an EJB JAR which has a structure determined by a specified ARCHDEF.

BWBWEBA

Sample Ant XML J2EE WEB build skeleton

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

BWBC9DTJ

Cloud 9 to SCLM DT conversion sample.

BWBDEPJ1

Sample to update SQLJ .ser files within a JAR at deployment time using db2sqljcustomize.

BWBDEPJ2

Sample for db2sqljcustomize where the property longname will copy the specified JAR from the indicated group and type locations in SCLM to the destination directory specified by "dest".

BWBDEPJ3

This sample routine will customize the .ser files contained within selected JAR files for deployment using db2sqljcustomize.

BWBTRANX

Sample SCLM build translator for SYSXML build error messaging for COBOL.

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.

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.

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 is 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. While 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 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.

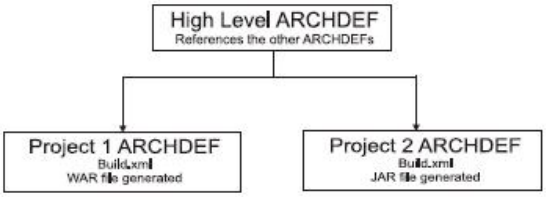


Figure 13. SCLM build hierarchy

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 refers to 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 the following:

- 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 refer to 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.

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

1. Include the reference to the JAR through 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.
   - 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 referring to the JAR will be resolved. However, at runtime, the JAR must be available in a PATH statement.
   - Refer to 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 plugin 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 `FEK.SFEKSAMV` 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	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 a 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 refers to 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 created as an ASCII file in a z/OS UNIX directory before the deployment process can be invoked.

Developer for System z customization provides a sample (ASCII) JACL script as `/etc/rdz/sclmdt/CONFIG/scripts/deploy.jacl` (where `/etc/SCLMDT` is the default `etc` directory for SCLM Developer Toolkit).

Additional JACL examples can be found in the WebSphere Application Server (WAS) documentation.

The directory locations of the wsadmin tool (`wsadmin.sh`) and the JACL script (`deploy.jacl`) can be configured in the preference page under **Team > SCLM Preferences > Build Script Options**. The SCLMDT client 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.

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 option provides a means to copy deployable objects to a remote host by using 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 using 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`.

Note: The IBM Ported Tools for z/OS will need to be ordered, installed and configured to support secure deployment. Refer to *IBM Rational Developer for System z Host Planning Guide* (GI11-8296) to learn how Ported Tools can be obtained. The installation and customization of this product is not described in this manual.

IBM Ported Tools for z/OS provides the following:

- 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 want to provide deployment capabilities.

For instructions on how to set up automated key-based authentication using ssh-agent and ssh-add, refer to *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 45.

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 refer to 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 code page. 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 or 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

Table 4. SCLM Language Translators and ASCII/EBCDIC

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`.

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 or import and build performance, as files require no translation. This is only suitable if editing in ISPF/SCLM is not required.

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 so 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 code page 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 Java command with the `ENCODING=ISO8859-1` to compile the ASCII source. The default ENCODING code page is the EBCDIC code page `IBM-1047`.

$GLOBAL

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 is 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: When using the ant.conf configuration file, then the `JAVA_HOME` specified will override any `JAVA_BIN` specified in `$GLOBAL` for Java/J2EE project compiles. This is not true if Ant configuration is done in `rsed.envvars`, which is what is described in *Rational Developer for System z Host Configuration Guide* (SC23-7658).

A sample member `BWBGLOB` is provided in the `FEK.SFEKSAMV` 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 5. $GLOBAL variables

Variable	Description
ANT_BIN	z/OS UNIX System Services file system directory path of Ant runtime **Example:** `<property name="ANT_BIN" value="/usr/lpp/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="/usr/lpp/java/5.0/bin"/>`
_SCLMDT_WORK_HOME	The location of the SCLM Developer Toolkit WORKAREA directory **Example:** `<property name="_SCLMDT_WORK_HOME" value="/var/rdz"/>`
_SCLMDT_CONF_HOME	The location of the SCLM Developer Toolkit CONFIG directory **Example:** `<property name="_SCLMDT_CONF_HOME" value="/etc/rdz/sclmdt"/>`
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:** `<property name="CLASSPATH_JARS" value="/var/rdz/CLASSPATH"/>`
TRANTABLE	VSAM file containing the long/short name translations **Example:** `<property name="TRANTABLE" value="FEK.#CUST.LSTRANS.FILE"/>`
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 all group levels in the SCLM project, it is a 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, for example, at different development groups, a $GLOBAL member can be created in each of the development groups.

TRANSLATE.conf

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 VSAM.

`TRANSLATE.conf` is located in `/etc/rdz/sclmdt/CONFIG`. You can edit the file with the TSO OEDIT command.

The following example shows the TRANSLATE.conf code, which must be customized to match your system environment. Comment lines start with an asterisk (*).

```
*============================================================
* cross system sharing
TRANVRLS = NO
*============================================================
* codepage
CODEPAGE ASCII = ISO8859-1
CODEPAGE EBCDIC = IBM-1047
*============================================================
* ascii/ebcdic translation
TRANLANG JAVABIN
TRANLANG J2EEBIN
TRANLANG J2EEOBJ
TRANLANG TEXTBIN
TRANLANG BINARY
TRANLANG DOC
TRANLANG XLS
*============================================================
* long/short name translation
LONGLANG JAVA
LONGLANG SQLJ
LONGLANG J2EEPART
LONGLANG JAVABIN
LONGLANG J2EEBIN
LONGLANG J2EEOBJ
LONGLANG DOC
LONGLANG XLS
```

Figure 14. TRANSLATE.conf - SCLMDT ASCII/EBCDIC translation configuration file

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

code page ASCII
Indicates the ASCII code pages to use in translation. The default is ISO8859-1.

There must be a code page directive for both ASCII and EBCDIC for SCLM Developer Toolkit to determine how to convert files being transferred.

Code page EBCDIC
Indicates the EBCDIC code pages to use in translation The default is IBM-1047.

There must be a code page directive for both ASCII and EBCDIC for SCLM Developer Toolkit to determine how to convert files being transferred.

TRANVRLS
Indicates whether the long/short name translate VSAM data set can be shared across systems. The default is NO. The only valid values are YES and NO.

SCLM uses VSAM Record Level Sharing (RLS) to allow sharing of the VSAM data set and maintain integrity in a shared environment. The VSAM data sets must be defined with the correct STORAGECLASS for RLS use. Refer to *DFSMS(TM) Using Data Sets (SC26-7410)* for more information on RLS.

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

Note that there is no equal sign (=) in this directive to separate the TRANLANG keyword and the name of the (dummy) Language Translator.

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

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 that there is no equal sign (=) in this directive to separate the LONGLANG keyword and the name of the SCLM Language.

Note: It is possible to override values set in the TRANSLATE.conf file at a SITE and SCLM Project level, as described in "SITE and project-specific options."

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 the following:

- 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 (bidi) 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.conf file while others can be set at an SCLM project-specific level. Alternatively there can be no SITE-specific file 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 SITE.conf file in the z/OS UNIX /etc/rdz/sclmdt/CONFIG/PROJECT/ directory (where /etc/rdz/sclmdt is the default etc directory for SCLM Developer Toolkit). This directory is created during the customization of Developer for System z.

A sample SITE.conf file is provided in the /usr/lpp/rdz/samples/ directory. Copy this directory and the directives to match your needs. You can edit the file with the TSO OEDIT command. The following example shows the SITE.conf configuration file.

```
* SCLM Developer Toolkit Site Specific option
*
* 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=&SYSUID,REGION=512M
BATCHBUILD3=//*
BATCHBUILD4=//*
*
* Batch Promote default jobcard
BATCHPROMOTE1=//SCLMPROM JOB (#ACCT),'SCLM PROMOTE',CLASS=A,MSGCLASS=X,
BATCHPROMOTE2=//         NOTIFY=&SYSUID,REGION=128M
BATCHPROMOTE3=//*
BATCHPROMOTE4=//*
*
* 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 catalog searches for all SCLM projects.
*
PROJECTLISTALL=Y
```

Figure 15. 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 (not case-sensitive).

A sample project config file is provided in the `/usr/lpp/rdz/samples/` directory as file `SCLMproject.conf`. Copy this sample to the PROJECT directory, using the correct target name, and customize the directives to match your needs.

You can edit the file with the TSO OEDIT command. The following example shows the Project configuration code.

```
* SCLM Developer Toolkit Project Specific option
*
* SCM Approver processing applies to this project?
BUILDAPPROVER=BREEZE
PROMOTEAPPROVER=BREEZE
*
* Change Code entry on check-in is mandatory?
CCODE=Y
*
* Foreground or On-line builds/promotes allowed for this project?
FOREGROUNDBUILD=N
FOREGROUNDPROMOTE=N
*
* Batch Build default jobcard
BATCHBUILD1=//SCLMBILD JOB (#ACCT),'SCLM BUILD',CLASS=A,MSGCLASS=X,
BATCHBUILD2=//         NOTIFY=&SYSUID,REGION=512M
BATCHBUILD3=//*
BATCHBUILD4=//*
*
* Batch Promote default jobcard
BATCHPROMOTE1=//SCLMPROM JOB (#ACCT),'SCLM PROMOTE',CLASS=A,MSGCLASS=X,
BATCHPROMOTE2=//         NOTIFY=&SYSUID,REGION=128M
BATCHPROMOTE3=//*
BATCHPROMOTE4=//*
*
* BUILD Security flag for SAF/RACF security call and possible Surrogate
* ID switch
BUILDSECURITY=N
* PROMOTE Security flag for SAF/RACF security call and possible
* Surrogate ID switch
PROMOTESECURITY=N
* J2EE DEPLOY security flag for SAF/RACF security call and possible
* Surrogate ID switch
DEPLOYSECURITY=N
```

Figure 16. Sample PROJECT-specific SCLM project setting

All of the listed options are optional. Defaults are used if nothing is specified in the `SITE.conf` or the `project.conf`.

Table 6. SITE/Project options

`BUILDAPPROVER=approval product/NONE`	Specify the name of the approval product used for the build process. Currently the only supported product is IBM Breeze for SCLM, which is selected with the `BREEZE` keyword. Default is `NONE`.
`PROMOTEAPPROVER=approval product/NONE`	Specify the name of the approval product used for the promote process. Currently the only supported product is IBM 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=N/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.
`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.

Table 6. SITE/Project options (continued)

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 for the build step and possible do a surrogate ID switch. For more information see Chapter 4, “SCLM security,” on page 45.
PROMOTESECURITY=Y/N	Specify Y to invoke SAF/RACF security call for the promote step and possible do a surrogate ID switch. For more information see Chapter 4, “SCLM security,” on page 45.
DEPLOYSECURITY=Y/N	Specify Y to invoke SAF/RACF security call for the deploy step and possible do a surrogate ID switch. For more information see Chapter 4, “SCLM security,” on page 45.
ASCII=ASCII codepage	Specify the ASCII code page to override the ASCII code page specified in TRANSLATE.conf. For example: ASCII=UTF-8
EBCDIC=EBCDIC codepage	Specify the EBCDIC code page to override the EBCDIC code page specified in TRANSLATE.conf. 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
NOTRANLANG=SCLM Language	Use the NOTRANLANG keyword to remove an already specified TRANLANG from the list allowable for this SCLM project as specified in TRANSLATE.conf. For example: NOTRANLANG=JAVA
LONGLANG=SCLM Language	Specify a LONGLANG parameter to be added to the list of LONGLANG parameters specified in 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 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 Developer for System z.
PROJECTLISTALL=Y	The 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 code page support and default SCLM language support to be applied across SCLM Developer Toolkit. In this example, TRANSLATE.conf has the values listed below.

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

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

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

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

SCLM Project SCLMPRJ2 might have some different override settings in SCLMPRJ2.conf:

```
* 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 code pages for source translations to the Hebrew code page pair. Additionally SCLM Languages of DOC and XLS are added to the defaults from `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 `SITE.conf` 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 `SITE.conf`:

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

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 `SITE.conf` for the JAVA and COBOL language definitions. All other languages will have the default settings, as specified in `SITE.conf`.

Chapter 3. 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.

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 1970s : System R . Since then, SQL has grown, been standardized (by ANSI and ISO) and appeared in many flavors 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 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 that 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 expenditures.

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 which stands for *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 of the Redbook *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More*.

Table 7. Comparing JDBC and SQLJ

	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.
MONITORING	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 timeout), and so on.	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.

Table 7. Comparing JDBC and SQLJ (continued)

	SQLJ (static)	JDBC (dynamic)
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?

A Serialized Profile is a 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 preprocessed 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 an SCLM user exit.

SQLJ program preparation

Before discussing how to use SCLM Developer Toolkit to build SQLJ programs, 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.

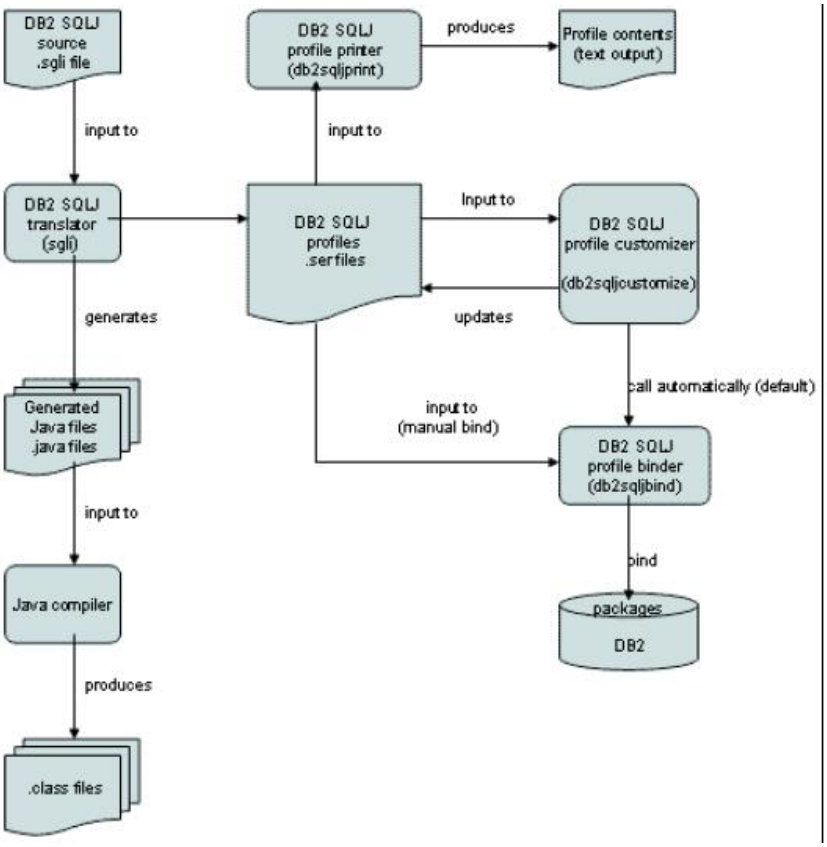


Figure 17. SQLJ program preparation

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.

Many SQLJ source files will only refer to 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:

progname`_SJProfile`*X*`.ser`

Where:

- *progname* represents the name of the program. This is determined by removing the `.sqlj` extension from the input source filename.

- *X* represents 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 *db2profc*. 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 five profiles, then those five 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 through 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 an SQLJ source file and produces serialized profiles and Java source code.

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

An 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 want 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.

Table 8. SCLM translator types for SQLJ

SQLJSER	This is required to store the generated serialized profile files (`.ser` files) created or 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.

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. 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, 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 there will be a reference to the master build script. This can be found in the property. Most of the configuration listed for the translation and customization steps goes on in this file.

Note: 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.

sqlj.* properties

Each property listed inTable 9 on page 39 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.

Table 9. sqlj. properties*

Name	Value	Description
sqlj.exec	usr/lpp/rdz/bin/bwbsqlc.rex	Specifies the location of the sqlj & db2sqljcustomize exec routine `bwbsqlc.rex`, which is located in the Developer for System z install directory.
sqlj.class	sqlj.tools.Sqlj	Specify the sqlj class name. This is the name of the class invoked by the sqlj 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 Table 10 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.

Table 10. db2sqljcustomize. 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 want 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 DBRMs,

rather than binding through the serialized profiles. In order to use the serialized profiles, changes must be made to BWBSQLB. This is discussed in the subtopic “Binding [Serialized Profile]” on page 42.

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 want 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 or 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 by 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 **Note:** The property file being created needs to be basedir’/’propfile.
Sqljf	A list of file names, representing the serialized profiles (.ser) to be processed by db2sqljcustomize

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 through serialized profiles, and will not generate a DBRM.

If you provide this parameter, SCLM Developer Toolkit will pick up the generated DBRMs, and store them in SCLM for future use. One advantage of using this technique is that you can easily perform a DB2 bind in an 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 the following four steps:

1. Set the appropriate arguments for *sqlj*, as follows:

   ```
   <!-- 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 *no* when used in conjunction with *compile=false*.
status	Prints immediate status display of SQLJ processing.

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

   ```
   <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.

Argument	Description
onlinecheck yes	Perform online checking on the system specified by the *url* parameter. Defaults to yes if *url* is supplied, and no otherwise.
Bindoptions ISOLATION(CS)	Instructs the binder to create a single package (cursor stability). Used in conjunction with *singlepkgname* (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 the 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 eight 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 the following three steps:

1. Set the appropriate arguments for *sqlj*, as follows:

   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 *no* when used in conjunction with *compile=false*.
status	Prints immediate status display of SQLJ processing.

2. Set the appropriate arguments for *db2sqljcustomize*, as follows:

   ```
   <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 *db2bind*.
onlinecheck yes	Perform online checking on the system specified by the *url* parameter. Defaults to yes if *url* 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"/>
```

Chapter 4. SCLM security

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

- The security functionality is controlled by security flags set in the SCLM Developer Toolkit configuration files and by profiles in your security product.
- If the security flag is set for a function and the requesting user ID meets the authority check by your security product, then the function is allowed to continue. If the requester fails the authority check, then the function ends with RC=8 and a security error message.
- If defined that way in your security product, the function continues using another (surrogate) user ID.

Security flag

You can set a Build, Promote, Deploy security flag in the SITE/PROJECT configuration files. Refer to"SITE and project-specific options" on page 25 to learn more about the SITE/PROJECT configuration files. The following directives control the security flag of the Build, Promote and Deploy functions, respectively:

- `BUILDSECURITY = Y`
- `PROMOTESECURITY = Y`
- `DEPLOYSECURITY = Y`

Set up in your Security product

The SCLM Build, Promote, and Deploy function security uses the SAF/RACROUTE security interface, which is supported by all major security products.

The listed sample commands are for RACF®. Refer to your security product documentation if you use another product.

- 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.

Security profiles

The security administrator defines the required profiles in the `XFACILIT` class using the **RDEFINE** command, and access permissions with the **PERMIT** command. Refer toTable 11 on page 46 to learn which profiles need to be defined.

Table 11. SCLMDT Developer Toolkit security profiles

Function	XFACILIT profile	Required access
Build	SCLM.BUILD.project.projdef.group.type.member	READ
Promote	SCLM.PROMOTE.project.projdef.group.type.member	READ
Deploy	SCLM.DEPLOY.server.application.node.cell.project.projdef.group.type	READ

The list below describes the meaning of the different profile part names:

SCLM	Constant, indicates an SCLM function profile
BUILD	Constant, indicates the BUILD function
PROMOTE	Constant, indicates the PROMOTE function
DEPLOY	Constant, indicates the DEPLOY function
project	The SCLM project name, or * for all projects
projdef	The alternate project name (equals the Project name by default), or * for all alternate projects
Group	The SCLM group to build/promote/deploy, or * for all groups
Type	The SCLM type, or * for all types
Member	The SCLM member to build/promote, or * for all members
Server	The target deployment server (`SERVER_NAME` in the Ant deploy script), or * for all servers
Application	The target WAS application name (`APPLICATION_NAME` in the Ant deploy script), or * for all applications
Node	The target WAS node name (`NODE_NAME` in the Ant deploy script), or * for all nodes
Cell	The target WAS cell name (`CELL_NAME` in the Ant deploy script), or * for all cells

Surrogate user ID

The Build, Promote, and Deploy functions support the usage of a surrogate user ID to execute the function. If activated, then all authorized calls to the function will result in the function being executed with the permissions of the surrogate user ID, not the requesting user ID.

The activation of the surrogate user ID is profile specific, and is controlled by the "`SUID=userid`" string in the `APPLDATA` field of the security profile, where `userid` is the surrogate user ID. If the string is present, the surrogate user ID will be used, if not, the requester's user ID will be used.

Example: Build

This example lists the security product definitions needed to protect the Build function for a given project. The same sample can be used for securing the Promote function by replacing the SCLM.BUILD.* rule with the SCLM.PROMOTE.* rule.

The following profile definition secures all members for build in project =TESTPROJ at group level PROD. A surrogate user ID is also defined.

```
RDEFINE XFACILIT SCLM.BUILD.TESTPROJ.TESTPROJ.PROD.*.* UACC(NONE)
 APPLDATA('SUID=IBMUSER')
SETROPTS RACLIST(XFACILIT) REFRESH
```

This example defines an SCLM build profile where:

- Project = TESTPROJ

- Alternate project definition = TESTPROJ
- SCLM group = PROD
- SCLM type = all types
- Member = all members
- Universal access = NONE (by default, nobody is authorized to the profile)
- Surrogate user ID = IBMUSER

Note: The SETROPTS command updates the in-memory tables of RACF, and thus activates the definition.

The following example shows security permissions defined for individual users (or user groups) for the TESTPROJ project of the previous example:

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

The PERMIT matches the original RDEFINE profile and permits user USERID to build any member from project TESTPROJ and group PROD. Since there is a surrogate user ID stored in the application data (APPLDATA) field of the matching profile, the BUILD will be processed under that surrogate user ID (in this case IBMUSER).

Example: Deploy

Below is an example for creating a deployment profile.

```
RDEFINE
XFACILIT SCLM.DEPLOY.SERVERY.TESTAPP.NODE1.CELL1.TESTPROJ.TESTPROJ.PROD.J2EEDEP
UACC(NONE)
```

WAS details:

- Server name = SERVERY
- Application name = TESTAPP
- Node name = NODE1
- Cell name = CELL1

SCLM project details:

- Project = TESTPROJ
- Alternate project definition = TESTPROJ
- SCLM group = PROD
- SCLM type = J2EEDEP

Other information:

- Universal access = NONE (by default, nobody is authorized to the profile)
- No surrogate user ID

The following example shows security permissions defined for a user group for the previously defined deployment profile:

```
PERMIT SCLM.DEPLOY.SERVERY.TESTAPP.NODE1.CELL1.TESTPROJ.TESTPROJ.PROD.J2EEDEP
   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 through 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 through 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 Developer for System z sample directory, `/usr/lpp/rdz/samples/`. A copy, which can be customized to match your needs, has been placed in `/etc/rdz/sclmdt/script/` during Developer for System z customization.

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 redirected to a z/OS UNIX file or log. The REXX 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.

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.

bwbsqld.rex
: Sample ANT XML SQLJ JAVA build script

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 ($PATH) and locate the SCLM Developer Toolkit load modules ($STEPLIB) if the SCLM Developer Toolkit load modules are not in the LINKLIST. For example:

```
PATH=/etc/rdz/sclmdt/script:$PATH
STEPLIB=FEK.SFEKLOAD:FEK.SFEKAUTH:$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 pipe symbol.

Invocation Example

Using the sample names as provided the CRON build exec can be invoked as follows ($ is the z/OS UNIX prompt):

```
$   BWBCRONB | BWBCRON1 > $HOME/bwbcronb.log
  30 19 * * 1-5 BWBCRONB | BWBCRON1 > /u/userid/bwbcronb.log ;
```

Refer to *UNIX System Services Command Reference* (SA22-7802) and *UNIX System Services Planning* (GA22-7800) for more information about the CRON services available and the CRONTAB format.

Alternatively, use the online z/OS UNIX command, **man**:

- `man cron`
- `man crontab`
- `man at`

CRON Build job samples

This section shows the BWBCRON1, BWBCRONB, BWBCRONP job samples as provided in the SFEKSAMV library.

The following example shows the BWBCRON1 code.

```
/* REXX */
/* Customize STEPLIB, _SCLMDT_CONF_HOME and _SCLMDT_WORK_HOME */
/*
The STEPLIB should reflect the load libraries for
Rational Developer for System z.
If these datasets resides in the LINKLIST then set STEPLIB to '' .
*/
STEPLIB = 'FEK.SFEKLOAD:FEK.SFEKAUTH'
/*
The Environment variables _SCLMDT_CONF_HOME and _SCLMDT_WORK_HOME determines the HOME
directories where the configuration files and workarea reside for
SCLM Developer Toolkit. Refer to the Rational Developer for System z
configuration file /etc/rdz/rsed.envvars for the correct value.
*/
_SCLMDT_CONF_HOME = '/etc/rdz/sclmdt'
_SCLMDT_WORK_HOME = '/var/rdz'

/* 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 '_SCLMDT_CONF_HOME',_SCLMDT_CONF_HOME
CALL ENVIRONMENT '_SCLMDT_WORK_HOME',_SCLMDT_WORK_HOME


CALL BWBINT

EXIT
```

Figure 18. BWBCRON1 - CRON Build exec

The following example shows the BWBCRONB code.

```
/* 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  = ''                           /* SCLM Project            */
PROJDEF  = ''                           /* Alt proj definition     */
TYPE     = ''                           /* SCLM Type               */
MEMBER   = ''                           /* SCLM Member name        */
GROUP    = ''                           /* SCLM Group              */
GROUPBLD = ''                           /* Build at Group          */
REPDGRP  = ''                           /* 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 dataset         */
BLDRPTDS = ''                           /* Report  dataset         */
BLDLSTDS = ''                           /* list    dataset         */
BLDEXTDS = ''                           /* Exit    dataset         */
SUBMIT   = 'BATCH'                      /* Online or Batch         */
/*
   IF running in BATCH and require a JCL JOBCARD to override the
   default then add up to 4 lines of JOBCARD lines.
   Specify in the format of LINE. and include LINECNT variable for
   number of lines.
*/
LINECNT = 2
LINE.1 = '//SCLMBLD JOB (XXX),SCLMBUILD,MSGCLASS=X,NOTIFY=&SYSUID,'
LINE.2 = '//         CLASS=A,REGION=0M'

/* 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
If (SUBMIT = 'BATCH') & (LINECNT > 0) then
  Do
    SAY '<JOBCARD>'
    Do i = 1 to LINECNT
    SAY LINE.i
    End
    SAY '</JOBCARD>'
  End
```

Figure 19. BWBCRONB - Build parameter file

The following example shows the BWBCRONP code.

```
/* SAMPLE PROMOTE PARAMETER FILE USED FOR CRON INITIATED PROMOTES */
/* Update Promote parms below in quotes. */
/* if parameter required as Blank then set as ''  */
FUNCTION = 'PROMOTE'
PROJECT  = ''                          /* SCLM Project             */
PROJDEF  = ''                          /* Alt proj definition(opt) */
TYPE     = ''                          /* SCLM Type                */
MEMBER   = ''                          /* SCLM Member name         */
GROUP    = ''                          /* SCLM Group               */
GROUPPRM = ''                          /* Promote at Group (opt)   */
REPDGRP  = ''                          /* Users Development group  */
PRMREPT  = 'Y'                         /* Generate Promote report  */
PRMMSG   = 'Y'                         /* Generate Promote Messages*/
PRMSCOPE = 'N'                         /* Promote Scope E/L/N/S    */
PRMMODE  = 'C'                         /* Promote Mode C/F/R/U     */
PRMMSGDS = ''                          /* Message dataset          */
PRMRPTDS = ''                          /* Report  dataset          */
PRMEXTDS = ''                          /* Exit    dataset          */
SUBMIT   = 'BATCH'                     /* Online or Batch          */
/*
   IF running in BATCH and require a JCL JOBCARD to override the
   default then add up to 4 lines of JOBCARD lines.
   Specify in the format of LINE. and include LINECNT variable for
   number of lines.
*/
LINECNT = 2
LINE.1 = '//SCLMBLD JOB (XXX),SCLMBUILD,MSGCLASS=X,NOTIFY=&SYSUID,'
LINE.2 = '//         CLASS=A,REGION=0M'

/* DO NOT ALTER PARM PROMOTE VARIABLE BELOW */
PARM1 = 'SCLMFUNC='FUNCTION'&PROJECT='PROJECT'&PROJDEF='PROJDEF||,
        '&TYPE='TYPE'&MEMBER='MEMBER'&GROUP='GROUP'&GROUPPRM='GROUPPRM||,
        '&REPDGRP='REPDGRP'&PRMREPT='PRMREPT||,
        '&PRMMSG='PRMMSG'&PRMSCOPE='PRMSCOPE'&PRMMODE='PRMMODE||,
        '&PRMMSGDS='PRMMSGDS'&PRMRPTDS='PRMRPTDS'&PRMEXTDS='PRMEXTDS||,
        '&SUBMIT='SUBMIT
/* outputs parameter string as input to BWBCRON1 */
SAY PARM1
If (SUBMIT = 'BATCH') & (LINECNT > 0) then
  Do
    SAY '<JOBCARD>'
    Do i = 1 to LINECNT
    SAY LINE.i
    End
    SAY '</JOBCARD>'
  End
```

Figure 20. BWBCRONP - Promote parameter file

Appendix A. SCLM overview

SCLM Developer Toolkit, a function of IBM Rational Developer for System z, provides the means by which distributed applications written in Eclipse can be managed and built using Software Configuration and Library Manager (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 SCLM 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 through 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 SCLM 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 format” on page 9.

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.

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 refer to 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 plugin 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 refer to 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 SCLM 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 16.

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. 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 1.4 and later. For z/OS 1.8 and later, this PTF is not required.

The conversion algorithm performs the following 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.

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.

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 12.

Syntax:

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

Invocation example:

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

Table 12. 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 through 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 through 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 does not match the VSAM records are read and compared until the `ls_lngname_key` does not 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, as follows:

- 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 the following:

```
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.

The following example shows sample REXX code for invoking 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('FEK.#CUST.LSTRANS.FILE') SHR REUSE"
   "ALLOC DD(LSTRNPTH) DA('FEK.#CUST.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 21. Sample REXX for Translate module invocation

Appendix C. SCLM Developer Toolkit API

This appendix documents the Application Programming Interface (API) for the SCLM Developer Toolkit host services. The API uses an XML-based request and response format and must be accessed through z/OS UNIX Systems Services.

The API enables users to use the SCLM Developer Toolkit host services with their own client/plugin and transport layer if desired.

A sample Java program (with input and output) is provided, accessing the SCLMDT host services API using an HTTP server as the transport mechanism.

Many of the function requests are based on the current SCLM host services and more information on the similar parameter settings for common functions may be found in the SCLM guide and reference for the relevant z/OS release.

Note: This API documents the host services currently used by the SCLM Developer Toolkit Client. Therefore many of the functions may be less suitable for customer usage, or will require additional client side validation.

Invocation format

The following sample command illustrates how the SCLMDT host services can be invoked:

```
cat sclmdt_request.xml | BWBXML > sclmdt_response.xml
```

`cat`	z/OS UNIX command to display text files
`sclmdt_request.xml`	The XML input file with the user request
`|`	z/OS UNIX command to pipe the output of the previous command as the input of the next
`BWBXML`	The XML conversion script, which resides in the Developer for System z `/bin` directory, that invokes the SCLMDT service interface
`>`	z/OS UNIX command to redirect the output of the previous command to a file
`sclmdt_response.xml`	The XML output file containing the service response

Notes:

1. The `PATH` environment variable must contain the directory location of `BWBXML`, for example:

   ```
   export PATH=/usr/lpp/rdz/bin:$PATH
   ```

2. When using HTTP as transport mechanism:
   - The interface script `BWBXML` can be invoked as a CGI script (`EXEC` type in the HTTP server directives).
   - The request is read as `STDIN` via `BWBXML` so may be passed to the CGI script as a `POST` request (see the sample program).

XML schema for SCLMDT commands

The following example shows the XML schema for SCLMDT commands referenced in the XML input file. This sample is also available as member BWBXSD1 in the sample library FEK.SFEKSAMV.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="SCLMDT-INPUT">
 <xs:complexType>
  <xs:all>

   <xs:element name="SERVICE-REQUEST">
    <xs:complexType>
     <xs:all>

   <xs:element name="service">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <!-- Specifies native TSO or ISPF service call -->
      <xs:enumeration value="ISPF"/>
      <xs:enumeration value="TSO"/>
      <xs:enumeration value="SCLM"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:element>

   <xs:element name="session" minOccurs="0">
    <xs:simpleType>
     <xs:restriction base="xs:string">
      <!-- Default NONE : Session terminates after service call -->
      <xs:enumeration value="NONE"/>
      <!-- Reuseable ISPF session stays active between calls -->
      <xs:enumeration value="REUSE"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:element>

    <!-- Use existing ISPF profile in call -->
   <xs:element name="ispprof" type="xs:string" minOccurs="0"/>

    <!-- Free form TSO/ISPF command -->
   <xs:element name="command" type="xs:string" minOccurs="0"/>

    <!-- List of all SCLM DT functions available -->

    <!-- sclmfunc : SCLM function selected -->
   <xs:element name="sclmfunc" minOccurs="0">
    <xs:simpleType>
     <xs:restriction base="xs:string">

      <xs:enumeration value="BUILD"/>    <!-- SCLM build function -->
<!-- Build parms : project,projdef,group,repdgrp,type,member,bldmode,bldlist,
     bldrept,bldmsg,bldmsgds,bldlist,bldlstds,bldextds,groupbld,submit -->

      <xs:enumeration value="PROMOTE"/>  <!-- SCLM promote function -->
<!-- Promote parms : project,projdef,group,repdgrp,type,member,prmmode,prmrept,
     prmmsg,prmsgds,prmextds,groupprm,submit -->

      <xs:enumeration value="EDIT"/>     <!-- EDIT member -->
<!-- Edit parms : project,projdef,group,repdgrp,type,member,lang -->
       <!-- Note: member transferred to DTinstall/WORKAREA -->

      <xs:enumeration value="BROWSE"/>   <!-- Browse member -->
<!-- Browse parms : project,projdef,group,repdgrp,type,member,lang -->
       <!-- Note: member transferred to DTinstall/WORKAREA -->

      <xs:enumeration value="SAVE"/>     <!-- Save edited member -->
<!-- Save parms : project,projdef,group,repdgrp,type,member,lang -->
       <!-- Note: member received from DTinstall/WORKAREA -->

      <xs:enumeration value="DELETE"/>   <!-- SCLM delete member -->
<!-- Delete parms : project,projdef,group,repdgrp,type,member,delflag -->

      <xs:enumeration value="UNLOCK"/>   <!-- SCLM unlock member -->
<!-- Unlock parms : project,projdef,group,repdgrp,type,member -->

      <xs:enumeration value="DEPLOY"/>   <!-- J2EE deploy function -->
```

Request functions and parameters

Function format

The following example shows the basic structure of the XML input file, where `#function` represents the function called, and `#parameter` and `#value` are a parameter and its value.

```
<?xml version="1.0"?>
<SCLMDT-INPUT
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="sclmdt.xsd">
 <SERVICE-REQUEST>
  <service>SCLM</service>
  <session>NONE</session>
  <sclmfunc>#function</sclmfunc>
  <#parameter>#value</#parameter>
  ...
 </SERVICE-REQUEST>
</SCLMDT-INPUT>
```

Figure 23. Basic structure of the XML input file

Note: If a parameter is specified more than once, the definitions in the last instance will be used.

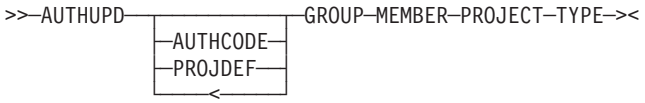
the following list represents some general remarks about the parameters and the way they are documented in this publication:

- Parameters are not positional.
- Square brackets ([]) denote an optional parameter for the function.
- Curly brackets ({}) denote a list of possible values for the parameter.
- A horizontal bar (|) separates multiple values in a list. The underlined value, if any, is the default one.
- Uppercase keywords represent constants that must be coded as-is, lowercase keywords represent placeholders that must be replaced with custom values.
- Each parameter requires a value, but only the values that are part of a list are documented.
- References to `WORKAREA` refer to the z/OS UNIX `WORKAREA` directory, by default located in `/var/rdz/sclmdt/`.
- References to `member` require the usage of the shortname (SCLM) member name, unless explicitly stated otherwise.

Function list

AUTHUPD – Change SCLM authority code

This function changes the authority code of a member.



Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

[AUTHCODE]
SCLM Authority Code - New authority code to be assigned to member.

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

BROWSE – Browse SCLM member

This function copies a member from SCLM to z/OS UNIX directory `WORKAREA/userid/EDIT/`. No edit lock is done in SCLM.

Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

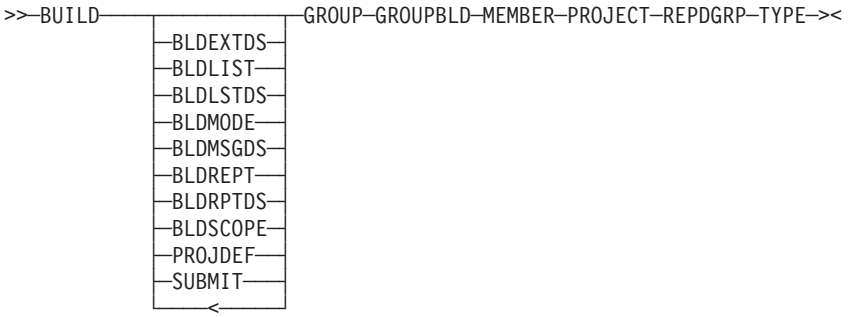
TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

BUILD – Build SCLM member

This function instructs SCLM to compile, link, and integrate software components according to the project architecture definitions.



Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

GROUPBLD
SCLM Group - The SCLM group selected to build the required member at.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

REPDGRP
SCLM Development Group - The development group of the user.

TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

[BLDEXTDS [dsn | NONE]]
Build Exit Data Set - NONE or the name of the data set that will hold the build exit output, if using a build exit. If the data set does not exist, a new data set will be allocated. Default is NONE.

[BLDLIST [Y | N]]
Build Listing - Indicates if build translator listings are only to be copied to the list data set on error. Default is Y.

[BLDLSTDS [dsn | NONE]]
Build List Data Set - NONE or the name of the data set that will hold the build listings. If the data set does not exist, a new data set will be allocated. The build listings will also be returned in the XML response file, regardless of this setting. Default is NONE.

[BLDMODE [C | F | R | U]]
Build Mode - Indicates the build mode (C=conditional, F=forced, R=report, U=unconditional). Default is C.

[BLDMSGDS [dsn | NONE]]
Build Message Data Set - NONE or the name of the data set that will hold the build messages. If the data set does not exist, a new data set will be allocated. The build messages will also be returned in the XML response file, regardless of this setting. Default is NONE.

[BLDREPT [Y | N|]
Build Report - Indicates if a build report is to be produced. Default is Y.

[BLDRPTDS [dsn | NONE]]
Build Report Data Set - NONE or the name of the data set that will hold the build report. If the data set does not exist, a new data set will be allocated. The build report will also be returned in the XML response file, regardless of this setting. Default is NONE.

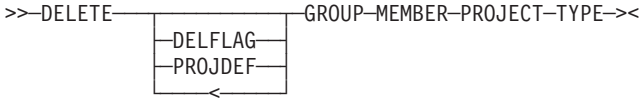
[BLDSCOPE [E | L | N | S]]
Build Scope - Indicates the build scope (E=extended, L=limited, N=normal, S=subunit). Default is N.

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

[SUBMIT [BATCH | ONLINE]]
Submit Method - The build is submitted either online or in batch. Default is ONLINE.

DELETE – Delete SCLM member

This function deletes an SCLM member.



Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

TYPE SCLM Type - The SCLM type containing the selected member.

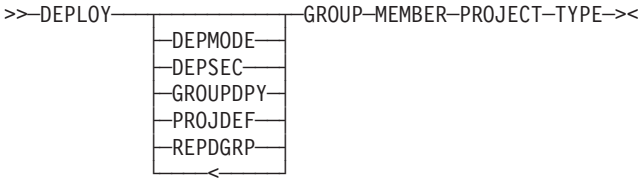
Optional parameters:

[DELFLAG [TEXT | ACCT | TXBM | BMAP | ALL]]
Delete Flag - Indicates that either Text, Account, Build Map, a combination of Build Map and Text, or everything is to be deleted for a given member. Default is ALL.

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

DEPLOY – Deploy a J2EE EAR file

The DEPLOY function will run the deploy script referenced by member to deploy a J2EE enterprise archive file (EAR) from USS or SCLM into a Websphere Application Server (WAS). Refer to the SCLM Developer Toolkit user guide for more information on creating a deploy script member.



Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

[DEPSEC {Y | N}]
Secure Deploy - Flag to indicate if a security rule check and a possible surrogate user ID switch is done for this deployment.

[DEPMODE {R}]
Deploy Mode - If set to 'R', report mode only. No deploy will occur.

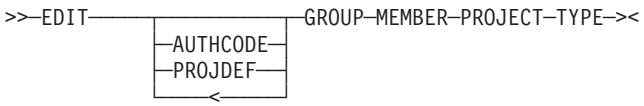
[GROUPDPY]
Deploy Group - Is the SCLM group where the EAR file will be deployed from (or search up group hierarchy if not found). If set to 'USS', the EAR file will be deployed straight from the z/OS UNIX directory specified as part of the EAR name in variable EAR_FILE_NAME. This variable is defined in the deploy script member referenced as 'MEMBER'.

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

[REPDGRP]
SCLM Development Group - The development group of the user.

EDIT – Edit SCLM member

This function copies a member from SCLM to z/OS UNIX directory `WORKAREA/userid/EDIT/`. It also creates an SCLM lock on the member with the user ID as access key.



Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

TYPE SCLM Type - The SCLM type containing the selected member.

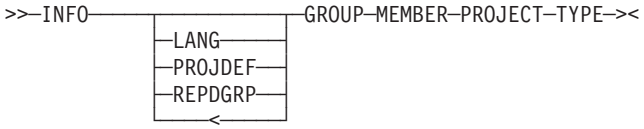
Optional parameters:

[AUTHCODE]
SCLM Authority Code - New authority code to be assigned to member.

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

INFO – SCLM member status information

This function provides SCLM member status information.



Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

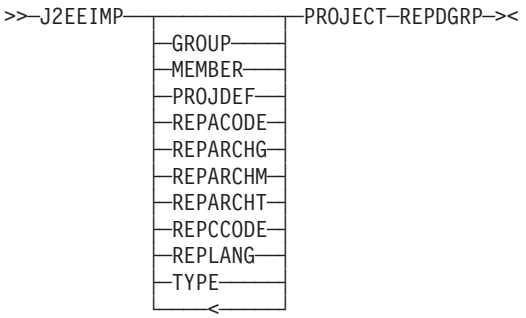
[LANG]
SCLM Language - The SCLM language of the selected member.

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

[REPDGRP]
SCLM Development Group - The development group of the user.

J2EEIMP – Import project from SCLM

This function imports a project from SCLM into the z/OS UNIX `/var/rdz/WORKAREA/userid` directory in JAR (zipped) format. The JAR project file can then be copied to the client. The name of the project file is returned in the `J2EEFILE` keyword of the (XML) function output.



Required parameters:

[PROJECT]
SCLM Project Name - The SCLM project name.

[REPDGRP]
SCLM Development Group - The development group of the user.

Optional parameters:

[GROUP {group | group* | *}]
Group Hierarchy - The group hierarchy for member selection. If `REPARCHM` is set, the related to that Archdef are located in the `group*`.

[MEMBER {member | member* | *}]
SCLM Member - Selected SCLM member(s).

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

[REPACODE]
Authority Code - Authority code to filter on.

[REPARCHG]
Archdef Group - The SCLM group where the Archdef member resides in.

[REPARCHM]
Archdef Member - Name of the Archdef member to be selected for import.

[REPARCHT]
Archdef Type - The SCLM type the Archdef member resides in.

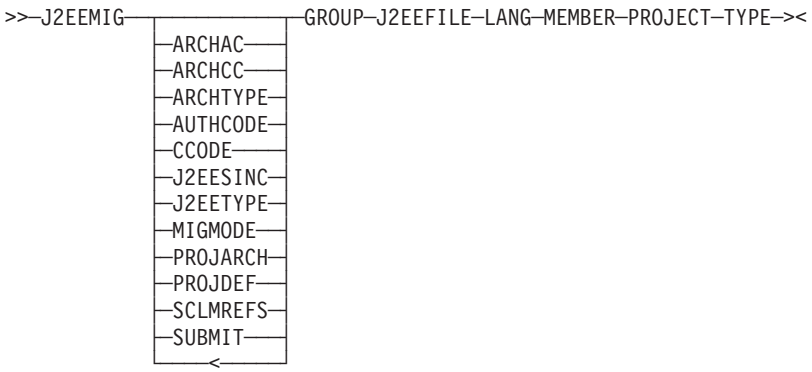
[REPCCODE]
Change Code - Change Code to filter on.

[REPLANG]
SCLM Language - Language to filter on.

[TYPE {type | type* | *}]
SCLM Type - SCLM type(s) for member selection.

J2EEMIG – Migrate project into SCLM

This function will take a zipped file (JAR format) from the USS directory, extract it, and migrate all the members contained in this file into SCLM , translating longname to shortname if required.



Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

J2EEFILE
Input File Name - The file name of the JAR (zipped) project to be imported in SCLM. The JAR file must reside in the z/OS UNIX `/var/rdz/WORKAREA/userid` directory.

LANG
SCLM Language - The SCLM language of the selected member.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

[ARCHAC]
Archdef Authority Code - Set Archdef authority code.

[ARCHCC]
Archdef Change Code - Set Archdef change code.

[ARCHTYPE [ARCHDEF | archtype]]
Archdef SCLM Type - Set Archdef type.

[AUTHCODE]
Member Authority Code - Set authority code of Archdef members.

[CCODE]
Member Change Code - Set change code of Archdef members.

[J2EESINC]
J2EE SINC Build Scrip.- Name of the build script which resides in TYPE J2EEBLD and is referenced by the `SINC` keyword in the ARCHDEF member.

[J2EETYPE {JAR | WAR | EAR}]
J2EE Type - Specify JAR WAR or EAR for the J2EE Archdef project.

[MIGMODE {FORCE}]
Migrate mode – `FORCE` will replace existing members Default is a conditional migrate.

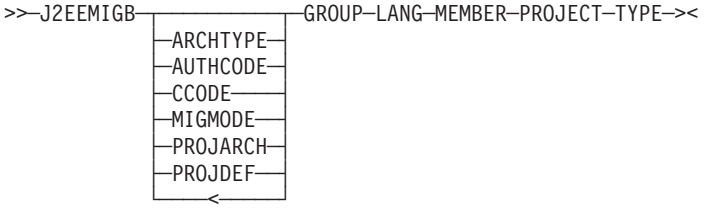
[PROJARCH]
Archdef Project - Name of the Archdef to be updated or created

[SCLMREFS]
SCLM references - Additional Archdefs or parts.

[SUBMIT [BATCH | ONLINE]]
Submit Method - The migration is submitted either online or in batch. Default is `ONLINE`.

J2EEMIGB – Batch Migrate project into SCLM

This function will set up and run the BATCH job for migrate.



Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

LANG
SCLM Language - The SCLM language of the selected member.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

[ARCHTYPE]
Archdef SCLM Type - Set Archdef type.

[AUTHCODE]
Member Authority Code - Set authority code of Archdef members.

[CCODE]
Member Change Code - Set change code of Archdef members.

[MIGMODE {FORCE}]
Migrate mode – `FORCE` will replace existing members Default is a conditional migrate.

[PROJARCH]
Archdef Project - Name of the Archdef to be updated or created.

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

JARCOPY – Copy JAR file

This function copies a JAR file stored in SCLM into a z/OS UNIX directory, which can be used as a CLASSPATH directory.

```
>>—JARCOPY—┬————————┬—CLASSDIR—GROUP—JARNAME—PROJECT—REPDGRP—TYPE—><
           └—PROJDEF—┘
```

Required parameters:

CLASSDIR
Classpath Directory - Name of the target z/OS UNIX directory.

GROUP
SCLM Group - The SCLM group where the selected member resides.

JARNAME
JAR File Name - Long name of target JAR file. The SCLM short name is looked up automatically.

PROJECT
SCLM Project Name - The SCLM project name.

REPDGRP
SCLM Development Group - The development group of the user.

TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

JOBSTAT – Retrieve batch job status

This function retrieves the status of a specific batch job.

```
>>—JOBSTAT—JOBFUNC—JOBID—JOBNAME—PROJECT—><
```

Parameters:

JOBFUNC {JOBSTAT | JOBRETR}
Function Select - Retrieve the job status (JOBSTAT) or the job output (JOBRETR).

JOBID
Job ID - Job number of batch job.

JOBNAME
Job Name - Name of batch job.

PROJECT
SCLM Project Name - The SCLM project name.

LRECL – Retrieve LRECL of SCLM data set

This function retrieves the logical record length of an SCLM data set.

```
>>-LRECL-------------------GROUP-PROJECT-REPDGRP-TYPE-><
            '-PROJDEF-'
```

Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

PROJECT
SCLM Project Name - The SCLM project name.

REPDGRP
SCLM Development Group - The development group of the user.

Optional parameters:

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

MIGDSN – List NON-SCLM data sets and members

This function lists selected data sets and members that are not SCLM managed (for subsequent migration into SCLM).

```
>>-MIGDSN-------------------MIGDSN-MIGMEM-PROJECT-><
          |__PROJDEF_|
```

Required parameters:

MIGDSN [dsn | *]
Data Set Filter - Data set filter for list. Default is *.

MIGMEM [member | *]
Member Filter - Member filter for list. Default is *.

PROJECT
SCLM Project Name - The SCLM project name.

Optional parameters:

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

MIGPDS – Migrate NON-SCLM data sets and members into SCLM

This function migrates selected data sets and members that are not SCLM managed into SCLM.

```
>>-MIGPDS-------------------GROUP-PROJECT-TYPE-><
          |_PROJDEF__|
```

Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

PROJECT
SCLM Project Name - The SCLM project name.

TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

PROJGRPS – Retrieve SCLM groups for a project

This function retrieves the SCLM groups for a selected project.

Required parameters:

PROJECT
SCLM Project Name - The SCLM project name.

Optional parameters:

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

PROJINFO – Retrieve SCLM project information

This function retrieves the SCLM project information for a selected project.

Required parameters:

PROJECT
SCLM Project Name - The SCLM project name.

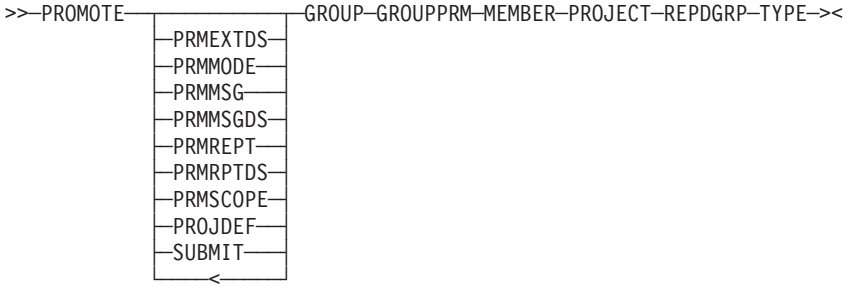
REPDGRP
SCLM Development Group - The development group of the user.

Optional parameters:

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

PROMOTE – Promote SCLM member

This function promotes an SCLM member (or Archdef) up the SCLM group hierarchy according to the projects architecture definition and project definition.



Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

GROUPPRM
SCLM Group - The SCLM group selected to promote the required member from.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

REPDGRP
SCLM Development Group - The development group of the user.

TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

[PRMEXTDS [dsn | NONE]]
Promote Exit Data Set - NONE or the data set name that will hold the promote exit output, if using a promote exit. If the data set does not exist, a new data set will be allocated. Default is NONE.

[PRMMODE [C | R | U]]
Promote Mode - Indicates the promote mode (C=conditional, R=report, U=unconditional). Default is C.

[PRMMSG [Y | N]]
Promote Messages - Set to Y to include promote messages. Default is N.

[PRMMSGDS [dsn | NONE]]
Promote Message Data Set - NONE or the data set name that will hold the promote messages. If the data set does not exist, a new data set will be allocated. The promote messages will also be returned in XML response file, if PRMMSG is set to Y. Default is NONE.

[PRMREPT [Y | N]]
Promote Report - Set to Y to include promote report. Default is N.

[PRMRPTDS [dsn | NONE]]
Promote Report Data Set - NONE or the name of the data set that will hold the promote report. If the data set does not exist, a new data set will be allocated. The promote messages will also be returned in XML response file, if PRMMSG is set to Y. Default is NONE.

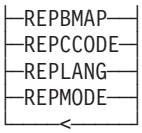
[PRMSCOPE [E | N | S]]
Promote Scope - Indicates the promote scope (E=extended, N=normal, S=subunit). Default is N.

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

[SUBMIT [BATCH | ONLINE]]
Submit Method - The promote is submitted either online or in batch. Default is ONLINE.

REPORT – Create project report

The report function provides a DBUTIL hierarchy report of the project, according to the report parameters and filters used.



Required parameters:

PROJECT
SCLM Project Name - The SCLM project name.

REPDGRP
SCLM Development Group - The development group of the user.

REPGRP {group | group* | * }
SCLM Group - SCLM group(s) to be filtered on.

REPMEM {member | mem* | * }
SCLM Member - SCLM member(s) to be filtered on.

REPTYPE {type | type* | * }
SCLM Type - SCLM type(s) to be filtered on.

Optional parameters:

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

[REPACODE]
Authority Code - Authority code to filter on.

[REPBMAP [Y | N]]
Build Maps - Flag to include build maps in DBUTIL report. Default is `N`.

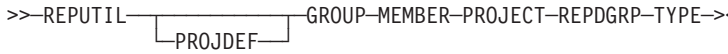
[REPCCODE]
Change Code - Change code to filter on.

[REPLANG]
Language - SCLM language type to filter on.

[REPMODE [D | E]]
Mode - Developer (D) or explorer (E) mode to report on. Default is `D`.

REPUTIL – SCLM DBUTIL report

The DBUTIL service retrieves information from the project database and creates tailored output and a report. It describes the contents of the project database based on the selection criteria you supply.



Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

REPDGRP
SCLM Development Group - The development group of the user.

TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

SAVE – Save SCLM member

This function copies a member from directory WORKAREA/userid/EDIT/ to SCLM under a user's development group. A new member will be created if the member does not exist in the SCLM project hierarchy.

Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

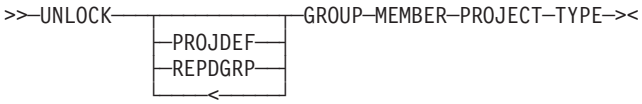
TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

UNLOCK – Unlock SCLM member

This function unlocks a SCLM member that was locked by the EDIT function.



Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

TYPE SCLM Type - The SCLM type containing the selected member.

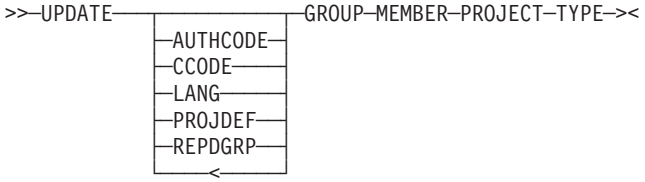
Optional parameters:

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

[REPDGRP]
SCLM Development Group - The development group of the user.

UPDATE – Update SCLM member information

This function updates the member information in SCLM.



Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

[AUTHCODE]
Authorization Code - Authorization code to update the member with.

[CCODE]
Change Code - Change code to update the member with.

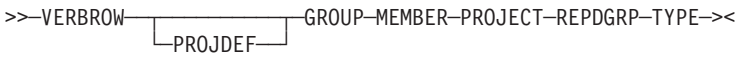
[LANG]
SCLM Language - The SCLM language of the selected member.

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

[REPDGRP]
SCLM Development Group - The development group of the user.

VERBROW – SCLM browse versions

This function browses a version of a member from the version data set.



Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

REPDGRP
SCLM Development Group - The development group of the user.

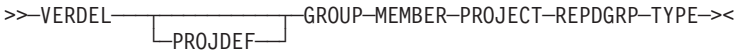
TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

VERDEL – SCLM delete versions

This function deletes the information about a versioned or audited member from SCLM.



This function deletes all older versions of a member in SCLM.

Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

REPDGRP
SCLM Development Group - The development group of the user.

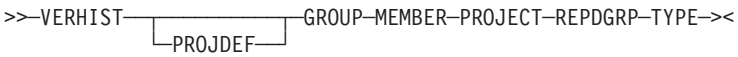
TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

VERHIST – SCLM version history

This function shows the version history of a selected member version in SCLM.



Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

REPDGRP
SCLM Development Group - The development group of the user.

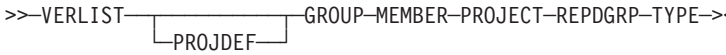
TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

VERLIST – SCLM list versions

This function lists the various versions for a particular member.



Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

REPDGRP
SCLM Development Group - The development group of the user.

TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

VERREC – SCLM recover versions

This function recovers a selected version of a member into the development group.

Required parameters:

GROUP
SCLM Group - The SCLM group where the selected member resides.

MEMBER
SCLM Member - Selected SCLM member.

PROJECT
SCLM Project Name - The SCLM project name.

REPDGRP
SCLM Development Group - The development group of the user.

TYPE SCLM Type - The SCLM type containing the selected member.

Optional parameters:

[PROJDEF]
SCLM Alternate Project Name - The project definition name (defaults to Project).

Samples

A sample Java program (with input and output) is provided, accessing the SCLMDT host services API using an HTTP server as the transport mechanism.

sclmdt_request.xml

The sample request in the following example calls the BUILD function to compile and link the ALLOCEXT member in the SCLMVCM project.

```
<?xml version="1.0"?>
<SCLMDT-INPUT
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="sclmdt.xsd">
 <SERVICE-REQUEST>
  <service>SCLM</service>
  <session>REUSE</session>
  <bldrept>Y</bldrept>
  <groupbld>DEV1</groupbld>
  <projdef>SCLMVCM</projdef>
  <bldmode>C</bldmode>
  <repdgrp>DEV1</repdgrp>
  <sclmfunc>BUILD</sclmfunc>
  <member>ALLOCEXT</member>
  <project>SCLMVCM</project>
  <group>TEST</group>
  <type>SOURCE</type>
 </SERVICE-REQUEST>
</SCLMDT-INPUT>
```

Figure 24. sclmdt_request.xml – sample XML command input file

xmlbld.java

The following example shows a sample Java program (in Eclipse) using the above XML input request. When run, it will do a URL connect to the HTTP server on z/OS (CDFMVS08), passing the XML input stream as a POST request to the `BWBXML` CGI routine.

```
package com.ibm.sclmCaller;

import java.io.*;
import java.net.*;
import java.util.*;

public class XMLBLD {

 public static void main(String[] args) {

  try {

   BufferedReader in = new BufferedReader(new FileReader("C:\\logon.txt"));
   String logon = in.readLine();
   in.close();

   Date d = new Date() ;
   System.out.println("START Transfer DATE/TIME is :" + d.toString()  );

   // URL details for CGI POST
   URL url = new URL("http", "CDFMVS08", 8080, "/BWBXML");
   HttpURLConnection con = (HttpURLConnection) url.openConnection();

   con.setUseCaches(false);
   con.setDoInput(true);
   con.setDoOutput(true);
   con.setRequestMethod("POST");
   con.setRequestProperty( "Authorization", "Basic "
     + Base64Encoder.encode( logon ));

   System.out.println("At url openConnection.. ");

   // POST CGI routines
   doPut(url, con);
   doGet(url, con);

   Date c = new Date() ;
   System.out.println("TOTAL Completion DATE/TIME is :" + c.toString()  );

  }
  catch (IOException exception)
  {
   System.out.println("Error: " + exception);
  }
 }

 public static void doPut(URL url, HttpURLConnection con) throws IOException
 {

   PrintWriter out = new PrintWriter(con.getOutputStream());
   // Below is a sample inline XML input for an ISPF service request
   // This could alternatively be read from an external file

   out.println( "<?xml version=\"1.0\"?>"                                    );
   out.println( "<SCLMDT-INPUT"                                              );
   out.println( "xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\""     );
   out.println( "xsi:noNamespaceSchemaLocation=\"sclmdt.xsd\">"               );
   out.println( "<SERVICE-REQUEST>"                                          );
   out.println( "<service>SCLM</service>"                                    );
   out.println( "<session>NONE</session>"                                    );
   out.println( "<ispprof>IBMUSER.TEST.ISPPROF</ispprof>"                    );
   out.println( "<sclmfunc>BUILD</sclmfunc>"                                 );
   out.println( "<project>SCLMVCM</project>"                                 );
   out.println( "<projdef>SCLMVCM</projdef>"                                 );
   out.println( "<member>ALLOCEXT</member>"                                  );
   out.println( "<group>TEST</group>"                                        );
   out.println( "<type>SOURCE</type>"                                        );
   out.println( "<repdgrp>DEV1</repdgrp>"                                    );
   out.println( "<groupbld>DEV1</groupbld>"                                  );
   out.println( "<bldrept>Y</bldrept>"                                       );
   out.println( "<bldmsg>Y</bldmsg>"                                         );
   out.println( "<bldmode>C</bldmode>"                                       );
   out.println( "</SERVICE-REQUEST>"                                         );
```

Notes:

1. Dependent on Base64Encoder.class for user ID encryption.
2. The file C:\logon.txt contains USERID:PASSWORD.

sclmdt_response.xml

The following example shows the sample output of the BUILD function invoked using the sample Java program.

Figure 26 shows the sample output of the BUILD function invoked using the sample Java program.

Figure 26. sclmdt_response.xml – sample XML output file

```
START Transfer DATE/TIME is :Wed Mar 26 09:44:03 WST 2008
At url openConnection..
About to accept response from Server
Response from Server received
<?xml version="1.0"?>
<SCLMDT-OUTPUT>
<SERVICE-REQUEST>
<service>SCLM</service>
<session>NONE</session>
<ispprof>IBMUSER.TEST.ISPPROF</ispprof>
<sclmfunc>BUILD</sclmfunc>
<project>SCLMVCM</project>
<projdef>SCLMVCM</projdef>
<member>ALLOCEXT</member>
<group>TEST</group>
<type>SOURCE</type>
<repdgrp>DEV1</repdgrp>
<groupbld>DEV1</groupbld>
<bldrept>Y</bldrept>
<bldmsg>Y</bldmsg>
<bldmode>C</bldmode>
</SERVICE-REQUEST>
<SERVICE-RESPONSE>
  <RETURN-CODE>0</RETURN-CODE>
  <REASON-CODES>
   <REASON-CODE ID="BWB00158">SELECT DETAILS-- BUTTON FOR BUILD MESSAGES,
     REPORTS, LISTINGS</REASON-CODE>
  </REASON-CODES>
  <BUILD-MESSAGES>
   <BUILD-MESSAGE ID="FLM42000">- BUILD PROCESSOR INITIATED - 09:53:01 ON
     2008/03/26</BUILD-MESSAGE>
   <BUILD-MESSAGE ID="FLM46000">- BUILD PROCESSOR COMPLETED - 09:53:01 ON
     2008/03/26</BUILD-MESSAGE>
  </BUILD-MESSAGES>
  <BUILD-REPORT>
  <![CDATA[
    ************************************************************************
    ************************************************************************
    **                                                                    **
    **                                                                    **
    **         SOFTWARE CONFIGURATION AND LIBRARY MANAGER (SCLM)           **
    **                                                                    **
    **                      B U I L D    R E P O R T                      **
    **                                                                    **
    **                      2008/03/26   09:53:01                         **
    **                                                                    **
    **                       PROJECT:      SCLMVCM                        **
    **                       GROUP:        DEV1                           **
    **                       TYPE:         SOURCE                         **
    **                       MEMBER:       ALLOCEXT                       **
```

```
 **                    ALTERNATE:    SCLMVCM                       **
 **                    SCOPE:        NORMAL                        **
 **                    MODE:         CONDITIONAL                   **
 **                                                                **
 **                                                                **
 ********************************************************************
 ********************************************************************
1    *** B U I L D   O U T P U T S   G E N E R A T E D ***  Page 1

 MEMBER     TYPE       VERSION      KEYWORD
 ------     ----       -------      -------
          ******* NO MODULES GENERATED *******
1          ******* B U I L D   M A P S   G E N E R A T E D *******  Page 2

                                   (REASON FOR REBUILD)
 MEMBER     TYPE       VERSION      MEMBER     TYPE
 ------     ----       -------      -------    ----
          ***** NO BUILD MAPS GENERATED *****
1    ******* B U I L D   O U T P U T S    D E L E T E D   *******  Page 3

 MEMBER     TYPE       VERSION      KEYWORD
 ------     ----       -------      -------
          ******* NO MODULES DELETED *******
1          ******* B U I L D   M A P S    D E L E T E D   *******  Page 4

                                   (REASON FOR DELETE)
 MEMBER     TYPE       VERSION      MEMBER     TYPE
 ------     ----       -------      -------    ----
          ***** NO BUILD MAPS DELETED *****
]]&#62;
</BUILD-REPORT>
<SCLM-MESSAGES>
 <SCLM-MESSAGE ID="FLM87107">- BUILD SUCCEEDED FOR MEMBER ALLOCEXT AT
   09:53:01, CODE: 0</SCLM-MESSAGE>
</SCLM-MESSAGES>
</SERVICE-RESPONSE>
<OPERATIONS-LOG>
<![CDATA[
Status: 200
Content-Type: text/plain


Entering BWBINT (Service initialization)
Host driver level : V4 12Feb08
09:52:57.48
Function ID timestamp = ID35577
Environment variables :
1 CONTENT_TYPE application/x-www-form-urlencoded
2 QUERY_STRING
3 PATH /usr/lpp/rdz/bin:/bin:.:/usr/sbin:/usr/lpp/internet/bin:/usr/lpp/
   internet/sbin:/usr/lpp/java/J5.0/bin
4 AUTH_TYPE Basic
5 DOCUMENT_URI /BWBXML
6 SHELL /bin/sh
7 HTTPS OFF
8 HTTP_USER_AGENT Java/1.5.0
9 HTTP_ACCEPT text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
10 RULE_FILE //DD:CONF
11 SERVER_PORT 8080
12 CONTENT_LENGTH 554
13 PATH_INFO
14 GATEWAY_INTERFACE CGI/1.1
15 _CEE_RUNOPTS ENVAR("_CEE_ENVFILE=//DD:ENV")
16 REFERER_URL
17 _BPX_SPAWN_SCRIPT YES
18 _ ./BWBCRON1
19 CLASSPATH .:/usr/lpp/internet/server_root/CAServlet
```

```
20 REMOTE_ADDR 192.168.124.236
21 REQUEST_METHOD POST
22 STEPLIB CURRENT
23 CGI_TRANTABLE FEK.#CUST.LSTRANS.FILE
24 LANG C
25 REMOTE_USER IBMUSER
26 LIBPATH /bin:/usr/lpp/internet/bin:/usr/lpp/internet/sbin
27 FSCP IBM-1047
28 SERVER_ADDR 56. 9.42.112.75
29 HTTP_CONNECTION keep-alive
30 PATH_TRANSLATED
31 HTTP_HOST CFDFMVS08
32 SERVER_TOKEN 1
33 HTTP_CACHE_CONTROL no-cache
34 SERVER_SOFTWARE IBM HTTP Server/V5R3M0
35 _BPX_SHAREAS YES
36 NETCP ISO8859-1
37 DOCUMENT_ROOT /usr/lpp/rdz/bin
38 REPORTBITS 77
39 COUNTERDIR NULL
40 LC_ALL en_US.IBM-1047
41 SERVER_PROTOCOL HTTP/1.1
42 _BPX_USERID IBMUSER
43 _SCLMDT_CONF_HOME /etc/rdz/sclmdt
44 HTTPS_KEYSIZE
45 JAVA_HOME /usr/lpp/java/J5.0/
46 TZ EST5EDT
47 SCRIPT_NAME /BWBXML
48 _BPX_BATCH_SPAWN SPAWN
49 _CEE_ENVFILE //DD:ENV
50 NLSPATH /usr/lib/nls/msg/%L/%N:/usr/lpp/internet/%L/%N:/usr/
   lib/nls/msg/En_US.IBM-1047/%N
51 DOCUMENT_NAME /usr/lpp/rdz/bin/BWBXML
52 _SCLMDT_WORK_HOME /var/rdz
53 HTTP_PRAGMA no-cache
54 SERVER_NAME CDFMVS08
Timecheck1:09:52:57.49
Connection Protocol : HTTP
Server Name : CDFMVS08
Server Port : 8080
SCRT check: /var/rdz/WORKAREA/scrtTimeStamp Time:1206492777
   File: 1206492602
Timecheck2:09:52:57.51
Timecheck2b:09:52:57.51
FSCP = IBM-1047
NETCP = ISO8859-1
_SCLMDT_CONF_HOME = /etc/rdz/sclmdt
_SCLMDT_WORK_HOME = /var/rdz
CGI_TRANTABLE = FEK.#CUST.LSTRANS.FILE
Server PATH = /usr/lpp/rdz/bin:/bin:.:/usr/sbin:/usr/lpp/internet/
   bin:/usr/lpp/internet/sbin:/usr/lpp/java/J5.0/bin

Check: userdir = /var/rdz/WORKAREA/IBMUSER
Timechecklsa:09:52:57.51
Timecheck3:09:52:57.51
** Development Group = DEV1
** Selected Group = TEST
Plugin version : NONE
Host   version : 4.1.0
Temporary data set prefix set to : SCLMVCM.IBMUSER

Timecheck4:09:52:58.04
Parameters to be written to temporary data set 'SCLMVCM.IBMUSER.SCLMDT.
   VCMISPF.ID35577' :ISPPROF=IBMUSER.TEST.ISPPROF&SCLMFUNC=BUILD
   &PROJECT=SCLMVCM&PROJDEF=SCLMVCM&MEMBER=ALLOCEXT&GROUP=TEST
   &TYPE=SOURCE&REPDGRP=DEV1&GROUPBLD=DEV1&BLDREPT=Y&BLMSG=Y&BLDMODE=C
/etc/rdz/sclmdt;/var/rdz
```

```
/usr/lpp/rdz/bin:/bin:.:/usr/sbin:/usr/lpp/internet/bin:/usr/lpp/internet/
   sbin:/usr/lpp/java/J5.0/bin/~~FEK.#CUST.LSTRANS.FILE

Processing SCLM request :
Value for SESSFLG = NONE
Value for SESSION = NONE
Value for SCLMFUNC = BUILD
Value for PROJ = SCLMVCM
Value for PROJDEF = SCLMVCM
Value for Development GROUP = DEV1
Value for Selected GROUP = TEST
Value for TYPE = SOURCE
Value for LANG = NONE
Value for MEMBER = ALLOCEXT
Value for J2EEFILE = NONE
Value for PROFDSN = IBMUSER.TEST.ISPPROF
Value for PARMS = NONE
pcnt = 3
parmline.1 = ISPF SELECT CMD(BWBBLD ID35577 SCLMVCM.IBMUSER SCLMVCM SCLMVCM
   TEST DEV1 SOURCE ALLOCEXT /) NEST LANG(CREX)
time check before ISPZINT call :09:52:58.17
time check after ISPZINT call :09:53:02.51
ispzint rc = 0
check: respline count = 226
***  ISPZINT OUTPUT ***
Content-type: text/plain

Entering ISPZINT (Service initialization)
About to read from fileno(stdin) = 0
Data read from STDIN is ISPF SELECT CMD(BWBBLD ID35577 SCLMVCM.IBMUSER SCLMVCM
 SCLMVCM TEST DEV1 SOURCE ALLOCEXT /) NEST LANG(CREX)
EPOCH secs = 1206492778
Local Date & time: Tue Mar 25 20:52:58 2008
Hour: 20 Min: 52 Sec 58
Function ID timestamp = ID075178
Environment variables:
0 QUERY_STRING=
1 CONTENT_TYPE=application/x-www-form-urlencoded
2 PATH=/usr/lpp/rdz/bin:/bin:.:/usr/sbin:/usr/lpp/internet/bin:/usr/lpp/
   internet/sbin:/usr/lpp/java/J5.0/bin
3 AUTH_TYPE=Basic
4 DOCUMENT_URI=/BWBXML
5 SHELL=/bin/sh
6 HTTPS=OFF
7 HTTP_ACCEPT=text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
8 HTTP_USER_AGENT=Java/1.5.0
9 SERVER_PORT=8080
10 RULE_FILE=//DD:CONF
11 GATEWAY_INTERFACE=CGI/1.1
12 PATH_INFO=
13 CONTENT_LENGTH=554
14 _CEE_RUNOPTS=ENVAR("_CEE_ENVFILE=//DD:ENV")
15 _BPX_SPAWN_SCRIPT=YES
16 REFERER_URL=
17 _=/u/SCLMDTIS/bin/ISPZINT
18 CLASSPATH=.:/usr/lpp/internet/server_root/CAServlet
19 STEPLIB=CURRENT
20 REQUEST_METHOD=POST
21 REMOTE_ADDR=192.168.128.236
22 CGI_TRANTABLE=FEK.#CUST.LSTRANS.FILE
23 LANG=C
24 LIBPATH=/bin:/usr/lpp/internet/bin:/usr/lpp/internet/sbin
25 REMOTE_USER=IBMUSER
26 SERVER_ADDR=9.42.112.75
27 FSCP=IBM-1047
28 PATH_TRANSLATED=
29 HTTP_CONNECTION=keep-alive
```

```
  30 SERVER_TOKEN=1
  31 HTTP_HOST=CDFMVS08
  32 _BPX_SHAREAS=YES
  33 SERVER_SOFTWARE=IBM HTTP Server/V5R3M0
  34 HTTP_CACHE_CONTROL=no-cache
  35 REPORTBITS=77
  36 DOCUMENT_ROOT=/usr/lpp/rdz/bin
  37 NETCP=ISO8859-1
  38 COUNTERDIR=NULL
  39 LC_ALL=en_US.IBM-1047
  40 _SCLMDT_CONF_HOME=/etc/rdz/sclmdt
  41 _BPX_USERID=IBMUSER
  42 SERVER_PROTOCOL=HTTP/1.1
  43 JAVA_HOME=/usr/lpp/java/J5.0/
  44 HTTPS_KEYSIZE=
  45 TZ=EST5EDT
  46 _CEE_ENVFILE=//DD:ENV
  47 _BPX_BATCH_SPAWN=SPAWN
  48 SCRIPT_NAME=/BWBXML
  49 NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lpp/internet/%L/%N:
/usr/lib/nls/msg/En_US.IBM-1047/%N
  50 _SCLMDT_WORK_HOME=/var/rdz
  51 DOCUMENT_NAME=/usr/lpp/rdz/bin/BWBXML
  52 SERVER_NAME=CDFMVS08
  53 HTTP_PRAGMA=no-cache
  Number of environment variables is 54
  Connection Protocol = HTTP
  Server Name  = CDFMVS08
  Server Port  = 8080
  ***ERROR: Unable to get status information for ISPZTSO
  FSCP = IBM-1047
  NETCP = ISO8859-1
  Server PATH = /usr/lpp/rdz/bin:/bin:.:/usr/sbin:/usr/lpp/internet/bin:/
     usr/lpp/internet/sbin:/usr/lpp/java/J5.0/bin/
  ISPF standalone function invoked
  ISPF COMMAND = ISPF SELECT CMD(BWBBLD ID35577 SCLMVCM.IBMUSER SCLMVCM
     SCLMVCM TEST DEV1 SOURCE ALLOCEXT /) NEST LANG(CREX)
  ISPF PROFILE = NONE
  Re-usable ISPF session =
  About to spawn task for ISPZTSO
  Parameters passed to ISPZTSO - PROFILE
  Return code from ISPZTSO is 0
  About to process PROFILE data in /tmp/IBMUSER.ID075178.ISPF.SYSTSPRT
  About to malloc() 252 bytes for profdat
  Temporary data set prefix set to : SCLMVCM
  About to call bpxwdyn to allocate VCMTEMP
  Allocating data set SCLMVCM.ISPF.VCMISPF.ID075178 to the VCMTEMP DD
  1024 bytes of ISPF SELECT CMD(BWBBLD ID35577 SCLMVCM.IBMUSER SCLMVCM
     SCLMVCM TEST DEV1 SOURCE ALLOCEXT /) NEST LANG(CREX)
     written to VCMTEMP
  1024 bytes of /etc/rdz;/var/rdz written to VCMTEMP
  1024 bytes of /usr/lpp/rdz/bin:/bin:.:/usr/sbin:/usr/lpp/internet/bin:
     /usr/lpp/internet/sbin:/usr/lpp/java/J5.0/bin/~~
     written to VCMTEMP
  Parameter to be passed to ISPZTSO CALL *(ISPZCNT) 'ISPF ID075178 SCLMVCM
     NONE ISPF SELECT CMD(BWBBLD ID35577 SCLMVCM.IBMUSER SCLMVCM
     SCLMVCM TEST DEV1 SOURCE ALLOCEXT /) NEST LANG(CREX)'
  About to spawn task for ISPZTSO
  Parameters passed to ISPZTSO - CALL *(ISPZCNT) 'ISPF ID075178 SCLMVCM
     NONE ISPF SELECT CMD(BWBBLD ID35577 SCLMVCM.IBMUSER SCLMVCM
     SCLMVCM TEST DEV1 SOURCE ALLOCEXT /) NEST LANG(CREX)'
  Return code from ISPZTSO is 0
  About to open /tmp/IBMUSER.ID075178.ISPF.SYSTSPRT
  IKJ56003I PARM FIELD TRUNCATED TO 100 CHARACTERS
  Entering ISPZCNT (ISPF Initialization)
  Parameters ISPF ID075178 SCLMVCM NONE ISPF SELECT CMD(BWBBLD ID35577
     SCLMVCM.IBMUSER SCLMVCM SCLMVCM TEST DEV1
```

```
REC: ispmlib=isp.sispmenu
Allocation successful for ISPMLIB
REC: isptlib=isp.sisptenu
Allocation successful for ISPTLIB
REC: ispplib=isp.sisppenu
Allocation successful for ISPPLIB
REC: ispslib=bzz.sbzzsenu,isp.sispsenu,isp.sispslib
Allocation successful for ISPSLIB
REC: ISPF_timeout = 300
NOTE: Data set allocations took 0.28 elapsed seconds
Running user ISPF command: ISPSTART CMD(BWBBLD ID35577 SCLMVCM.IBMUSER
   SCLMVCM SCLMVCM TEST DEV1 SOURCE ALLOCEXT /) NEST
<ISPINFO>
ISPF COMMAND : ISPSTART CMD(BWBBLD ID35577 SCLMVCM.IBMUSER SCLMVCM
   SCLMVCM TEST DEV1 SOURCE ALLOCEXT /) NEST LANG(CREX) N
<ISPF>
Entering BWBBLD
Temporary data set prefix : SCLMVCM.IBMUSER
about to allocate VCMISPF
rc from alloc is 0
Translate table allocated successfully
SCLMDT CONFIG directory = /etc/rdz/sclmdt
SCLMDT Working directory = /var/rdz
SCLMDT Translate table = FEK.#CUST.LSTRANS.FILE

******   BUILD PARMS   ******
PROJECT = SCLMVCM
PROJDEF (Project Name) = SCLMVCM
GROUP (SCLM Group) = TEST
GROUPBLD (Build at group) = DEV1
TYPE (SCLM Type) = SOURCE
MEMBER (SCLM Build Member) = ALLOCEXT
BLDSCOPE (Build Scope) = N
BLDMODE (Build Mode) = C
BLDLIST (Listing Flag ) = Y
BLDREPT (Report Flag ) = Y
BLDMSG (Messages Flag ) = Y
BLDLSTDS (Listing Data set name) =
BLDRPTDS (Report Data set name) =
BLDMSGDS (Messages Data set name) =
BLDEXTDS (Exit Data set name) =
SUBMIT (Submission type) = ONLINE
******   End PARMS   ******

projfile = /etc/rdz/sclmdt/CONFIG/PROJECT/SCLMVCM.conf
Security Flag set to N
rc from FLMMSGS alloc is 0
PARMS=SCLMVCM,SCLMVCM,DEV1,SOURCE,ALLOCEXT,,N,C,Y,Y,,tmpmsg,
   tmprpt,tmplst,BLDEXIT
BWBBLD CHECK: PARMS = SCLMVCM,SCLMVCM,DEV1,SOURCE,ALLOCEXT,,
   N,C,Y,Y,,tmpmsg,tmprpt,tmplst,BLDEXIT
*** BUILD SUCCESSFUL ***

Cleaning up workarea directories


**********************  SCLM MESSAGES  **********************
FLM87107 - BUILD SUCCEEDED FOR MEMBER ALLOCEXT AT 09:53:01, CODE: 0
************************************************************

*** XML-NOTE *** Reference tagged SERVICE-RESPONSE
</ISPF>
RC=0
</ISPINFO>
ISPRC = 0
********************** ISPLOG CONTENTS **********************
1  Time *** ISPF transaction log *** Userid: IBMUSER Date: 08/03/26 Page:
```

```
    09:53   Start of ISPF Log - - -  - Session # 1 ----------------------
    09:53  TSO - Command  -  - BWBBLD ID35577 SCLMVCM.IBMUSER SCLMVCM
            SCLMVCM TEST DEV1 SOURCE ALLOCEXT /
    09:53  TSO - Command  -  - FLMCMD
    09:53                       BUILD,SCLMVCM,SCLMVCM,DEV1,SOURCE,ALLOCEXT
            ,,N,C,Y,Y,,tmpmsg,tmprpt,tmplst,BLD
    09:53                       EXIT
    09:53  End of ISPF Log - - - - - Session # 1 ----------------------
  ********************* END ISPLOG CONTENTS *********************
  IKJ56247I FILE ISPLLIB NOT FREED, IS NOT ALLOCATED
  Leaving ISPZCNT
  READY
  END
   return code from ISPFInit = 0
   PROCESSING COMPLETE
  End of SCLM Processing
  FUNC: BUILD - Cleaning up old 'SCLMVCM.IBMUSER.SCLMDT.VCMISPF.ID35577'
  EXIT BWBINT 09:53:02.86 WITH RC=0
]]&#62;
</OPERATIONS-LOG>
</SCLMDT-OUTPUT>
TOTAL Completion DATE/TIME is :Wed Mar 26 09:44:11 WST 2008
```

Appendix D. Rational Application Developer for WebSphere Software Build utility

Overview of Rational Application Developer for WebSphere Software Build utility

This section covers enhancements to the Java/J2EE build process using Rational Application Developer for WebSphere Software Build utility (previously known as AST: Application Server Toolkit). Rational Application Developer for WebSphere Software Build Utility is delivered as part of Rational Application Developer (RAD). RAD must be ordered, installed, and configured separately. The installation and customization of this product is not described in this manual. Refer to the documentation delivered with RAD for installation and customization instructions for the Build Utility function. This section will subsequently refer to Rational Application Developer for WebSphere Build utility as "the Build utility".

The Build utility allows replicated project workspaces from the IDE that have been stored in SCLM to be built by SCLM using headless mode Eclipse on z/OS.

This section is to be used in conjunction with the SCLM Developer Toolkit online user guide and the installation and customization guide supplied with the product. SCLM Developer Toolkit provides Java/J2EE language translators to SCLM to enable full Java/J2EE build capability.

SCLM Developer Toolkit not only provides functionality to store JAVA/J2EE source and objects but through these translators enables SCLM to build and fully control Java/J2EE applications. The Java/J2EE objects supported through these language translators are Java classes, Java archive files (JAR), Enterprise Java Beans (EJB jar files), Web archive files (WAR) and Enterprise archive files (EAR).

SCLM Developer Toolkit integrates with the Build utility to control SCLM J2EE Projects being replicated in the Build utility workspace under UNIX System Services on z/OS. The replicated projects are then built by ARCHDEF build processing which reference Build utility scripts. Sample build scripts are shipped with SCLM Developer Toolkit. Objects resulting from build/compilation are stored back into SCLM.

The Build utility supports builds of all J2EE projects that are supported under Rational Application Developer V7. Projects that were built on previous versions of RAD will have to be migrated into a RAD V7 workspace on the IDE Client before adding them to SCLM. Normal Eclipse Java projects are also supported.

Storing Java/J2EE objects in SCLM

Java/J2EE objects are stored in SCLM as follows:

- Java Source is compiled into classes. Classes are stored in SCLM under type JAVACLAS. The short name and the long name are stored in Translate tables.
- Support for EJB and regular JAR files (contains Classes and may contain other Java project components such as XML/HTML/JSP in a packaged structure). JAR files stored in SCLM type J2EEJAR.
- Support for WAR files assembled based on J2EE web.xml file in J2EE project. WAR files stored in SCLM type J2EEWAR.

- Support for EAR files created for deployment based on application.xml in J2EE project. EAR files stored in SCLM type J2EEEAR.
- Deployment support of EAR files into Websphere Application server (WAS) on z/OS.
- SQLJ build support
- All output listings are stored in SCLM type J2EELIST.

The Build utility compared to native ANT build process

SCLM Developer Toolkit ships with a native ANT build process for Java/J2EE support. Refer to the SCLM Developer Toolkit online user guide plug-in and installation/customization guide.

The Rational Application Developer for WebSphere Software Build utility enhances the Java/J2EE build support by fully replicating the Eclipse IDE environment on z/OS when building.

As in the ANT build process, the Build utility process uses the archdef, which contains members that make up the Java/J2EE project, and are a short name representation of how the project exists in an Eclipse workspace.

The user may choose either the Build utility process or the native ANT build process by assigning the build script (referenced by the ARCHDEF) with the appropriate language translator:

- J2EEAST for the Build utility
- J2EEANT for the native ANT build

The archdef when built invokes the pre-build verify language translator (J2EEAST). This language type J2EEAST is assigned to the build script referenced in the archdef by the SINC keyword. This Language translator copies all archdef components into the z/OS UNIX System Services file system to be built using the Build utility.

Project components and Java source may either be stored in SCLM as EBCDIC or ASCII. Under the Build utility, text components and Java source are by default translated into ASCII in the USS workspace before build processing occurs.

Rational Application Developer for WebSphere Software Build utility Installation notes

Rational Application Developer for WebSphere Software Build Utility is delivered as part of Rational Application Developer (RAD). RAD must be ordered, installed, and configured separately. The installation and customization of this product is not described in this manual. Refer to the documentation delivered with RAD for installation and customization instructions for the Build Utility function. The Build utility directory package will reside in the z/OS UNIX file system.

J2EE build processing may require large region sizes to avoid ‘out of memory’ or storage errors. To cater for large online builds specify a minimum REGION=512M on the Developer for System z RSE daemon (by default this is the RSED started task). For batch processing it is recommended to have this region size parameter specified on the batch JOBCARD.

SCLM integration with the Build Utility

Initial J2EEAST processing is similar to J2EEANT where the language translator determines from the archdef which parts are required to be rebuilt depending on the build mode (conditional or forced). Project source files and included generated objects are then copied into the UNIX Systems services file system workarea for the Build utility. The run script and build XML that are stored in SCLM under type J2EEBLD are also copied into the same workarea. The run script is invoked to bring up Eclipse in headless mode on z/OS and to run the referenced application build XML. The Build utility compiles and generates the required Java/J2EE objects specified by the run script, build XML and archdef.

J2EEAST examines the generated objects and selects for SCLM update only those parts/objects that were required to be rebuilt according to SCLM from the build mode (conditional/forced).

SCLM processes each individual archdef component running each language translator associated with the component. The Language translator JAVA associated with each selected Java source member copies the class files associated with each back into SCLM.

The final part of build processing is processing the ARCHDEF itself where the language translator J2EEOBJ executes. This archdef translator determines what J2EE objects have generated (JAR, WAR, EAR) and copies these parts back into SCLM.

Rational Application Developer for WebSphere Software Build utility implementation and usage

Implementation and usage take place as follows:

- By default SCLM Developer Toolkit comes with archdef and build script wizards that generate project archdefs and associated build scripts. These generate native ANT build process scripts and assign the build scripts with a language of J2EEANT. SCLM Developer Toolkit provides a checkbox on the SCLM Build Preference page to enable these wizards to generate build scripts instead.
- The **Team > SCLM Preferences > Build Script Options** page should have each build script sample (Java, EJB, WAR, EAR, SQLJ) associated with a run script of BWBASTR.
- The run script (sample BWBASTR) must be copied into each SCLM project that requires the Build utility. The script must reside in SCLM type J2EEBLD and be of language type TEXT or J2EEPART. This script must be customized. Instructions for doing so are detailed within the run script itself. Refer to the Build utility scripts and formats for more information on run script BWBASTR.
- Each project component (JAR, WAR, EAR) requires both an archdef (refer to the Build utility scripts for archdef format layout) and a corresponding build script (refer to section 2.4 for build script layouts). These build scripts must be created in an SCLM type of J2EEBLD and be defined with a language type of J2EEAST. These build scripts and archdefs may be created by the user in SCLM or the user may generate these by use of the Client archdef and build script wizards, using **TEAM > Generate Java/J2EE build script** or **TEAM > Add to SCLM**.
- Compilation or component build failure details will be returned within the build operations log. If a build fails, you can locate compilation errors by finding the "COMPILE LISTING" in the operations log. If the log indicates an error relating to the configuration or installation of the Build utility, notify the Systems administrator. Eclipse build failure logs can be found in the UNIX System

Services file system under directory AST_INSTALL/Eclipse/configuration. These specific error logs will be unreadable on z/OS as they will be in ASCII format. They must be translated to EBCDIC or transferred in binary mode to the Windows® desktop to browse.

- If running J2EE builds in batch mode it is recommended to include a region size parameter REGION=512M on the batch jobcard. Smaller region sizes may result in 'out of memory' storage problems.

SCLM Build utility language translators

For the Build utility, certain language translators are assigned to the J2EE source, ARCHDEF, and build script.

Samples for generating these languages within an SCLM project are supplied with SCLM Developer Toolkit in the installation sample library, SBWBSAMP.

Java source
Language = JAVA | JAVABIN

J2EE text
Language = J2EEPART | J2EEBIN (for example, XML)

J2EE binary
Language = J2EEBIN (for example, jpg)

build script
Language = J2EEAST

run script
Language = TEXT | J2EEPART

ARCHDEF
Language = archdef

Note: The LKED=J2EEOBJ keyword will process the archdef with language translator J2EEOBJ.

J2EEAST: J2EE Verify/Build Translator

This is a language translator of the Build utility script XML.

Summary: SAMPLE: BWBTRAN4

The J2EEAST Verify translator is invoked when the archdef is built. J2EEAST is the language type of the J2EEBLD build XML referenced by the SINC keyword in the archdef. This verify translator copies selected source and all of the archdef objects regardless of build mode into the z/OS HFS workarea to be referenced at build time as the Project workspace. This project workspace will temporarily exist under the SCLM Developer Toolkit workarea. SCLM DT creates a temporary project workspace under the following directory structure: `/var/SCLMDT/WORKAREA/userid/project/group/type/member/project`

The Build utility requires the full project structure and so all Project parts are copied into the UNIX System Services file system. All text-based components are copied into the project workspace as ASCII. The build XML is also copied into the HFS workarea. The build XML contains project details and J2EE build requests as well as other property details referenced by SCLM.

The build XML references an associated Build utility run script. This run script is also copied into the workarea (EBCDIC), and when it is run, brings up Eclipse in

headless mode on z/OS to run the application build XML. The current implementation will result in a full build in the workspace, though SCLM will only process parts selected if conditional build was requested and the translator determines which objects generated are required to be updated in SCLM, according to the build mode.

J2EE objects generated such as JAR, WAR or EAR archive files will be stored back into SCLM when the archdef translator J2EEOBJ runs. Selected class files are logged and these are updated in SCLM when the archdef processes the individual Java components (Java translator).

J2EEOBJ: J2EE ARCHDEF translator

This is an archdef language translator referenced by LKED keyword.

Summary: SAMPLE: BWBTRAN3

This is the final build translator invoked as part of the archdef build process. This translator determines what J2EE objects were previously built in translator J2EEAST and copies these objects into SCLM with the generated shortname provided.

1. If SCLM_BLDMAP variable set, retrieve Build map information and update J2EE object in META-INF directory.
2. Copy these J2EE objects (JAR, WAR, EAR) into SCLM with the associated short name.
   - JAR/EJB into SCLM type J2EEJAR
   - WAR into SCLM type J2EEWAR
   - EAR into SCLM type J2EEEAR
3. Copy the build log detail files into SCLM type J2EELIST

JAVA: Java language translator (EBCDIC)

This is a language translator of Java source stored in EBCDIC.

Summary: SAMPLE: BWBTRAN1

The language type for Java source is 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 may 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 code page conversions need to take place when migrating or importing projects from the client to the host.

JAVABIN: Java language translator (ASCII)

This is a language translator of Java source stored in ASCII.

Summary: SAMPLE: BWBTRAN1

Language type that is similar to Java and is used when storing Java source as ASCII in SCLM.

J2EEPART: J2EE text language translator

This is a language translator of J2EE text part stored in EBCDIC.

Summary: SAMPLE: BWBTRANJ

J2EEPART is a 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 may 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 may be used.

J2EEBIN: J2EE binary language translator

This is a language translator of Java source stored in EBCDIC.

Summary: SAMPLE: BWBTRANJ

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 may be generically slotted under this language definition if no particular build parsing is required.

Build utility Build scripts and formats

As with the traditional J2EE build service method the Build utility method involves the Build utility scripts being referenced by the ARCHDEF by the SINC keyword.

The invoked build script is a Build utility application to build XML which would be customized and unique for each J2EE project. This build script also references a Build utility RUN script which contains global Build utility properties and Eclipse runtime commands to bring up Eclipse on z/OS in headless mode and run the selected build XML. Generally the one customized BWBASTR script would be used by all Build utility build scripts. The language type for all Build utility build scripts must be J2EEAST. The language type for the BWBASTR script should be a text only language translator such as TEXT or J2EEPART.

The Build utility run script (BWBASTR) , the sample build scripts for Java/Jar project (BWBASTJ), EJB project (BWBASTEJ), Application Client project (BWBASTAP), Web project (BWBASTW), EAR project (BWBASTE) may be copied from the SCLM Developer Toolkit Build utility sample library into the customers SCLM projects under type of J2EEBLD.

Build script format

The build script format includes the following:

SCLM_ARCHDEF
Name of referenced archdef being built

AST_SHSCRIPT
Name of AST RUN script to run AST on z/OS. (see sample BWBASTR)

PROJECT NAME
The J2EE project name (not the SCLM project name)

JAR_FILE_NAME
For Java project the name of the generated JAR

WAR_NAME
For Web project the name of the generated WAR

EJB_NAME
For EJB project the name of the generated JAR

EAR_NAME
For Enterprise application the name of the generated EAR that may be deployed

SCLM_BLDMAP
If YES, include in MANIFEST directory in JAR, WAR, and EAR. Provides audit and build map of parts included.

Also included are the AST ANT routines to generate the required project. The SCLM required build scripts are modified versions of the AST supplied samples below:

- ProjectImport
- ProjectBuild
- Jar
- EJBexport
- WARexport
- EARexport

Unmodified AST routines

The routines below have been extracted from the Application Server Toolkit guide.

projectImport: This task imports an existing file system project into a workspace.

Table 13. projectImport parameters

Attribute	Description	Required
ProjectName	Name of project to be imported	Yes
ProjectLocation	The fully qualified location of the project (either under the workspace, or elsewhere on the file system).	No, default is ${workspaceLocation}/ ${projectName}

Example: Import a project which is under the workspace directory, but not presently in the workspace:

```
<projectImport
ProjectName="myProject"/>
```

projectBuild: This task builds the specified project.

Table 14. projectBuild parameters

Attribute	Description	Required
ProjectName	Name of project to be built	Yes
BuildType	Type of build	No, default is Incremental. Can be Incremental or Full.
FailOnError	Whether or not builds should fail on error	No, default is true.
DebugCompilation	Whether on not compilations should be debug	No, default is true.
Quiet (deprecated)	Whether or not to print out messages	No, default is false.
ShowErrors	Whether or not to show the project errors in the ant build log	No, default is true.
SeverityLevel	The problem level to count and treat as a build error	No, Default is ERROR. May be ERROR or WARNING or INFORMATION.

Table 14. projectBuild parameters (continued)

Attribute	Description	Required
CountValidationErrors	Whether or not to count Validation problems as project Errors	No, default is true.
PropertyCountName	Property to receive the project error count	No, Default is ProjectErrorCount .
PropertyMessagesName	Property to receive the project error messages	No, Default is ProjectErrorMessages.

Examples:

- Build ″myProject″. Default is incremental with debug information:

```
<projectBuild ProjectName="myProject" />
```

- Do a production build of ″myProject″, a full build without debug information:

```
<projectBuild
ProjectName="myProject"
failonerror="true"
DebugCompilation="false"
BuildType="full" />
<echo message="projectBuild: projectName=${projectName}
project Error Count=${ProjectErrorCount}
project Error Messages=${ProjectErrorMessages}" />
```

EJBexport: This task performs the same operation as the EJB JAR file export wizard for exporting an EJB Project to an EJB Jar file. This task is not available in products that do not include EJB development tools.

Table 15. EJBexport parameters

Attribute	Description	Required
EJBProjectName	Name of the EJB Project (Case Sensitive)	Yes
EJBExportFile	Absolute path of the EJB JAR file.	Yes
ExportSource	Whether to include source files or not.	No, default is false.
Overwrite	Whether to overwrite if the file already exists.	No, default is false.

Example: Export the project ″EJBProject″ to ″EJBProject.jar″

```
<ejbExport
EJBProjectName="EJBProject"
EJBExportFile="EJBProject.jar"/>
```

WARExport: This task performs the same operation as the WAR file export wizard for exporting a Web Project to a WAR file.

Table 16. WARExport parameters

Attribute	Description	Required
WARProjectName	Name of the Web Project (Case Sensitive)	Yes
WARExportFile	Absolute path of the WAR file	Yes
ExportSource	Whether to include source files or not.	No, default is false.
Overwrite	Whether to overwrite if the file already exists.	No, default is false.

Example: Export the project ″ProjectWeb″ to ″ProjectWeb.war″

```
<warExport WARProjectName="ProjectWeb" WARExportFile="ProjectWeb.war"/>
```

EARExport: This task performs the same operation as the WAR file export wizard for exporting a Web Project to a WAR file.

Table 17. EARExport parameters

Attribute	Description	Required
EARProjectName	Name of the Enterprise Application Project (Case Sensitive)	Yes
EARExportFile	Absolute path of the EAR file	Yes
ExportSource	Whether to include source files or not.	No, default is false.
IncludeProjectMetaFiles	Whether to include the project meta-data files that include the Java build path, the project names, etc. Used when reimporting as binary projects.	No, default is false.
Overwrite	Whether to overwrite if the file already exists.	No, default is false.

Example: Export the project ″EARProject″ to ″EARProject.ear″

```
<earExport EARProjectName="EARProject" EARExportFile="EARProject.ear"/>
```

AppClientExport: This task performs the same operation as the WAR file export wizard for exporting a Web Project to a WAR file.

Table 18. AppClientExport parameters

Attribute	Description	Required
AppClientProjectName	Name of the Application Client Project (Case Sensitive)	Yes
AppClientExportFile	Absolute path of the Application Client JAR file	Yes
ExportSource	Whether to include source files or not.	No, default is false.
Overwrite	Whether to overwrite if the file already exists.	No, default is false.

Example: Export the project ″ProjectClient″ to ″ProjectClient.jar″:

```
<appClientExport
AppClientProjectName="ProjectClient"
AppClientExportFile="ProjectClient.jar"/>
```

AST Run Script (sample BWBASTR): The AST Run Script is referenced by AST build scripts (type=J2EEBLD lang=TEXT).

```
/*

 This is a sample build script to build J2EE projects using
 Application Server toolkit (AST) on z/OS.
 AST is the headless mode Eclipse builder which is a separate
 installable item outside of SCLM Developer toolkit.
 This sample script will need customizing and should reside in the
 SCLM type of J2EEBLD being invoked from a J2EE archdef member via
 the SINC archdef keyword. It should be stored with a text language
 such as TEXT or J2EEPART.

 The BUILDFILE and WORKSPACE arguments are passed internally from
 the J2EE AST language translator within SCLM Developer Toolkit

*/

parse arg $args
parse var $args BUILDFILE WORKSPACE

/*------------------ User Customization ----------------------------*/
```

```
/* Customize the following variable settings : AST_DIR and JAVA_DIR */

/* AST_DIR is the z/OS eclipse directory where AST is installed     */
AST_DIR='/u/AST/eclipse'
/* JAVA_DIR is the appropriate z/OS Java bin directory              */
/* Set to the Java bin that is packaged in the AST delivery         */
JAVA_DIR='/u/AST/eclipse/jdk/bin'

/*------------------ End user customization ------------------------*/
/* The rest of the sample should not have to be modified            */

say 'AST Eclipse directory = 'AST_DIR
say 'Java bin directory = 'JAVA_DIR
say 'BUILDFILE = 'BUILDFILE
say 'WORKSPACE = 'WORKSPACE

If SUBSTR(WORKSPACE,1,1) /= '/' Then
 Do
  say 'ERROR : Invalid WORKSPACE ... Build terminated'
  exit 8
 End

/* The following parameters are set for headless Eclipse invocation */
/* Java memory parameter options of min 256M & max 512M set         */
/* Increase max memory parameter if Java memory failure             */

M_parm   = '-Xms256m -Xmx512m'
A_parm   = '-Dfile.encoding=ISO8859-1 -Xnoargsconversion'
D_parm   = '-Dwtp.autotest.noninteractive=true'
cp_parm  = '-cp 'AST_DIR'/startup.jar org.Eclipse.core.launcher.Main'
ap_parm  = '-application com.ibm.etools.j2ee.ant.RunAnt'
w_parm   = '-data "'workspace'"'
f_parm   = '-f 'buildfile

PARMS = M_parm' 'A_parm' 'D_parm' 'cp_parm' 'ap_parm' 'w_parm' 'f_parm

/* Java dumping options have been turned off to prevent java dumps */
/* on memory allocation failures                                    */

JAVA_NODUMP='export JAVA_DUMP_OPTS="ONANYSIGNAL(NONE)"'
JAVA_CMD  = JAVA_DIR'/java 'PARMS
AST_BUILD = JAVA_NODUMP' ; 'JAVA_CMD

say "Invoking java launch of Eclipse ..."
say AST_BUILD
say "Executing ..."

AST_BUILD
ASTrc = rc
If ASTrc = 23 Then
  say 'WARNING: UNINITIALIZED workspace'
  Else
  If ASTrc = 15 Then
    say 'ERROR :  WORKSPACE is already BEING USED'

If ASTrc /= 0 then
 Do
  say 'ERROR : BUILD FAILED - return code = 'ASTrc
  EXIT 8
 End

EXIT 0
```

Java/JAR Build Script (sample): The following build Script is type=J2EEBLD lang=J2EEAST. It may be any name (up to 8 chars). Variables defined in standard XML format.

```
<!--

BWBASTJ: J2EE JAR Sample for AST

This is a sample build XML script to be run using AST Eclipse
headless mode on z/OS.
The SCLM Build script will be copied into the appropriate workarea
directory on the z/OS USS filesystem.
The projectlocation keyword will be overlaid dynamically with the
appropriate assigned workspace. Ensure that the projectlocation line
is left as is :

Customize the project and property variables below

   project name  : Change ASTJAR to the project name of the Java
                   application
   AST_SHSCRIPT  : Name of skeleton AST run script for build
   SCLM_ARCHDEF  : Name of archdef to be built
   JAR_FILE_NAME : Name of created JAR
   SCLM_BLDMAP   : If YES then include in MANIFEST directory in JAR.
                   Provides audit and build map of parts included

-->


<project name="ASTJAR" default="init" basedir=".">
<property name="AST_SHSCRIPT" value="ASTRUN"/>
<property name="SCLM_ARCHDEF" value="JAR1"/>
<property name="JAR_FILE_NAME" value="ASTjar.jar"/>
<property name="SCLM_BLDMAP" value="NO"/>
<target name="init">

     <projectImport projectname="${ant.project.name}"
      projectlocation="$WORKSPACE" />
     <Eclipse.refreshLocal resource="${ant.project.name}" depth="infinite" />
     <echo message="refresh done" />


      <projectBuild ProjectName="${ant.project.name}" failonerror="true"
        DebugCompilation="true" BuildType="full" />


       <jar update="true" destfile="${JAR_FILE_NAME}">
       <fileset dir="." excludes="**/*.java"/>
       </jar>

  </target>
</project>
```

WAR Build Script (sample): The following Build Script is type=J2EEBLD lang=J2EEAST.

```
<!--

BWBASTW: J2EE Sample for AST

This is a sample build XML script to be run using AST Eclipse
headless mode on z/OS.
The SCLM Build script will be copied into the appropriate workarea
directory on the z/OS USS filesystem.
The projectlocation keyword will be overlaid dynamically with the
appropriate assigned workspace. Ensure that the projectlocation line
is left as is :
```

```
Customize the project and property variables below

   project name  : Change ASTWAR to the project name of the WAR
                   application
   AST_SHSCRIPT  : Name of skeleton AST run script for build
   SCLM_ARCHDEF  : Name of archdef to be built
   WAR_NAME      : Name of created WAR
   SCLM_BLDMAP   : If YES then include in MANIFEST directory
                   in WAR.
                   Provides audit and build map of parts included

-->


<project name="ASTWAR" default="init" basedir=".">
<property name="AST_SHSCRIPT" value="BWBASTR"/>
<property name="SCLM_ARCHDEF" value="WAR1"/>
<property name="WAR_NAME" value="ASTwar.war" />
<property name="SCLM_BLDMAP" value="NO"/>
<target name="init">

     <projectImport projectname="${ant.project.name}"
      projectlocation="$WORKSPACE" />
     <Eclipse.refreshLocal resource="${ant.project.name}" depth="infinite" />
     <echo message="refresh done" />


      <projectBuild ProjectName="${ant.project.name}" failonerror="true"
        DebugCompilation="true" BuildType="full" />

      <warExport WARProjectName="${ant.project.name}"
       ExportSource="true"
       WARExportFile="${WAR_NAME}"/>

  </target>
</project>
```

EJB Build Script (sample): The following Build Script is type=J2EEBLD lang=J2EEAST.

```
<!--

BWBASTEJ: J2EE EJB Sample for AST

This is a sample build XML script to be run using AST Eclipse
headless mode on z/OS.
The SCLM Build script will be copied into the appropriate workarea
directory on the z/OS USS filesystem.
The projectlocation keyword will be overlaid dynamically with the
appropriate assigned workspace. Ensure that the projectlocation line
is left as is :

Customize the project and property variables below

   project name  : Change ASTEJB to the project name of the EJB
                   application
   AST_SHSCRIPT  : Name of skeleton AST run script for build
   SCLM_ARCHDEF  : Name of archdef to be built
   EJB_NAME      : Name of created EJB JAR
   SCLM_BLDMAP   : If YES then include in MANIFEST directory
                   in JAR, WAR, or EAR.
                   Provides audit and build map of parts included

-->
```

```
<project name="ASTEJB" default="init" basedir=".">
<property name="AST_SHSCRIPT" value="ASTRUN"/>
<property name="SCLM_ARCHDEF" value="EJB1"/>
<property name="EJB_NAME" value="ASTejb.jar" />
<property name="SCLM_BLDMAP" value="NO" />
<target name="init">

     <projectImport projectname="${ant.project.name}"
      projectlocation="$WORKSPACE" />
     <Eclipse.refreshLocal resource="${ant.project.name}" depth="infinite" />
     <echo message="refresh done" />


      <projectBuild ProjectName="${ant.project.name}" failonerror="true"
        DebugCompilation="true" BuildType="full" />

      <ejbExport ExportSource="true"
      EJBProjectName="${ant.project.name}"
      EJBExportFile="${EJB_NAME}"/>

  </target>
</project>
```

EAR Build Script (sample): The following Build Script is type=J2EEBLD lang=J2EEAST.

```
<!--

BWBASTE: J2EE EAR Sample for AST

This is a sample build XML script to be run using AST Eclipse
headless mode on z/OS.
The SCLM Build script will be copied into the appropriate workarea
directory on the z/OS USS filesystem.
The projectlocation keyword will be overlaid dynamically with the
appropriate assigned workspace at build time.
Ensure that the projectlocation line is left as is.

Customize the project and property variables below

   project name  : Change ASTEAR to the project name of the EAR
                   application
   AST_SHSCRIPT  : Name of skeleton AST run script for build
   SCLM_ARCHDEF  : Name of archdef to be built
   EAR_NAME      : Name of created EAR
   SCLM_BLDMAP   : If YES then include in MANIFEST directory
                   in JAR, WAR, or EAR.
                   Provides audit and build map of parts included

-->


<project name="ASTEAR" default="init" basedir=".">
<property name="AST_SHSCRIPT" value="BWBASTR"/>
<property name="SCLM_ARCHDEF" value="EAR1"/>
<property name="EAR_NAME" value="ASTear.ear"/>
<property name="SCLM_BLDMAP" value="NO"/>
<target name="init">

     <projectImport projectname="${ant.project.name}"
      projectlocation="$WORKSPACE" />
     <Eclipse.refreshLocal resource="${ant.project.name}" depth="infinite" />
     <echo message="refresh done" />


      <projectBuild ProjectName="${ant.project.name}" failonerror="true"
        DebugCompilation="false" BuildType="full" />
```

```
      <earExport EARProjectName="${ant.project.name}"
       ExportSource="true"
       EAREXportFile="${EAR_NAME}"/>

  </target>
</project>
```

J2EE ARCHDEF format: The format for the ARCHDEF is the same as for the normal J2EEANT build process.

SINC Source includes the J2EEBLD build script.

INCLD
SCLM includes J2EE component (for example, Java source).

INCL SCLM include another archdef

OUT1 Indicates the J2EE object type created by this archdef:

- J2EEEAR
- J2EEWAR
- J2EEJAR

LIST Is the summary component listing and audit of the ARCHDEF built. Contained in TYPE=J2EELIST under archdef member name.

LKED Indicates LEC archdef and gives the language of the archdef translator to be invoked (for J2EE archdefs this is always J2EEOBJ).

```
SINC  SCRIPT1      J2EEBLD
INCLD XX000001   SOURCE
INCL  PAULWAR2  ARCHDEF

OUT1 * J2EEEAR
LIST   * J2EELIST
LKED   J2EEOBJ
```

SQLJ build support

SCLM provides SQLJ support through the Java/J2EE build translators shipped with SCLM Developer Toolkit. These translators in conjunction with the AST build scripts provide support to store and build sqlj projects. They also provide integration points to allow customer routines to set db2 sqlj customization properties and to support db2 bind processing through the SCLM build and promote steps via the SCLM build and promote exits.

For additional information on SQLJ support reference the SCLM Developer Toolkit users guide.

SQLJ support in SCLM (using the AST method) can be summarized as such:

- Storing SQLJ source in SCLM and assigning the source a language type of SQLJ
- Building a Java/J2EE archdef which may include SQLJ members as part of the archdef. The archdef will reference a SQLJ build script in type J2EEBLD which will contain sqlj properties to be set and customized by the user (see sample scripts BWBASTSQ and BWBASTSE).
- Optional post DB2 bind processing (using generated and SCLM stored DBRM files) The post bind processing to be driven by build and promote user exits. The DBRM files accessed directly within the SCLM controlled datasets.
- Deploying the built JAR/EAR through SCLM DT deployment processing

SQLJ customization (for the SCLM Administrator)

System requirements: DB2 sqlj support is installed. The following db2 directory (or similar depending on site install directory) should reside in the z/OS USS file system: `/usr/lpp/db2/db2810/*` (this is the db2 v8 directory). This directory will be required for SQLJ build script customization. For SQLJ, classpath dependencies will exist on `/usr/lpp/db2/db2810/jcc/classes/sqlj.zip`. For `db2sqljcustomize`, the classpath dependencies will exist on `/usr/lpp/db2/db2810/jcc/classes/db2jcc.jar`.

Language translators:

SQLJ: A new language translator SQLJ is provided which should be assigned as the language type to all SQLJ source code stored in SCLM. The new translator requires additional SCLM Types to be defined. The new translator is similar to the JAVA translator but contains additional IOTYPE definitions for output SCLM types SQLJSER and DBRMLIB. If the customer is not generating DBRM files as part of the db2sqljcustomize step then this DBRMLIB IOTYPE may be removed from the SQLJ language definition. (reference the SQLJ sample translator definition BWBTRANS). Generate within the project PROJDEF the new SQLJ translator and the additional types, as follows:

- SQLJSER: A type required to contain the generated serialized profile files (.ser files) created/customized in the sqlj and db2sqljcustomize steps. 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 db2sqljcustomize step. This type is only required for customers using generated DBRM files as part of their DB2 bind processing. Recommended to define this SCLM type dataset as recfm=VB , lrecl= 256.

SQLJ user customization

SQLJ build property script: Customers need to define SQLJ DB2 properties in the main SQLJ build script (sample BWBASTSQ) which will reside in type J2EEBLD. These properties will be used in the “sqlj” and “db2sqljcustomize” steps.

Note: The supplied SQLJ skeleton BWBASTSQ is just an extension of the normal Java build skeleton BWBASTJ. Also BWBASTSE is a sqlj extension of the EJB sample BWBASTEJ. Customers can mix SQLJ & Java source members in the same archdef and use BWBASTSQ . The resultant build will create a JAR file containing all generated class and .ser files.

Below is a list of required “sqlj” and “db2sqljcustomize” property definitions.

General property settings

```
<!-- specify the JAVA bin runtime location -->
<property name="JAVA_BIN" value="/usr/lpp/java/J5.0/bin"/>
```

Sqlj

```
<!-- specify the location of the sqlj & db2sqljcustomize exec routine
     bwbsqlc.rex (sample BWBSQLC) -->
<!-- By default this sample should be located or copied to the
     SCLM DT install directory -->
<property name="sqlj.exec" value="/etc/SCLMDT/bwbsqlc.rex"/>

<!-- specify global property arguments below for sqlj processing -->
<property name="sqlj.arg" value="-compile=false -status
     -linemap=NO -db2optimize"/>
```

db2sqljcustomize

```
<!-- specify the db2jcc.jar location to be included in the
     db2sqljcustomize classpath -->
<property name="db2sqljcustomize.cp" value="/usr/lpp
  /db2/db2810/jcc/classes/db2jcc.jar":./SRC:/usr/lpp/db2810/jcc/
     classes/db2jcc_license_cisuz.jar"/ >

<!-- specify global property arguments below for db2sqljcustomize -->
<property name="db2sqljcustomize.arg" value='-automaticbind NO -onlinecheck
     YES -staticpositioned YES -bindoptions "ISOLATION(CS)" -genDBRM'/>


<!--   Below is the temporary property file name to be passed to a
     User property routine for storing additional argument properties
     It requires no tailoring -->

<property name="db2sqljcustomize.propfile" value="user.properties"/>

<!-- Below is the name of an optional user program which will be run
     immediately before the db2sqljcustomize process.
It dynamically updates a property file db2sqljcustomize.propfile to be
     used as input to the db2sqljcustomize command
 The db2sqljcustomize routine (property: sqlj.exec) will concatenate
     and use both argument properties set in this build.xml
(db2sqljcustomize.arg) and the argument properties read from the
     user property file.
     The user routine will be passed the following arguments
     basedir  : Base directory which is the workspace directory
     propfile : The name of the property file to create (temporary
                file to be created in workspace)
     sqljf    : All the .ser file names to be processed by db2sqljcustomize

     Note: The property file being created needs to be basedir'/'propfile
           The properties should be set in the file in the following format
           argument=value (eg: singlepkgname=longname:pkgname)
           (one argument declaration per line)
eg:
 singlepkgname=src/TeSQLJ986_SJProfile0.ser:SQLJ986
singlepkgname=src/TeSQLJ987_SJProfile0.ser:SQLJ987
pkgversion=1
url=jdbc:db2://site1.com:80/MVS01
qualifier=DBT

     If no User program is required specify :
     <property name="sqlj.userpgm" value="NONE"/>

-->

<property name="sqlj.userpgm" value="/u/userdir/BWBSQLD"/>

***  end of property settings ***
```

The argument properties would be used for building up the required argument string in the SQLJ or 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}
```

SCLM Archdef (sample ASTSQLJ):

```
*
*
 LKED  J2EEOBJ            * J2EE Build translator
```

```
*
* Source to include in build
*
 INCLD XX000001 ASTSQLJ   * .classpath
 INCLD XX000002 ASTSQLJ   * .project
 INCLD XX000103 ASTSQLJ   * .runtime
 INCLD BU000082 ASTSQLJ   * build.xml
 INCLD DE000155 ASTSQLJ   * deploy.xml
 INCLD SQ000001 ASTSQLJ   * SQLJAntScripts/sqlj.customize.xml
 INCLD SQ000002 ASTSQLJ   * builder/sqlJava.java
 INCLD SQ000003 ASTSQLJ   * builder/sqlJava.sqlj
 INCLD TE000137 ASTSQLJ   * Tester.java
 INCLD SQ000004 ASTSQLJ   * builder/sqlJava_SJProfile0.ser
*
* Build script and generated outputs
*
 SINC  ASTSQLJ  J2EEBLD   * J2EE JAR Build script
 OUT1  *        J2EEJAR
 LIST  *        J2EELIST
```

SCLM AST build script (sample ASTSQLJ):

```
<project name="ASTSQLJ" default="compile" basedir=".">
<property name="AST_SHSCRIPT" value="BWBASTR"/>
<property name="SCLM_ARCHDEF" value="ASTSQLJ"/>
<property name="JAR_FILE_NAME" value="ASTSQLJ.jar"/>

<!-- specify the JAVA bin runtime location -->
<property name="JAVA_BIN" value="/u/java/J5.0/bin"/>

<!-- specify the db2jcc.jar location to be included in the
     db2sqljcustomize classpath -->
 <property name="db2sqljcustomize.cp" value="/usr/lpp/products/db2/db2810/jcc/
     classes/db2jcc.jar:.:/usr/lpp/products/db2/db2810/jcc/classes/
     db2jcc_license_cisuz.jar"/>

 <!-- specify global property arguments below for sqlj processing -->
 <property name="sqlj.arg" value="-compile=false -status -linemap=NO -db2optimize"/>

 <!-- specify global property arguments below for db2sqljcustomize -->
 <property name="db2sqljcustomize.arg" value='-automaticbind NO -onlinecheck YES
   -bindoptions "ISOLATION(CS)" -genDBRM'/>

 <!-- specify the location of the db2sqljcustomize exec routine BWBSQLC -->
<property name="db2sqljcustomize.exec" value="/u/SCLMDT/bwbsqlc.rex&cdqg;/>

<!-- Below is the name of the user program to be run as part of the
     db2sqljcustomize process . It dynamically updates a property file
     to be used as input to the db2sqljcustomize command  -->
<property name="sqlj.userpgm" value="NONE"/>

<target description="Pre-Compile SQLJ Files" name="sqlj">
 <echo> SQLJ TRANSLATOR  </echo>
 <apply executable="java" skipemptyfilesets="true" type="file"  failonerror="true"
     logerror="true">
   <arg line="-Dfile.encoding=ISO8859-1 -Xnoargsconversion"/>
   <arg line="sqlj.tools.Sqlj"/>
   <arg line="${sqlj.arg}"/>
   <!-- SER  Files -->
   <fileset dir=".">
    <patternset>
     <include name="**/*.sqlj"/>
    </patternset>
   </fileset>
  </apply>
</target>
```

```
<target description="Customize SQLJ Files" name="db2sqljcustomize" depends="sqlj">
     <!-- Below just echoes properties into a property file db2 props ->
 <apply executable="${db2sqljcustomize.exec}" skipemptyfilesets ="true"
     parallel="true" type="file" failonerror="true" relative="true">
   <arg value="${basedir}"/>
   <arg value="${db2sqljcustomize.cp}"/>
   <arg value="${sqlj.userpgm}"/>
   <arg value="${db2sqljcustomize.propfile}"/>
   <arg value="db2sqljcustomize ${db2sqljcustomize.arg}"/>
   <arg value="sqlj-source"/>
   <!-- SER  Files -->
   <fileset dir=".">
    <patternset>
     <include name="**/*.ser"/>
    </patternset>
   </fileset>
  </apply>
</target>

<target name="compile" depends="db2sqljcustomize">

     <projectImport projectname=&odq;ASTSQLJ&cdqg;
      projectlocation="$WORKSPACE" />
     <eclipse.refreshLocal resource=&odq;ASTSQLJ&cdqg;depth="infinite" />
     <echo message="refresh done" />


      <projectBuild ProjectName=&odq;ASTSQLJ&cdqg;failonerror="true"
        DebugCompilation="true" BuildType="full" />

     <jar update="true" destfile="${JAR_FILE_NAME}">
     <fileset dir="." excludes="**/*.java"/>
     </jar>

  </target>
  </project>
```

Java source in archive files

By default Java source is copied into the USS workspace as ASCII before compilation builds take place. This occurs even if Java source is stored in SCLM as EBCDIC.

If the user requests Java source to be included in the archive file (JAR) then the source will by default be in ASCII. It is however possible to configure the build scripts so that Java source is copied into the workspace and compiled in EBCDIC format so that if Java source inclusion is desired it will be in EBCDIC format.

To include Java source in EBCDIC format the following build scripts must be modified accordingly, as follows:

- In the main build xml script include the line `<property name="ASTJAVA_ENCODING" value="EBCDIC"/ >` and remove the excludes keyword from the JAR update routine, as follows: `jar update="true" destfile="${JAR_FILE_NAME}"> <fileset dir="." excludes="**/*.java"/>`.
- In the AST run script referenced by keyword AST_SHSCRIPT remove the keyword –Dfile.encoding=ISO8859-1 A_parm = '-Dfile.encoding=ISO8859-1 -Xnoargsconversion'

USAGE SCENARIO: 'PlantsByWebSphere'

In WebSphere Application Server there are a number of sample projects. These can either be deployed using prebuilt EARs, or assembled using an ANT build process. Here is an example using SCLM Developer Toolkit to check in this project to SCLM and build/deploy on z/OS using AST as the build process.
PlantsByWebSphere is a J2EE project that implements an online shopping store for plants. It is made up of the following four components:

- Enterprise Archive (PBWProject)
- Enterprise JavaBean (PlantsByWebSphereEJB)
- Web Archive 1 (PlantsByWebSphereWEB)
- Web Archive 2 (PlantsGalleryWEB)

This document describes a scenario whereby this source is placed into an Eclipse project structure, checked into Developer Toolkit, and built and deployed on the host.

1. ADMINISTRATIVE TASK: Add the following types to the project definition:
   - PLANTEAR - Type for the EAR
   - PLANTEJB - Type for the EJB
   - PLANTWE1 - Type for the 1st WAR
   - PLANTWE2 - Type for the 2nd WAR
2. ADMINISTRATIVE TASK: Rebuild the project (submit job).
3. ADMINISTRATIVE TASK: Allocate the following datasets:
   - <HLQ>.<DEVGROUP>.PLANTEJB
   - <HLQ>.<DEVGROUP>.PLANTEAR (large)
   - <HLQ>.<DEVGROUP>.PLANTWE1 (large)
   - <HLQ>.<DEVGROUP>.PLANTWE2 (large)
4. Locate PlantsByWebSphere source code on a WAS installation. (~root/AppServer/samples/src/PlantsByWebSphere)
5. Start up the Developer Toolkit-enabled Eclipse product of your choice.
6. Import the 4 components into 4 projects in your Eclipse workspace.
7. Ensure .project and .settings files are included and set correctly for each project.
8. Set compiler settings and other Websphere-specific configuration files. (Project... Properties)
   - Set Java facet to 1.4
   - Ensure correct WAS runtime stub is associated with project
9. ADD to SCLM (Team->Add to SCLM on Project node)
   a. Languages to use are the following:
      - Images – J2EEBIN
      - Xml files – J2EEPART
      - Settings – J2EEPART
      - Java sourcecode – JAVA
   b. Types to use are the following:
      - EAR files – PLANTEAR
      - EJB files – PLANTEJB
      - WAR 1 files – PLANTWE1
      - WAR 2 files – PLANTWE2

c. Archdef (last page of Add to SCLM Wizard)
   - Each component gets an archdef with J2EE keywords and AST build script.
   - EAR Archdef should include other components' Archdefs.

   PLANTEAR ARCHDEF (sample):

```
*
*
 LKED  J2EEOBJ            * J2EE Build translator
*
* Source to include in build
*
 INCL  PLANTWE1 ARCHDEF
 INCL  PLANTWE2 ARCHDEF
 INCL  PLANTEJB ARCHDEF
*
 INCLD XX000002 PLANTEAR  * .project
 INCLD OR000005 PLANTEAR  * .settings/org.Eclipse.wst.common.component
 INCLD OR000006 PLANTEAR  * .settings/org.Eclipse.wst.common.project.fa
                         * cet.core.xml
 INCLD BU000147 PLANTEAR  * EarContent/build.xml
 INCLD BU000148 PLANTEAR  * EarContent/Database/PLANTSDB/build.xml
 INCLD MA000028 PLANTEAR  * EarContent/META-INF/MANIFEST.MF
 INCLD AP000004 PLANTEAR  * EarContent/META-INF/application.xml
 INCLD IB000007 PLANTEAR  * EarContent/META-INF/ibm-application-bnd.xmi
 INCLD IB000008 PLANTEAR  * EarContent/META-INF/ibm-application-ext.xmi
 INCLD WA000004 PLANTEAR  * EarContent/META-INF/was.policy
 INCLD PL000017 PLANTEAR  * EarContent/Database/PLANTSDB/PLANTSDB.zip
 INCLD OR000001 PLANTEAR  * .settings/org.Eclipse.core.resources.prefs
 INCLD SE000049 PLANTEAR  * EarContent/META-INF/ibmconfig/cells/default
                         * Cell/security.xml
 INCLD VA000001 PLANTEAR  * EarContent/META-INF/ibmconfig/cells/default
                         * Cell/applications/defaultApp/deployments/de
                         * faultApp/variables.xml
*
* Build script and generated outputs
*
 SINC  PLANTEAR J2EEBLD   * J2EE EAR Build script
 OUT1  *        J2EEEAR
 LIST  *        J2EELIST
```

d. Build scripts, with the following output models to be named:
   - PlantsByWebSphere.ear (PLANTEAR)
   - PlantsByWebSphereEJB.jar (PLANTEJB)
   - PlantsByWebSphere.war (PLANTWE1)
   - PlantsGallery.war (PLANTWE2)

   PLANTEAR BUILDSCRIPT (sample):

```
<project name="PBWProject" default="init" basedir=".">
<property name="AST_SHSCRIPT" value="ASTRUN"/>
<property name="SCLM_ARCHDEF" value="PLANTEAR"/>
<property name="EAR_NAME" value="PlantsByWebSphere.ear"/>
<target name="init">
     <projectImport projectname="PBWProject"
      projectlocation="$WORKSPACE" />
     <Eclipse.refreshLocal resource="PBWProject" depth="infinite" />
     <echo message="refresh done" />
      <projectBuild ProjectName="PBWProject" failonerror="true"
        DebugCompilation="false" BuildType="full" />
      <earExport EARProjectName="PBWProject"
       EARExportFile="${EAR_NAME}"/>
  </target>
  </project>
```

PLANTWE1 requires the EJB to be on the classpath at build time. Therefore, modify PLANTWE1's build script, to add a CLASSPATH_JARS_FILES property, value "PlantsByWebSphereEJB.jar".

10. Build project (through Archdef PLANTEAR)
11. Run deployment:
    a. Prepare deployment action script (deploy_ben.jacl). This deployment requires a datasource/connector/mail provider to be created in WAS so it requires a tailored deployment script.

```
#-------------------------------------------------------------------
#
#  Sample JACL script for J2EE application deployment
#
#-------------------------------------------------------------------
#
# IBM SCLM Developer Toolkit uses the IBM WebSphere Application Server
# wsadmin tool to deploy J2EE applications to WAS running on z/OS. The
# wsadmin tool requires a JACL script to guide the deployment process.
# Hence the JACL script must be installed under UNIX Systems services
# (USS) before the deployment process can be invoked.
# This sample JACL script should require no customization.

source /u/WebSphere/V6R0/AppServer/samples/bin/AdminUtil.jacl

proc ex1 {args} {

   #-------------------------------------------------------------------
   # set arguments
   #-------------------------------------------------------------------

   set app        [lindex $args 0]
   set appName    [lindex $args 1]
   set cellName   [lindex $args 2]
   set nodeName   [lindex $args 3]
   set serverName [lindex $args 4]

   #-------------------------------------------------------------------
   # set up globals
   #-------------------------------------------------------------------
   global AdminConfig
   global AdminControl
   global AdminApp


   #-------------------------------------------------------------------
   #     -- was a earfile name supplied
   #-------------------------------------------------------------------
   if {[llength $app] == 0} {
      puts "deploy: Error -- No application specified."
      return
   }

   #-------------------------------------------------------------------
   #     -- was the appname supplied
   #-------------------------------------------------------------------
   if {[llength $appName] == 0} {
      puts "deploy: Error -- Application name not specified."
      return
   }



  #-------------------------------------------------------------------
  # Create J2C Resource Adapter
  #-------------------------------------------------------------------
```

```
createJ2CResourceAdapter $nodeName $serverName

#--------------------------------------------------------------------
# Setup security cell
#--------------------------------------------------------------------
set secAuthAlias "$cellName/samples"
set secDescript  "JAAS Alias for WebSphere Samples"
set secUserID    "samples"
set secPassword  "s1amples"
createJAASAuthenticationAlias $cellName $secAuthAlias $secDescript
   $secUserID $secPassword


#--------------------------------------------------------------------
# Create JDBC Provider
#--------------------------------------------------------------------

set templName    "Cloudscape JDBC Provider (XA)"
# All Samples that need JDBC Provider should use/share this one
set provName     "Samples Cloudscape JDBC Provider (XA)"
createJDBCProvider $nodeName $serverName $templName $provName

#--------------------------------------------------------------------
# Create Datasource
#--------------------------------------------------------------------

set templName    "Cloudscape JDBC Driver XA DataSource"
set dsName       "PLANTSDB"
set dsJNDI       "jdbc/PlantsByWebSphereDataSource"
set dsDesc       "Data source for the Plants by WebSphere entity beans"
set dsAuthMech   "BASIC_PASSWORD"
set dbName       "\${APP_INSTALL_ROOT}/\${CELL}/PlantsByWebSphere.ear/
   Database/PLANTSDB"
set secAuthAlias "N_O_N_E"
set connAttrs    "upgrade=true"
createDatasource $nodeName $serverName $provName $templName $dsName
   $dsJNDI $dsDesc $dsAuthMech $dbName $secAuthAlias $connAttrs

#--------------------------------------------------------------------
# Create Connection Factory (use builtin_rra)
#--------------------------------------------------------------------
set dsName       "PLANTSDB"
set cfName       "PLANTSDB_CF"
set cfAuthMech   "BASIC_PASSWORD"
set secAuthAlias "N_O_N_E"
set cfi          "javax.resource.cci.ConnectionFactory"
createConnectionFactory $nodeName $serverName $provName $dsName $cfName
     $cfAuthMech $secAuthAlias $cfi

#--------------------------------------------------------------------
# Create Mail Session
#--------------------------------------------------------------------
set provName "Built-in Mail Provider"
set msName "PlantsByWebSphere Mail Session"
set jndiName "mail/PlantsByWebSphere"
set mailTransportHost "yourcompany.ComOrNet"
set mailFrom "userid@yourcompany.ComOrNet"
createMailSession $cellName $nodeName $provName $msName $jndiName
    $mailTransportHost $mailFrom

  #-----------------------------------------------------------------
  # Setup options for the deployment
  # Additinal options can be added here as required
  # For Example:
  #  lappend app_options -update
  #  lappend app_options -appname MyAppName
```

```
#  lappend app_options -contextroot MyAppName
#  lappend app_options -preCompileJSPs
#  lappend app_options -defaultbinding.force
# for a full list of options please use the AdminApp command
# wsadmin [return]
# $AdminApp options            - generic options
#      or
# $AdminApp options MyApp.ear  - valid options for your ear file
# lappend app_options -node WXP-KEFA25B
#-------------------------------------------------------------
puts "deploy: installing the application"

set app_options [list -server $serverName]
lappend app_options -node $nodeName
lappend app_options -verbose
lappend app_options -usedefaultbindings
lappend app_options -deployejb
lappend app_options -deployejb.dbtype DERBY_V10
#-------------------------------------------------------------
# Install the application onto the server
#-------------------------------------------------------------
$AdminApp install $app $app_options


#-------------------------------------------------------------
# Save all the changes
#-------------------------------------------------------------
puts "deploy: saving the configuration"
$AdminConfig save


#-------------------------------------------------------------
# Start the installed application
#-------------------------------------------------------------
puts "Starting the application..."
set appManager [$AdminControl queryNames cell=$cellName,
  node=$nodeName,type=ApplicationManager,
  process=$serverName,*]

$AdminControl invoke $appManager startApplication $appName
puts "Started the application successfully..."

puts "deploy: done."
}

#-------------------------------------------------------------
# Main
#-------------------------------------------------------------
if { !($argc == 5) } {
   puts "deploy: This script requires 5 parameter: ear file name,
     application name, cell name, node name and server name"
   puts "e.g.:     deploy  /WebSphere/AppServer/installableApps/
          jmsample.ear myapp1 myCell myNode myServer"
} else {
   set application      [lindex $argv 0]
   set appName          [lindex $argv 1]
   set cellName         [lindex $argv 2]
   set nodeName         [lindex $argv 3]
   set serverName       [lindex $argv 4]

   ex1 $application $appName $cellName $nodeName $serverName
}
```

b. Generate property script on front end, and initiate deployment. (Deploy skeleton)

12. Verify deployment messages. Assuming success, browse the project on your WAS server.

Appendix E. BUILD FORGE and SCLM

This section covers the implementation and configuration issues for accessing and using SCLM through Build Forge®. The section includes customization and testing samples for both SCLM builds and promotes.

Overview

The Build Forge console server generally will reside on a windows platform and will need to be configured with an SCLM service call in the console project area. This SCLM configured project when started will connect to the Build Forge agent server on z/OS which must already have been started. The RDz Build Forge plug-in enables console projects to be started from the RDz environment by socket connecting to the console server . Output from completed project runs can also be accessed from the RDz Build Forge plug-in within RDz.

Prerequisites

- Build Forge V7.1 plug-in installed on RDz 7.6
- Build Forge console/server V7.1
- Build forge agent for z V7.1 (src-bfagent-7.1.1.0-0-0022.tar.gz or later)
  http://www-01.ibm.com/support/docview.wss?&rs=3099&uid=swg24016541

Note: It is important that the correct version of the Build Forge plugin is used with the correct version of the server and z/OS agent. The compatible versions of the plugin and z agent are packaged within the main Build Forge server distributed package.

To upload the correct version of the plugin, update from your server location http://<console_host_name>/prism/eclipse/updateSite/site.xml

How to invoke the Build Forge agent on z/OS

Start Buildforge agent on z/OS. The default agent port is 5555. For example:

```
bfagent -s -f /install_directory/bfagent-7.1.1.007/src/bfagent.conf
```

```
start BF console -> go to servers
select hostname
(hostname:port)
test connection   (using z/OS id authentication)
```

Sample response is as follows:

```
Agent Test Initiated
Host:hostname Port:5555
Agent Version: 7.1.1.007
Authentication: userid
Platform:  os/390 18.00 03
Functional Test: OK
Agent Test Completed (Duration 9s)
```

```
Set up a project in the BF console and run.
```

Build Forge console server configuration

The following minimum configuration is required to enable SCLM functionality within Build Forge.

1. Server (Configure server to point to the Build Forge agent for System z)
   a. Set up Server authorization with z/OS userid / password. Click **Server -> Server Auth**

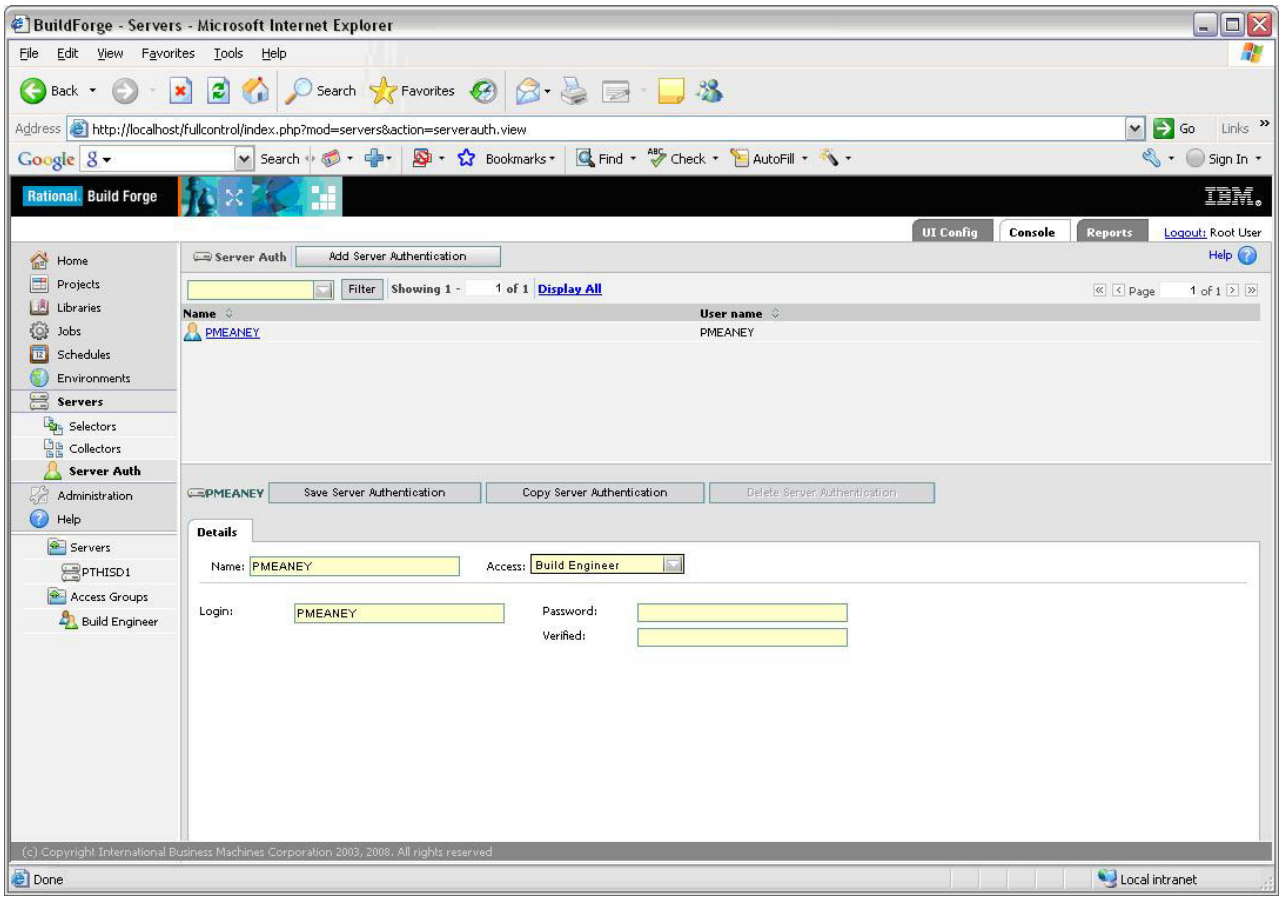


Figure 27. Server authorization

This Userid must be a valid userid and password existing on the z/OS system, where the agent is running and where you intend to run the SCLM service. The access group will be an existing access group or default access group residing in the console server with the appropriate privileges: **Administration -> access groups**.

   b. Set up main z/OS Server configuration. (Click **Servers**.)

NAME: Identifiable unique name for your server definition
PATH: This is the path directory on the z/OS host where the build directories will be created. The build directories will usually be of the form PATH/Project name/BUILD_#/*
The SCLM input file will be created here and the resulting output response from the service call stored here according to the naming convention specified in the console Project definition.
HOST: Enter the Destination host name (DNS or IP address)

and port (z/OS agent default 5555) Hostname:port
ACCESS: Enter the authorization access type
(For example: Build Engineer)

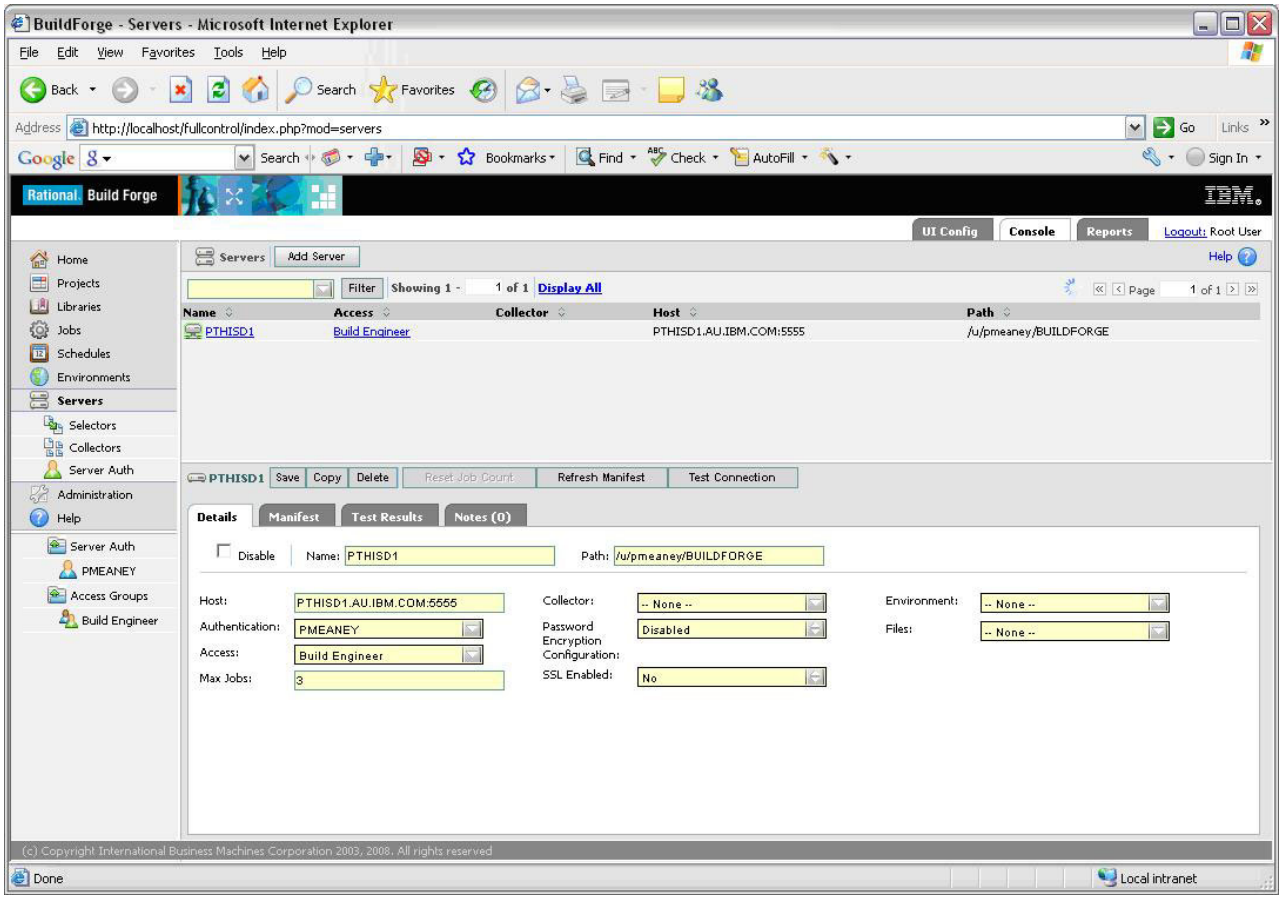


Figure 28. Server definition

Test connection to host by clicking **Test Connection** . This will do a connection test to the server agent running on z/OS. Expect a successful test as follows:

```
Agent Test Initiated
Host:pthisd1.au.ibm.com Port:5555
Agent Version: 7.1.1.007
Authentication: IBMUSER
Platform:  os/390 18.00 03
Functional Test: OK
Agent Test Completed (Duration 9s)
```

c. Set up main Selector for Server. Click **Servers -> Selector**.

Assign BF_NAME to your server name. (For example: PTHISD1)

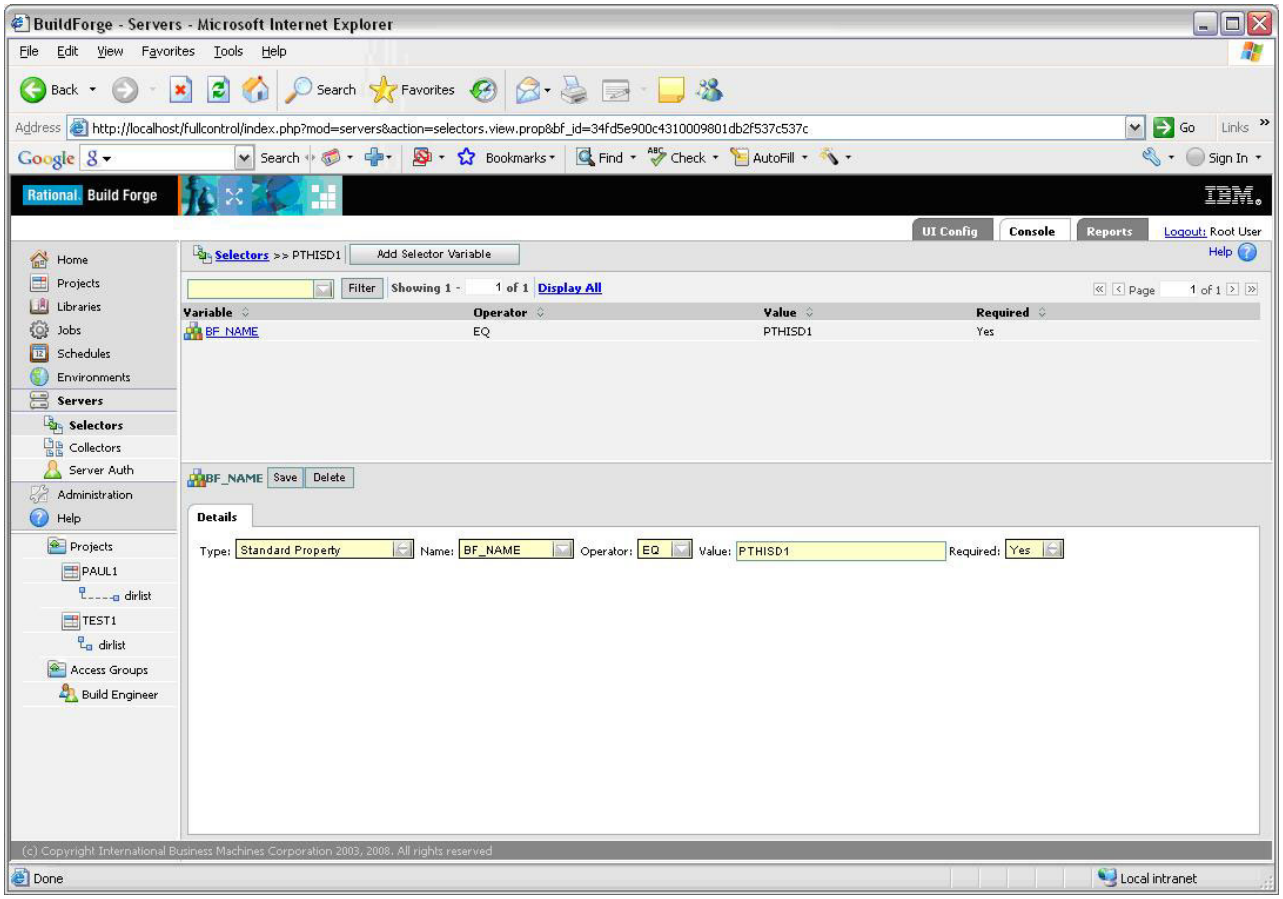


Figure 29. Server selector definition

2. Environment variable setup (Configure SCLM Developer toolkit environment variables for SCLM project.)
   a. Click **Environments** – Create a name for the SCLM DT environment variable list. For example: SCLMDT

      Configure the following SCLM DT environment variables:

      STEPLIB: The STEPLIB dataset where the SCLM Developer Toolkit modules reside. This will be located in the RDz SFEKLOAD dataset.
      (For example: RD4Z.V760.SFEKLOAD) This action type in the definition should be APPEND.
      _CMDSERV_BASE_HOME: The base directory where the ISPF gateway is installed (For example: /usr/lpp/ispf) .
      _CMDSERV_CONF_HOME: The configuration directory for the ISPF gateway (For example: /etc/ispf) .
      _CMDSERV_WORK_HOME: The work directory for the ISPF gateway (For example: /var/ispf) .
      _SCLMDT_CONF_HOME: The configuration directory for SCLM DT within RDz. (For example: /etc/rd4z760/sclmdt).
      _SCLMDT_WORK_HOME: The work directory for SCLM DT within RDz. Generally make this the same as the _CMDSERV_WORK_HOME variable
      (For example: $_CMDSERV_WORK_HOME).

PATH: The following directory paths -- RDz bin, the _CMDSERV_BASE_HOME bin, and the RDz sclmdt script directory. This should be action type APPEND.
For example:
/apc/trdz750/usr/lpp/rdz/bin:/etc/rd4z760/sclmdt/CONFIG/script:
$_CMDSERV_BASE_HOME/bin

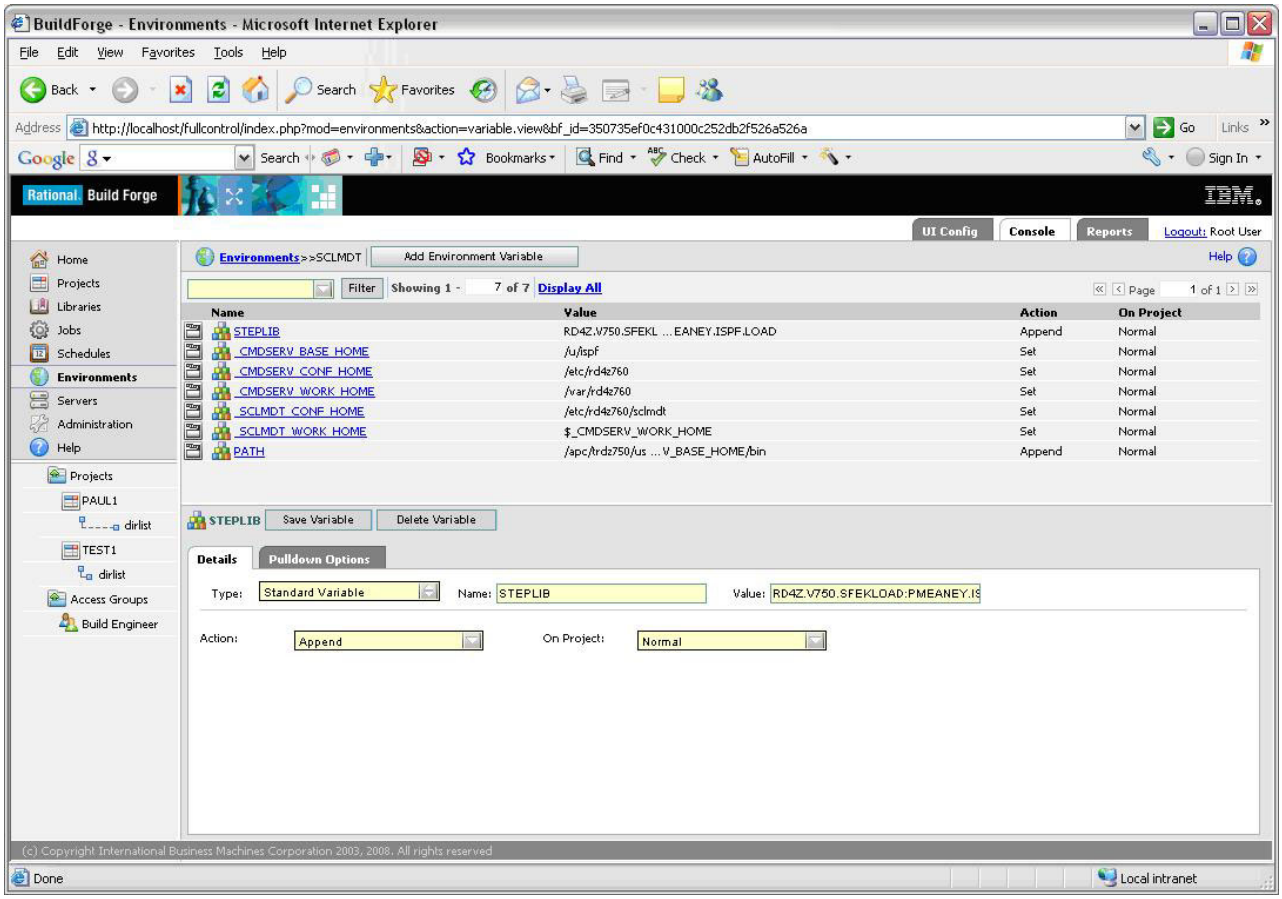


Figure 30. Environment variable definitions

3. SCLM Project setup in Build Forge

   Now we need to configure SCLM projects for our various SCLM service requests. The samples below will be for the SCLM Build and promote services which would generally be the main services that suit Build Forge job scheduling. The project script samples are based upon the SCLM XML API format documented in the *SCLM Developer Toolkit Administrators Guide*.

   a. First set up an SCLM project by clicking **Projects -> Add project**.

      NAME: Enter the name of this project.
      (For example: SCLM build sample1)
      SELECTOR: Enter the selector name configured in the Server selector table (as in previous section 1c on page 117).
      ENVIRONMENT: Enter the Environment name configured for this

SCLM project containing the SCLM environment variables (as in previous section 2a on page 118).
For example: SCLMDT

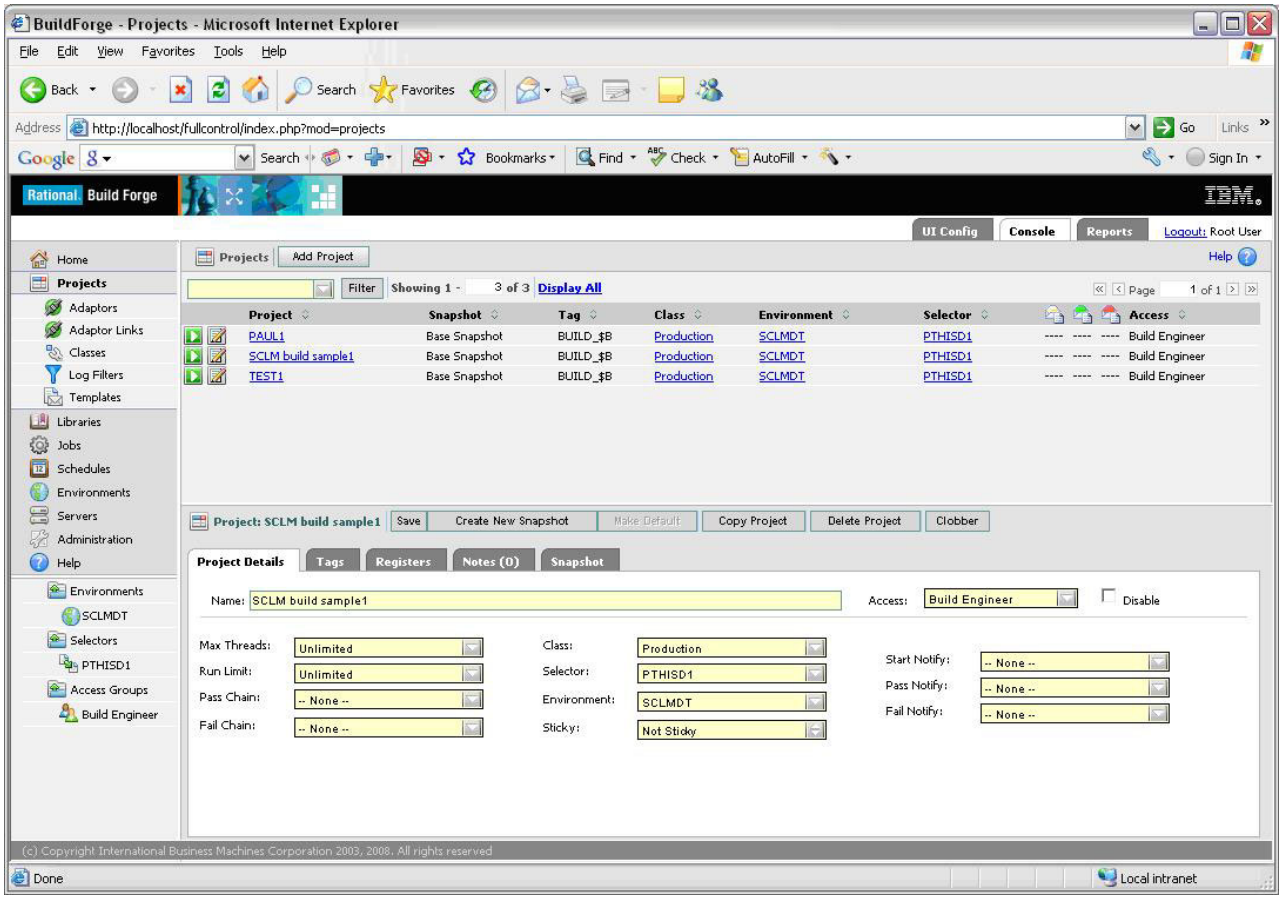


Figure 31. Project sample setup for SCLM build (SCLM build sample1)

b. Now add a Build step to this SCLM project.

   The following sample is a configured Build sample distributed in the Host SAMPLIB dataset (BWBBFBLD).

```
echo '<?xml version-"1.0"?>' > Build_input.txt
echo '<SCLMDT-INPUT' >> Build_input.txt
echo 'xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"' >> Build_input.txt
echo 'xsi:noNamespaceSchemaLocation="sclmdt.xsd">' >> Build_input.txt
echo '<SERVICE_REQUEST>' >> Build_input.txt
echo '<service>SCLM</service>' >> Build_input.txt
echo '<session>NONE</session>' >> Build_input.txt
echo '<ispprof>HLQ.SCLMDT.ISPPROF</ispprof>'  >> Build_input.txt
echo '<sclmfunc>BUILD</sclmfunc>' >> Build_input.txt
echo '<project>PROJECT</project>' >> Build_input.txt
echo '<projdef>PROJDEF</projdef>' >> Build_input.txt
echo '<member>MEMBER</member>' >> Build_input.txt
echo '<group>GROUP</group>' >> Build_input.txt
echo '<type>TYPE</type>' >> Build_input.txt
echo '<repdgrp>DEVGRP</repdgrp>' >> Build_input.txt
echo '<groupbld>BLDGRP</groupbld>' >> Build_input.txt
echo '<bldmode>C</bldmode>' >> Build_input.txt
echo '<bldlist>Y</bldlist>' >> Build_input.txt
echo '<bldlstds>NONE</bldlstds>' >> Build_input.txt
echo '<bldrept>Y</bldrept>' >> Build_input.txt
echo '<bldscope>N</bldscope>' >> Build_input.txt
echo '<bldmsg>Y</bldmsg>' >> Build_input.txt
echo '<bldmsgds>NONE</bldmsgds>' >> Build_input.txt
echo '<bldextds>NONE</bldextds>' >> Build_input.txt
echo '</SERVICE-REQUEST>' >> Build_input.txt
echo '<SCLMDT-INPUT>'  >> Build_input.txt

cat Build_input.txt | BWBXML >Build_output.txt
cat Build_output.txt
```

*Figure 32. ** SCLM Build sample1 **

Configure the above sample with your SCLM project, group, type, member and build options as detailed in the SCLM DT API section APPENDIX C for Build functionality.

Add this configured sample to the first step in Project SCLM Build sample1.

Projects -> SCLM Build sample1 -> add step
NAME: The name you choose to call this step
COMMAND: Include the above configured sample (BWBBFBLD) in this area

All the other fields may remain with the default. (This step will inherit the main project environment settings.)

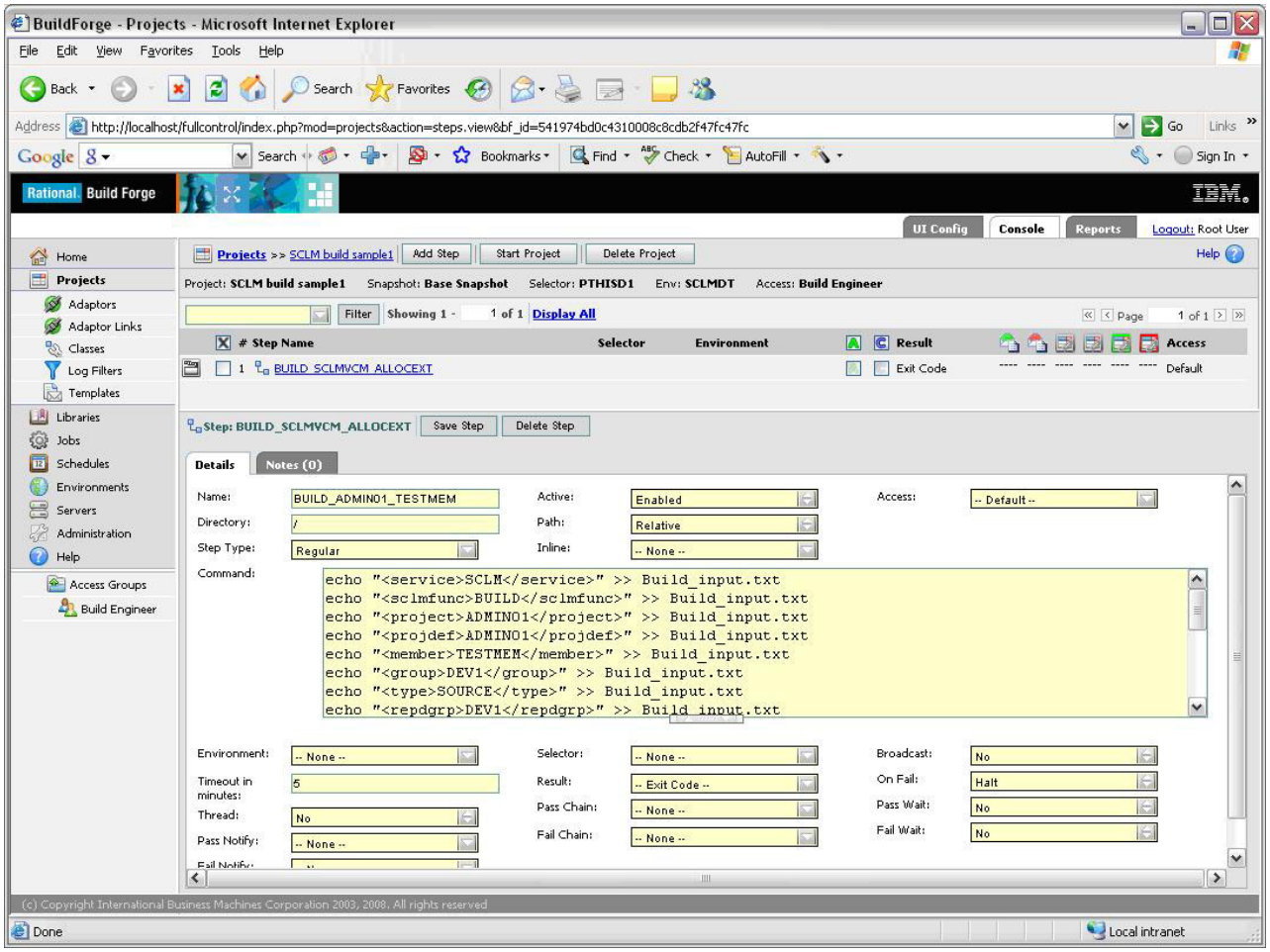


Figure 33. Project sample step within SCLM build sample1

c. Execute the SCLM Build project SCLM Build sample1.

   Click **Projects -> SCLM Build sample1 -> start project**. The output from the build is returned in the console **JOBS -> Build tag**.

   The request and response output will also be stored in the z/OS Unix Systems Services directory determined by the PATH specified in the Server configuration. In the previous example for Server PTHISD1 with PATH=/u/IBMUSER/BUILDFORGE, the request and response files would be stored under directory:

```
/u/IBMUSER/BUILDFORGE/SCLM_Build1_sample/BUILD_1
: >ls
Build_input.txt   Build_output.txt
```

   The above request and response files may be renamed within the customized Build step.

SCLM promote sample

The steps in the previous section can be replicated for creating an SCLM promote project in Build Forge. Replace the build script with a promote script sample in the Project "COMMAND" area. The following sample is a configured Promote sample distributed in the Host SAMPLIB dataset (BWBBFPRM).

```
echo '<?xml version="1.0"?>' > Promote_input.txt
echo '<SCLMDT-INPUT' >> Promote_input.txt
echo 'xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"' >> Promote_input.txt
echo 'xsi:noNamespaceSchemaLocation="sclmdt.xsd">' >> Promote_input.txt
echo '<SERVICE-REQUEST>' >> Promote_input.txt
echo '<service>SCLM</service>'     >>Promote_input.txt
echo '<session>NONE</session>'     >>Promote_input.txt
echo '<ispprof>HLQ.SCLMDT.ISPPROF</ispprof>'     >>Promote_input.txt
echo '<sclmfunc>PROMOTE</sclmfunc>'     >>Promote_input.txt
echo '<project>PROJECT</project>'     >>Promote_input.txt
echo '<projdef>PROJDEF</projdef>'     >>Promote_input.txt
echo '<member>MEMBER</member>'     >>Promote_input.txt
echo '<group>GROUP</group>'     >>Promote_input.txt
echo '<type>TYPE</type>'     >>Promote_input.txt
echo '<repdgrp>DEVGRP</repdgrp>'     >>Promote_input.txt
echo '<groupprm>PRMGRP</groupprm>'     >>Promote_input.txt
echo '<prmmode>C</prmmode>'     >>Promote_input.txt
echo '<prmrept>Y</prmrept>'     >>Promote_input.txt
echo '<prmscope>N</prmscope>'     >>Promote_input.txt
echo '<prmmsg>Y</prmmsg>'       >>Promote_input.txt
echo '<prmmsgds>NONE</prmmsgds>'  >>Promote_input.txt
echo '<prmextds>NONE</prmextds>'  >>Promote_input.txt
echo '</SERVICE-REQUEST>'     >>Promote_input.txt
echo '</SCLMDT-INPUT>'     >>Promote_input.txt


cat Promote_input.txt | BWBXML >Promote_output.txt
cat Promote_output.txt
```

*Figure 34. ** SCLM Promote sample1 **

Configure the previous sample with your SCLM project, group, type, member, and promote options as detailed in the Appendix C, “SCLM Developer Toolkit API,” on page 61 for Promote functionality.

Documentation notices for IBM Rational Developer for System z

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

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

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
3-2-12, Roppongi, Minato-ku, Tokyo 106-8711 Japan

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

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

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

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

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:

Intellectual Property Dept. for Rational Software
IBM Corporation

3039 Cornwallis Road, PO Box 12195
Research Triangle Park, NC 27709
U.S.A.

I

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

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

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

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

Copyright license

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

Trademark acknowledgments

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at `www.ibm.com/legal/copytrade.shtml`.

Rational are trademarks of International Business Machines Corporation and Rational Software Corporation, in the United States, other countries, or both.

Intel and Pentium are trademarks of Intel Corporation in the United States, or other countries, or both.

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

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

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

Index

Special characters

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

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

IBM Rational Developer for System z
SCLM Developer Toolkit Administrator's Guide
Version 7.6

Publication No. SC23-9801-01

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.
- Send a fax to the following number: 1-800-227-5088 (US and Canada)
- Send your comments via e-mail to: kfrye@us.ibm.com

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

Name	Address
Company or Organization	
Phone No.	E-mail address

Readers' Comments — We'd Like to Hear from You
SC23-9801-01

IBM

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 G7IA / Bldg. 503
P.O. Box 12195
Research Triangle Park, NC
27709-2195

Fold and Tape | **Please do not staple** | Fold and Tape

SC23-9801-01

Cut or Fold
Along Line

Program Number: 5724-T07

Printed in USA

SC23-9801-01