

IBM Rational Developer for System z
8.5.0

SCLMDT Administrator



IBM Rational Developer for System z
8.5.0

SCLMDT Administrator



Hinweis

Vor Verwendung dieser Informationen sollten Sie die allgemeinen Hinweise unter „Dokumentationshinweise für IBM Rational Developer for System z“ auf Seite 143 lesen.

Dritte Ausgabe (Juni 2012)

Diese Ausgabe bezieht sich auf IBM Rational Developer for System z Version 8.5.0 (Programmnummer 5724-T07) und - sofern in neuen Ausgaben nicht anders angegeben - auf alle nachfolgenden Releases und Modifikationen.

Diese Veröffentlichung ist eine Übersetzung der *IBM Rational Developer for System z 8.5.0 SCLMDT Administrator's Guide*,

IBM Form: SC23-9801-03

herausgegeben von International Business Machines Corporation, USA

Copyright International Business Machines Corporation 2007, 2012

Copyright IBM Deutschland GmbH 2007, 2012

Informationen, die nur für bestimmte Länder Gültigkeit haben und für Deutschland, Österreich und die Schweiz nicht zutreffen, wurden in dieser Veröffentlichung im Originaltext übernommen.

Möglicherweise sind nicht alle in dieser Übersetzung aufgeführten Produkte in Deutschland angekündigt und verfügbar; vor Entscheidungen empfiehlt sich der Kontakt mit der zuständigen IBM Geschäftsstelle.

Änderung des Textes bleibt vorbehalten.

Herausgegeben von:

SW NLS Center

Kst. 2877

Juni 2012

© Copyright IBM Corporation 2010, 2012.

Inhalt

Abbildungen	v
------------------------------	----------

Tabellen	vii
---------------------------	------------

Zu diesem Handbuch	ix
Zielgruppe	ix

Kapitel 1. Produktinstallation	1
---	----------

Kapitel 2. SCLM für SCLM Developer Toolkit anpassen	3
--	----------

SCLM für SCLM Developer Toolkit anpassen	3
Zusammenfassung zu JAVA/J2EE-Builds	3
Generierte JAVA/J2EE-Buildobjekte	4
Sprachumsetzer für JAVA/J2EE-Unterstützung	5
SCLM-Sprachdefinitionen	5
SCLM-Dateigruppen für JAVA/J2EE	7
SCLM-Typen	7
Empfohlene Datensatzattribute für wichtige Typen	7
SCLM-Memberformate	9
\$GLOBAL-Format	9
J2EE ARCHDEF-Format	9
J2EE ARCHDEF-Beispiel	10
Format des J2EE-Ant-Erstellungsscripts	13
CLASSPATH-Abhängigkeiten	14
J2EE-Beispielscripts	15
JAVA/J2EE-Ant-XML-Erstellungsentwürfe	16
Zuordnen, J2EE-Projekte zu SCLM	18
Empfehlungen für die Zuordnung von J2EE-Projekten in SCLM	19
SCLM Developer Toolkit - Implementierung	21
WebSphere Application Server (WAS) - Implementierung	22
Implementierung von SCLM in UNIX System Services	22
Sichere Implementierung	22
Public-Key-Authentifizierung	23
Weitere Implementierungsoptionen	23
ASCII- oder EBCDIC-Speicheroptionen	23
ASCII-/EBCDIC-Sprachumsetzer	24
\$GLOBAL	25
TRANSLATE.conf	26
SITE- und projektspezifische Optionen	28
Beispiel für die Verwendung von Kombinationen der TRANSLATE.conf-Überschreibungen	33
Beispiel für die Verwendung von Kombinationen der BIDIPROP-Überschreibungen	34

Kapitel 3. SQLJ-Unterstützung	37
--	-----------

Was ist SQL?	37
Was ist DB2?	37
Was ist JDBC?	37
Was ist SQLJ?	38

Vergleich von JDBC und SQLJ	38
Was ist ein serialisiertes Profil?	39
Was ist ein Datenbankanforderungsmodul (DBRM)?	40
SQLJ-Programmerstellung	40
Umsetzung	41
Anpassung	42
Bindung	42
SCLM DT-Typen und -Umsetzer	42
Anpassung des Erstellungsprozesses	43
Anpassung des Erstellungsscripts	43
sqlj*-Eigenschaften	44
db2sqljcustomize*-Eigenschaften	44
Angepasstes Benutzerscript	45
Bindung [DBRM]	46
Bindung [Serialisiertes Profil]	48

Kapitel 4. SCLM-Sicherheit	51
---	-----------

Sicherheitsflag	51
Einrichtung in Ihrem Sicherheitsprodukt	51
Sicherheitsprofile	52
Ersatzbenutzer-ID	52
Beispiel: Erstellung	52
Beispiel: Implementierung	53

Kapitel 5. CRON-aktivierte Erstellungen und Umstufungen	55
--	-----------

STEPLIB- und PATH-Anforderungen	55
CRON-Erstellungsjob, Ausführung	56
CRON-Erstellungsjob, Beispiel	56

Anhang A. SCLM-Übersicht	61
---	-----------

SCLM-Konzepte	61
Dateibenennung	61
Typ	61
Sprache	62
SCLM-Eigenschaften	62
SCLM-Projektstruktur	62
ARCHDEF	62
JAVA/J2EE-Konzepte	63

Anhang B. Umsetztabelle für Lang-/Kurznamen	65
--	-----------

Technische Zusammenfassung des SCLM-Umsetzungsprogramms	65
Verarbeitung einzelner Lang-/Kurznamensätze	66
FINDLONG-Verarbeitung	66
FINDSHORT-Verarbeitung	67
TRANSLATE-Verarbeitung	67
Verarbeitung mehrerer Lang-/Kurznamensätze	68
IMPORT-Verarbeitung	68
MIGRATE-Verarbeitung	68

Anhang C. SCLM Developer Toolkit-API	71
---	-----------

Aufrufformat	71
------------------------	----

XML-Schema für SCLMDT-Befehle	72
Funktionen und Parameter anfordern.	77
Funktionsformat.	77
Funktionsliste	78
AUTHUPD – SCLM-Berechtigungscode ändern	78
BROWSE – SCLM-Member durchsuchen	79
BUILD – SCLM-Member erstellen	79
DELETE – SCLM-Member löschen.	81
DEPLOY – J2EE-EAR-Datei implementieren	81
EDIT – SCLM-Member bearbeiten	82
INFO – Statusinformationen zum SCLM-Member	83
J2EEIMP – Projekt aus SCLM importieren	83
J2EEMIG – Projekt in SCLM migrieren	85
J2EEMIGB – Projekt als Batch in SCLM migrieren	86
JARCOPY – JAR-Datei kopieren	87
JOBSTAT – Status des Batch-Jobs abrufen	87
LRECL – LRECL aus SCLM-Dateigruppe abrufen	88
MIGDSN – Nicht-SCLM-Dateigruppen und	
-Member auflisten	88
MIGPDS – Nicht-SCLM-Dateigruppen und	
-Member in SCLM migrieren	88
PROJGRPS – SCLM-Gruppen für ein Projekt ab-	
rufen	89
PROJINFO – SCLM-Projektinformationen abrufen	89
PROMOTE – SCLM-Member umstufen	89
REPORT – Projektbericht erstellen	91
REPUTIL – SCLM-DBUTIL-Bericht	92
SAVE – SCLM-Member speichern	92
UNLOCK – SCLM-Member freigeben	93
UPDATE – SCLM-Memberinformationen aktuali-	
sieren	93
VERBROW – SCLM-Versionen anzeigen.	94
VERDEL – SCLM-Versionen löschen	94
VERHIST – SCLM-Versionsprotokoll	95
VERLIST – SCLM-Versionen auflisten	95
VERREC – SCLM-Versionen wiederherstellen	96
Beispiel.	96
sclmdt_request.xml	96
xmlbld.java	97
sclmdt_response.xml	99

Anhang D. Rational Application Developer for WebSphere Software - Erstellungsdiensprogramm. 107

Übersicht über das Erstellungsdiensprogramm von Rational Application Developer for WebSphere Software	107
Speichern von Java/J2EE-Objekten in SCLM	107
Das Erstellungsdiensprogramm im Vergleich mit dem nativen ANT-Erstellungsprozess	108
Installationshinweis zum Erstellungsdiensprogramm von Rational Application Developer for WebSphere Software	109
Kombinierter Einsatz von SCLM und Erstellungsdiensprogramm	109

Rational Application Developer for WebSphere Software - Implementierung und Verwendung des Erstellungsdiensprogramms	109
Sprachumsetzer des SCLM-Erstellungsdiensprogramms	110
J2EEAST: J2EE-Überprüfungs-/	
Erstellungsumsetzer	111
J2EEOBJ: J2EE-ARCHDEF-Umsetzer	111
JAVA: Java-Sprachumsetzer (EBCDIC)	112
JAVABIN: Java-Sprachumsetzer (ASCII).	112
J2EEPART: J2EE-Text-Sprachumsetzer	112
J2EEBIN: Sprachumsetzer für J2EE (binär)	113
Erstellungsscripts und Formate des Erstellungsdiensprogramms	113
Format des Erstellungsscripts	113
Nicht modifizierte AST-Routinen	114
projectImport	114
projectBuild	114
EJBexport.	115
WARExport	116
EARExport	116
AppClientExport	116
AST-Ausführungsscript (Beispiel-BW-	
BASTR)	117
Java/JAR-Erstellungsscript (Beispiel)	118
WAR-Erstellungsscript (Beispiel)	119
EJB-Erstellungsscript (Beispiel)	119
EAR-Erstellungsscript (Beispiel)	120
J2EE ARCHDEF-Format	121
SQLJ-Buildunterstützung	122
SQLJ-Anpassung (für den SCLM-Administra-	
tor).	122
Systemvoraussetzungen	122
Sprachumsetzer	122
SQLJ-Benutzeranpassung	123
SQLJ-Build-Eigenschaftenscript	123
SCLM ARCHDEF (Beispiel ASTSQLJ)	124
SCLM-AST-Erstellungsscript (Beispiel	
ASTSQLJ)	124
Java-Quelle in Archivdateien	126
EINSATZSZENARIO: 'PlantsByWebSphere'	126

Anhang E. BUILD FORGE und SCLM 133

Übersicht.	133
Voraussetzungen	133
Build Forge-Agent unter z/OS aufrufen	133
Build Forge - Konsolenserverkonfiguration	134
Beispiel für SCLM-Umstufung.	140

Dokumentationshinweise für IBM Rational Developer for System z 143

Copyright-Lizenz	145
Markenhinweise	146

Abbildungen

1. Beispiel für Jar-Anwendung (JAR) ARCHDEF	4	22. XML-Schema für SCLMDT-Befehle.	73
2. JAR-Beispiel für J2EE-Erstellungsscript	4	23. XML-Schema für SCLMDT-Befehle (Fortsetzung)	74
3. \$GLOBAL - SCLMDT-Umgebungsvariablen	9	24. XML-Schema für SCLMDT-Befehle (Fortsetzung)	75
4. Sample Jar application (JAR) ARCHDEF	10	25. XML-Schema für SCLMDT-Befehle (Fortsetzung)	76
5. Beispiel für Webanwendung (WAR) ARCHDEF	11	26. XML-Schema für SCLMDT-Befehle (Fortsetzung)	77
6. Beispiel für EJB-Anwendung (EJB) ARCHDEF	12	27. Grundstruktur der XML-Eingabedatei.	77
7. Beispiel für EAR-Anwendung (EAR) ARCH-DEF	12	28. sclmdt_request.xml – Beispiel-XML-Befehl-Eingabedatei	97
8. J2EE-Ant-Erstellungsscript	13	29. xmlbld.java – Beispiel-Java-Programm.	98
9. JAR-Beispiel für J2EE-Erstellungsscript	15	30. xmlbld.java – Beispiel-Java-Programm (Fortsetzung)	99
10. WAR-Beispiel für J2EE-Erstellungsscript	15	31. sclmdt_response.xml – Beispiel-XML-Ausgabedatei	99
11. EJB-Beispiel für J2EE-Erstellungsscript	16	32. Serverautorisierung	134
12. EAR-Beispiel für J2EE-Erstellungsscript	16	33. Serverdefinition.	135
13. SCLM-Buildhierarchie	20	34. Definition des Serverselektors	136
14. TRANSLATE.conf - SCLMDT ASCII/EBCDIC Umsetzungskonfigurationsdatei	27	35. Umgebungsvariablendefinitionen	137
15. Beispiel für SITE-spezifische SCLM-Projekteinstellung	30	36. Projektmustereinrichtung für SCLM-Build (SCLM build sample1)	138
16. Beispiel für projektspezifische SCLM-Projekteinstellung	31	37. ** SCLM-Build sample1 **	139
17. SQLJ-Programmerstellung.	41	38. Projektmusterschritt in SCLM build sample1	140
18. BWBCRON1 - CRON Build exec	57	39. ** SCLM-Umstufung sample1 **	141
19. BWBCRONB - Erstellungsparameterdatei	58		
20. BWBCRONP - Promote parameter file	59		
21. Beispiel REXX für den Aufruf des Umsetzungsmoduls	69		

Tabellen

1. Prüfliste für den SCLM-Administrator	1	11. SCLMDT Developer Toolkit-Sicherheitsprofile	52
2. Umsetzer für SCLM Developer Toolkit	5	12. Parameter für die Umsetzung von Lang-/	
3. Kundendefinierte Variablen	13	Kurznamen.	66
4. SCLM-Sprachumsetzer und ASCII/EBCDIC	24	13. Parameter für projectImport.	114
5. \$GLOBAL-Variablen	25	14. Parameter für projectBuild	114
6. SITE-/Projektoptionen	31	15. Parameter von EJBexport.	115
7. Vergleich von JDBC und SQLJ	38	16. Parameter für WARExport	116
8. SCLM-Umsetzertypen für SQLJ	43	17. Parameter für EARExport	116
9. sqlj.*-Eigenschaften	44	18. Parameter für AppClientExport	116
10. db2sqljcustomize.*-Eigenschaften	44		

Zu diesem Handbuch

In diesem Dokument wird die Konfiguration von IBM® Software Configuration and Library Manager (SCLM) beschrieben, die für die SCLM Developer Toolkit-Funktion von IBM Rational Developer for System z benötigt wird.

Im weiteren Verlauf dieses Handbuchs werden die folgenden Namen verwendet:

- *IBM Software Configuration and Library Manager*, kurz *SCLM*.
- *IBM Rational Developer for System z*, kurz *Developer for System z*.
- Die *SCLM Developer Toolkit*-Funktion von *IBM Rational Developer for System z* wird als *SCLM Developer Toolkit* oder kurz *Developer Toolkit* bzw. *SCLMDT* bezeichnet.

Die Informationen in diesem Dokument gelten für alle Pakete von Rational Developer for System z Version 8.5 einschließlich IBM Rational Developer for zEnterprise.

Zielgruppe

Dieses Dokument enthält Informationen für Administratoren von SCLM-Projekten, die mit dem SCLM Developer Toolkit verwendet werden. Dies umfasst Projekte, für die Java- und z/OS UNIX System Services-Komponentensprachen verwendet werden, sowie konventionelle SCLM-Projekte.

Die Administratoren dieser Projekte sollten mit z/OS UNIX System Services, REXX-Scripts, dem Java-Compiler sowie mit SCLM-Projekt- und Sprachdefinitionen vertraut sein.

Kapitel 1. Produktinstallation

Diese Veröffentlichung bezieht sich nicht auf die Implementierung und das Laden des Produktes „SCLM“, das im Lieferumfang des z/OS-Betriebssystems enthalten ist. Ebensovienig wird die Installation und Konfiguration des SCLM Developer Toolkits berücksichtigt, das eine Funktion von Rational Developer for System z ist.

Weitere Informationen zu diesen Tasks finden Sie im Handbuch *ISPF Software Configuration and Library Manager Project Manager's and Developer's Guide* (IBM Form SC34-4817-10) und *Rational Developer for System z Hostkonfiguration* (IBM Form SC12-4062-04).

Um die Aufgaben zur Anpassung und Projektdefinition ausführen zu können, muss der SCLM-Administrator bestimmte für Developer for System z anpassbare Werte kennen, wie in Tabelle 1 beschrieben. Wenden Sie sich an den z/OS-Systemprogrammierer, der für die Installation und Anpassung von Developer for System z verantwortlich ist, wenn Sie weitere Informationen benötigen.

Tabelle 1. Prüfliste für den SCLM-Administrator

Beschreibung	Standardwert	Entsprechende Quelle	Wert
Beispielbibliothek für Developer for System z	FEK.SFEKSAMP	SMP/E-Installation	
Beispielverzeichnis für Developer for System z	/usr/lpp/rdz/samples	SMP/E-Installation	
Java-Bin-Verzeichnis	/usr/lpp/java/J5.0/bin	rsed.envvars - \$JAVA_HOME/bin	
Ant-Bin-Verzeichnis	/usr/lpp/Ant/apache-ant-1.7.1/bin	rsed.envvars - \$ANT_HOME/bin	
WORKAREA-Ausgangsverzeichnis	/var/rdz	rsed.envvars - \$_CMDSEV_CONF_HOME	
Ausgangsverzeichnis für SCLMDT-Projektkonfigurationen	/var/rdz/sc1mdt	rsed.envvars	
VSAM für Umsetzung langer/kurzer Namen	FEK.#CUST.LSTRANS.FILE	rsed.envvars - \$_SCLMDT_TRANTABLE	

Kapitel 2. SCLM für SCLM Developer Toolkit anpassen

In diesem Kapitel wird erläutert, wie der SCLM-Administrator SCLM für die Verwendung mit dem SCLM Developer Toolkit anpassen kann.

SCLM für SCLM Developer Toolkit anpassen

In diesem Kapitel wird erläutert, wie der SCLM-Administrator SCLM für die Verwendung mit dem SCLM Developer Toolkit anpassen kann.

Zusammenfassung zu JAVA/J2EE-Builds

Im Folgenden finden Sie eine Zusammenfassung des Prozesses, der für Java- und J2EE-Builds mit den mitgelieferten Umsetzern ausgeführt wird.

Anmerkung: Sie können JAVA/J2EE-Member oder Architekturdefinitionen (ARCHDEF) direkt in TSO/ISPF auf dem Host oder über den Developer Toolkit-Client erstellen.

Die ARCHDEF enthält die Member, aus denen das JAVA/J2EE-Projekt besteht. Diese verdeutlichen als Kurznamendarstellung, wie das Projekt in einem Eclipse-Arbeitsbereich gespeichert ist.

Die ARCHDEF selbst wird erstellt. Dabei wird ein vorerstellter Sprachüberprüfungsumsetzer (J2EEANT) aufgerufen. Der Umsetzer liest das J2EE-Erstellungsscript, das in der ARCHDEF durch das Schlüsselwort SINC referenziert wird, und überschreibt die angegebenen Eigenschaften in der Entwurfs-Ant-XML, die durch die Eigenschaften SCLM-ANTXML(A) referenziert wird. Das Erstellungsscript wird, wenn es durch SCLM Developer Toolkit erstellt wird, in SCLM mit einer Sprache des Typs J2EEANT gespeichert (1).

Eine ARCHDEF generiert Java-Klassen für Java-Quellcode, der mit dem Schlüsselwort INCLD in der ARCHDEF angegeben wird (2). Jede ARCHDEF kann auch eine J2EE-Archivierungsdatei generieren, wie z. B. eine JAR-, WAR- oder EAR-Datei. Das erstellte J2EE-Objekt ist abhängig von dem entsprechenden referenzierten Erstellungsscript und der Verwendung des ARCHDEF-Schlüsselworts OUT1 (3), (B).

Wenn die ARCHDEF erstellt wird, wird der vorerstellte Sprachüberprüfungsumsetzer ausgeführt, der mit dem Erstellungsscript verknüpft ist (im SCLM-Typ J2EEBLD). Dieser ermittelt, welche Teile der ARCHDEF neu erstellt werden müssen (einschließlich verschachtelter ARCHDEFs (4), die durch die Verwendung des Schlüsselworts INCL in der ARCHDEF identifiziert werden). Diese Teile werden dann in den Dateisystem-Arbeitsbereich von z/OS UNIX System Services kopiert. Ant kompiliert und generiert die erforderlichen JAVA/J2EE-Objekte, die durch das Erstellungsscript und die ARCHDEF angegeben werden. Externe JAR- oder Klassenreferenzen, die in Ihrem IDE-Projekt aufgelöst werden müssen, werden mit dem in der Eigenschaft CLASSPATH_JARS definierten Pfad aufgelöst (C).

SCLM verarbeitet anschließend jede einzelne ARCHDEF-Komponente, indem alle Sprachumsetzer ausgeführt werden, die mit der Komponente verknüpft sind. Der JAVA-Sprachumsetzer, der mit dem Java-Quellcode verknüpft ist, kopiert die erstellten Klassendateien zurück in SCLM.

Schließlich ermittelt der ARCHDEF-Umsetzer, welche J2EE-Objekte erstellt wurden (JAR, WAR, EAR), und kopiert diese Teile zurück in SCLM.

Es ist wichtig, eine separate ARCHDEF für jede Anwendungskomponente zu erstellen, die möglicherweise eine Unternehmensanwendung (EAR) darstellt. Eine EAR-Datei, die eine WAR-Datei enthält, die wiederum eine EJB-JAR-Datei enthält, sollte daher eine ARCHDEF für die JAR-Datei haben, eine ARCHDEF für die WAR-Datei mit einem INCL der EJB-JAR-ARCHDEF. Die EAR-ARCHDEF sollte in diesem Fall ein INCL der WAR-ARCHDEF umfassen.

Das folgende Beispiel zeigt den 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
```

Abbildung 1. Beispiel für Jar-Anwendung (JAR) ARCHDEF

Das folgende Beispiel zeigt das zugehörige JAR-Script:

```
<ANTXML>
<project name="JAVA Project" default="jar" basedir=".">
<property name="env" environment="env" value="env"/>
<property name="SCLM_ARCHDEF" value="V2JAR1"/>
<property name="SCLM_ANTXML" value="BWBJAVA" /> (A)
<property name="SCLM_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>
```

Abbildung 2. JAR-Beispiel für J2EE-Erstellungsscript

Generierte JAVA/J2EE-Buildobjekte

Folgende Objekte werden generiert:

- Kompilierung des gesamten Java-Quellcodes in Ausgabeklassen, die im SCLM-Typ JAVACLAS gespeichert werden.
- Klassen, die in SCLM gespeichert werden, und Lang-/Kurznamen, die in Umsetztabelle gespeichert werden.
- Optional erstellte JAR-Datei (enthält Klassen und eventuell weitere Java-Projekt-komponenten, wie XML, HTML usw. in einer paketierte Struktur).
- JAR-Objekte, die in SCLM gespeichert werden, und Lang-/Kurznamen, die in Umsetztabelle gespeichert werden.

- Die JAR-Struktur wird durch die verwendete ARCHDEF bestimmt. Die ausgeschriebenen Namen, die den Members in der ARCHDEF zugeordnet sind, legen das JAR-Paketierungsformat fest.
- Optionales EJB JAR (enthält Klassen und eventuell weitere Java-Projektkomponenten, wie XML, HTML, JSP usw., in paketierter Struktur).
- Optionale Web-WAR-Datei basierend auf J2EE-Datei web.xml im J2EE-Projekt und gespeichert in SCLM, wie oben.
- Optionale EAR-Datei für Implementierung basierend auf application.xml im J2EE-Projekt und gespeichert in SCLM, wie oben.
- Alle Listenausgaben werden im SCLM-Typ J2EELIST gespeichert.

Sprachumsetzer für JAVA/J2EE-Unterstützung

Für das SCLM Developer Toolkit sind neue Sprachumsetzer erforderlich, die in SCLM für JAVA/J2EE-Unterstützung definiert sind. Diese Sprachumsetzer werden in den FEK.SFEKSAMV-Members bereitgestellt, wie im Folgenden aufgeführt:

Tabelle 2. Umsetzer für SCLM Developer Toolkit

Umsetzer	Beschreibung
BWBTRANJ	Beispiel-Standardmember-Umsetzer. Keine Syntaxanalyse. Ähneln SCLM FLM@TEXT. Dieser Umsetzer kann angepasst werden, um die Sprachdefinitionen J2EEPART, J2EEBIN, BINARY und TEXT zu erstellen.
BWBTRANS	Beispiel-SQLJ-Sprachumsetzer. LANG=SQLJ
BWBTRAN1	Beispiel-Java-Sprachumsetzer. LANG=JAVA
BWBTRAN2	Beispiel-JAVA/J2EE-Sprachumsetzer einschließlich Ant (für mehrfache Java-Kompilierungen und JAR-, WAR- und EAR-Builds).
BWBTRAN3	Beispiel-J2EE-Sprachumsetzer für Unterstützung von SCLM ARCHDEF J2EE. LANG=J2EEOBJ

Der SCLM-Administrator muss dieses Beispiel kopieren, eventuell umbenennen und anschließend in der Bibliothek PROJDEFS.LOAD für jedes SCLM-Projekt erstellen, bei dem Java-Unterstützung benötigt wird. Diese Umsetzer müssen in der Projektdefinition hinzugefügt oder kompiliert werden.

Ein Beispiel für eine Projektdefinition für JAVA/J2EE-Projekte und Hostkomponenten ist im Beispiel BWBSCLM angegeben.

SCLM-Sprachdefinitionen

Die Beispielumsetzer definieren die folgenden Sprachen:

J2EEANT

Dies ist der Haupt-Buildumsetzer für JAVA/J2EE-Builds und dieser Überprüfungsumsetzer wird aufgerufen, wenn eine J2EE-ARCHDEF erstellt wird. Der Umsetzer wird aufgerufen, da das JAVA/J2EE-Erstellungsscript, das im SCLM-Typ J2EEBLD gespeichert ist, in SCLM mit einer Sprache des Typs J2EEANT gespeichert wird. Es wird anschließend durch das Schlüsselwort SINC in der ARCHDEF referenziert.

Dieser Überprüfungsumsetzer ermittelt, welche Teile erstellt werden müssen (einschließlich verschachtelter ARCHDEFs), und kopiert diese Teile abhängig von den Erstellungsmodi in das WORKAREA-Verzeichnis von z/OS UNIX System Services. Eine Entwurfs-Ant-XML wird dynamisch angepasst anhand der Erstellungsscripts sowie der Teile, die im Arbeitsbereich erstellt wurden, der Ant verwendet. Die Klassendateien werden an die JAVA/JAVABIN-Sprachumsetzer übermittelt, um diese in SCLM zu

speichern. Die erstellten J2EE-Objekte, wie JAR-, WAR- oder EAR-Dateien werden an den ARCHDEF-Sprachumsetzer (J2EEOBJ) übermittelt, um in SCLM gespeichert zu werden.

J2EEBIN

Sprachtyp, der die binär oder in ASCII gespeicherte JAVA/J2EE-Komponente angibt und durch das Beispiel BWBTRANJ definiert wird. Es wird keine spezielle Syntaxanalyse bei der Erstellung dieser Sprachdefinition ausgeführt. JAVA/J2EE-Binärdateien und Textdateien, die als ASCII gespeichert werden sollen, können generisch unter dieser Sprachdefinition eingeschoben werden, wenn keine gesonderte Syntaxanalyse benötigt wird.

J2EEOBJ

Dies ist der endgültige Buildumsetzer, der während des ARCHDEF-Erstellungsprozesses aufgerufen wird. Dieser Umsetzer ermittelt, welche J2EE-Objekte (JAR, WAR, EAR) zuvor im Umsetzer J2EEANT erstellt wurden, und kopiert diese Objekte in SCLM mit dem angegebenen erstellten Kurznamen. Dieser Umsetzer wird mit dem Schlüsselwort LKED in der ARCHDEF selbst referenziert.

J2EEPART

Sprachtyp, der eine JAVA/J2EE-Komponente angibt und durch das Beispiel BWBTRANJ definiert wird. Es wird keine spezielle Syntaxanalyse bei der Erstellung dieser Sprachdefinition ausgeführt. Nicht-Java-Quellen oder J2EE-Komponenten, für die ASCII/EBCDIC-Sprachkonvertierung erforderlich ist, können unter dieser Sprachdefinition generisch eingeschoben werden, wenn keine bestimmte Build-Syntaxanalyse benötigt wird (z. B. html, XML, .classpath, .project oder Definitionstabellen). Optional kann die Sprachdefinition TEXT verwendet werden.

JAVA Sprachtyp für Java-Quellcode, der durch das Beispiel BWBTRAN1 definiert wird. Der Java-Umsetzer ermittelt, welcher Buildtyp für den Java-Quellcode ausgegeben wurde.

Anmerkung: Diese Sprachdefinition muss Java-Programmen zugewiesen werden, wenn Sie den Java-Quellcode in EBCDIC auf dem Host speichern möchten (d. h. der Quellcode kann über ISPF direkt auf dem Host angezeigt und bearbeitet werden). Der Vorteil des Definierens von Programmen mit dieser Sprachdefinition liegt darin, dass der Quellcode direkt auf dem z/OS-Host bearbeitet und angezeigt werden kann. Nachteilig ist, dass Codepagekonvertierungen vorgenommen werden müssen, wenn Projekte vom Client auf den Host migriert oder importiert werden.

- Szenario 1: Build ausgegeben für einzelnes Java-Programm.

Der Java-Umsetzer kompiliert die Quellen- in Ausgabeklassen. Die Klasse wird in SCLM im Typ JAVACLAS gespeichert. Die Java-Kompilierungsausgabe wird im Typ JAVALIST gespeichert.

Klassenpfadabhängigkeiten können durch Speichern von abhängigen JARs im Klassenpfadverzeichnis berücksichtigt werden, das im \$GLOBAL-Memberparameter CLASSPATH_JARS angegeben ist. Weitere Informationen finden Sie unter „\$GLOBAL“ auf Seite 25.

- Szenario 2: Build für ARCHDEF (ARCHDEF ruft J2EEANT-Erstellungsscript auf, das durch das Schlüsselwort SINC referenziert wird) überlässt dem angegebenen Ant-Script die Erstellung. Der Java-Umsetzer selbst kopiert, wenn durch die ARCHDEF aufgerufen, nur die Ausgabeklassen in SCLM. Eine Zusammenfassungsdatei für den ANT-Build wird in JAVALIST gespeichert. Einzelne Java-Komponenten haben eine Ausgabetable, die in JAVALIST gespeichert wird.

JAVABIN

Der Sprachtyp ähnelt Java und wird beim Speichern von Java-Quellcode als ASCII in SCLM verwendet.

SQLJ Sprachtyp für SQLJ-Quellcode, der durch das Beispiel BWBTRANS definiert wird. SQLJ-Member, die mit diesem Sprachumsetzer definiert werden, rufen zum Zeitpunkt der Erstellung den SQLJ-Sprachumsetzer auf. Der SQLJ-Quellcode wird in Java-Quellcode konvertiert und in Klassen und serialisierte Objekte (.ser-Dateien) im Typ SQLJSER kompiliert. Optional können DBRM-Member auch im Typ DBRMLIB erstellt werden.

Anmerkung: Bei allen Objekten, z. B. JAR-, WAR- und EAR-Dateien, werden die internen komprimierten Quellteile in ASCII gespeichert, damit diese auf allen Plattformen verteilt werden können.

SCLM-Dateigruppen für JAVA/J2EE

Es wird empfohlen, SCLM-Ziel-Quellendateien mit RECFM=VB, LRECL=1024 für alle JAVA/J2EE-Quellen zu erstellen, die vom Toolkit-Client aus in SCLM gespeichert werden sollen, um lange Datensatztypen zu ermöglichen.

Die Editoren auf dem Eclipse-basierten Client erstellen Dateien mit verschiedener Datensatzlänge. Um die Integrität zu wahren, sollten die Host-Zieldatensätze in SCLM ebenfalls dem Typ RECFM=VB entsprechen. Wenn Datensätze mit festgelegter Satzlänge (RECFM=FB) verwendet werden, werden bei importierten Mitgliedern Leerzeichen am Ende des Datensatzes angehängt.

SCLM-Typen

Für die Unterstützung von JAVA/J2EE müssen verschiedene SCLM-Typen erstellt werden. Einige dieser Typen sind obligatorisch und müssen erstellt werden, damit die JAVA/J2EE-Unterstützung funktionieren kann.

Empfohlene Datensatzattribute für wichtige Typen

Für die folgenden SCLM-Typen werden Standard-Datensatzattribute des Typs DSORG=PO TRACKS(1,5) DIR=50 BLKSIZE=0 (durch das System festgelegt) empfohlen.

Darüber hinaus werden für Satzformat und Satzlänge folgende Attribute empfohlen:

SCLM-Typ	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

SCLM-Typ	RECFM	LRECL
Weitere Quell- Dateigruppentypen für Java/ J2EE	VB	1024

ARCHDEF

Der ARCHDEF-Typ enthält JAVA/J2EE-ARCHDEF-Member.

Die Teile mit ausgeschriebenem Namen in jedem ARCHDEF-Member stellen die JAVA/J2EE-Projektstruktur dar. Die ARCHDEF für ein bestimmtes Projekt kann beim Migrieren in neuen Projekten dynamisch auf dem Client erstellt werden oder aktualisiert werden, wenn einem vorhandenen Projekt neue Teile hinzugefügt werden.

Die SCLM ARCHDEF ist die primäre SCLM-Datei zum Definieren der Elemente eines JAVA/J2EE-Projekts. Hinsichtlich JAVA/J2EE-Anwendungen stellt das ARCHDEF dar, wie die J2EE-Anwendung im Arbeitsbereich des Client-IDE-Projekts strukturiert ist.

Die Projektdateistruktur der Anwendung wird in der ARCHDEF repliziert (unter Verwendung des Kurznamens des SCLM-Hosts, um die Struktur des ausgeschriebenen Namens zuzuordnen). Zusatzschlüsselwörter in der ARCHDEF, wie LINK, SINC und OUT1, geben in SCLM die J2EE-Spezifik dieses Projekts an. Der Quellcode umfasst ein JAVA/J2EE-Erstellungsscript, um den Erstellungsprozess dieses Projekts zu vereinfachen.

J2EEBLD

Der Typ J2EEBLD wird für die Java- und J2EE-Erstellungs- und Implementierungsprozesse benötigt und enthält folgende Elemente:

- J2EEBLD-Erstellungsscripts, die für den Ant-Erstellungs- und Implementierungsprozess verwendet werden.
- Java- und J2EE-ANTXML-Scripts, die für Erstellungen und Implementierungen aufgerufen werden sollen.
- \$GLOBAL gibt die Standardeigenschaften für das SCLM-Projekt für den JAVA/J2EE-Erstellungsprozess an. Weitere Informationen hierzu enthalten die Abschnitte „\$GLOBAL-Format“ auf Seite 9 und „\$GLOBAL“ auf Seite 25.

Anmerkung: Es werden Beispiel-Java- und -J2EE-ANTXML-Scripts bereitgestellt. Im Allgemeinen erfordern diese Scripts wenig oder keine Anpassung durch den Benutzer. Standort- und benutzerabhängige Variablen werden in den J2EEBLD-Scripts selbst angepasst, um Standard-ANTXML-Variablen zu überschreiben. Weitere Informationen finden Sie unter „JAVA/J2EE-Ant-XML-Erstellungsentwürfe“ auf Seite 16.

JAVALIST

Der JAVALIST-Typ ist für den Java-Buildprozess erforderlich und enthält Listenausgaben von Java-Builds.

J2EELIST

Der Typ J2EELIST wird für den J2EE-Erstellungsprozess benötigt und enthält Listenausgaben aus J2EE-Erstellungen.

DBRMLIB

Technisch gesehen ein DB2-Typ.

DBRMLIB wird für die Unterstützung von SQLJ benötigt und speichert die Datenbankanforderungsmodule.

JAVACLAS

Der Typ JAVACLAS wird benötigt für Java- und J2EE-Erstellungsprozesse und enthält Ausgabeklassendateien aus Erstellungen, die mit den JAVA- und J2EEANT-Sprachdefinitionen verknüpft sind.

J2EEEAR

Der Typ J2EEEAR wird für den J2EE-Erstellungsprozess benötigt und enthält EAR-Ausgaben aus Erstellungen, die mit der J2EEANT-Sprachdefinition verknüpft sind.

J2EEJAR

Der Typ J2EEJAR wird für JAVA/J2EE-Erstellungen benötigt und enthält JAR-Ausgaben aus Erstellungen, die mit der J2EEANT-Sprachdefinition verknüpft sind.

J2EEWAR

Der Typ J2EEWAR wird für den J2EE-Erstellungsprozess benötigt und enthält WAR-Ausgaben aus Erstellungen, die mit der J2EEANT-Sprachdefinition verknüpft sind.

SQLJSER

Der Typ SQLJSER wird für den J2EE-/SQLJ-Erstellungsprozess benötigt und speichert serialisierte SQLJ-Profile.

<Java/J2EE>-Typen

Für jedes JAVA/J2EE-Projekt, das in SCLM gespeichert wird, wird ein separater SCLM-Typ benötigt. Dadurch werden Konflikte in Dateien mit demselben Namen vermieden, die bei JAVA/J2EE-Projekten auftreten können. Weitere Informationen finden Sie unter „Zuordnen, J2EE-Projekte zu SCLM“ auf Seite 18.

SCLM-Memberformate

In diesem Abschnitt werden die SCLM-Memberformate beschrieben.

\$GLOBAL-Format

Das Format \$GLOBAL hat den Typ J2EEBLD und die Sprache J2EEPART. Es muss den Namen \$GLOBAL verwenden. Die Variablen werden im Tag-Sprachformat definiert.

\$GLOBAL gibt die Standardeigenschaften für das SCLM-Projekt für JAVA/J2EE-Erstellungsprozesse an. Dieses muss im SCLM-Typ J2EEBLD gespeichert werden.

Eine detaillierte Beschreibung der \$GLOBAL-Memberparameter finden Sie unter „\$GLOBAL“ auf Seite 25.

Sehen Sie sich das folgende Codemuster an:

```
<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"/>
```

Abbildung 3. \$GLOBAL - SCLMDT-Umgebungsvariablen

J2EE ARCHDEF-Format

Das Format J2EE-ARCHDEF hat den Typ ARCHDEF und die Sprache ARCHDEF.

Die ARCHDEF verwendet Standard-SCLM-Architektur-Schlüsselwörter, um SCLM anzuweisen, wie die Erstellung der ARCHDEF verarbeitet werden soll.

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

LKED Gibt an, dass dies eine LEC-ARCHDEF ist, und gibt die Sprache des ARCHDEF-Umsetzers an, der aufgerufen werden soll (für J2EE-ARCHDEFs ist dies immer J2EEOBJ).

INCLD

SCLM-Include der J2EE-Komponente. *SourceFile* ist der Name des Quellmembers (z. B. Java-Quelle), der in dieser ARCHDEF enthalten ist. *SourceType* ist der SCLM-Typ, der das Member enthält. In einer mit dem SCLM Developer Toolkit erstellten ARCHDEF ist ein Kommentar enthalten, der den vollständigen Dateinamen der Datei angibt, wie er im Projekt in der Arbeitsumgebung vorhanden war.

INCL SCLM-Include einer anderen verschachtelten ARCHDEF, z. B. der ARCHDEF, die das Manifest für eine EJB-Anwendung enthält.

SINC Quellen-Include des J2EEBLD-Erstellungsscripts. *BuildScriptName* ist der Name des Erstellungsscripts. Der Quellentyp ist immer J2EEBLD.

OUT1 Gibt den J2EE-Objektyp an, der durch diese ARCHDEF erstellt wird. Der Membername ist immer *. Der *J2EEOutputObjectType* ist entweder auf J2EEEAR, J2EEWAR oder J2EEJAR gesetzt. Das erstellte Member erhält einen Namen, der dem für die JAR-, WAR- oder EAR-Datei erstellten Kurznamen entspricht.

LIST Zusammenfassungsliste der Komponenten und Prüfung der ARCHDEF-Erstellung. Der Membername ist immer *. Der Quellentyp ist immer J2EELIST. Das Member erhält einen Namen mit einem Wert, der dem ARCHDEF-Member entspricht.

J2EE ARCHDEF-Beispiel: Das folgende Beispiel zeigt den 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
```

Abbildung 4. Sample Jar application (JAR) ARCHDEF

Das folgende Beispiel zeigt den WAR-Code:

```

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

```

Abbildung 5. Beispiel für Webanwendung (WAR) ARCHDEF

Das folgende Beispiel zeigt den EJB-Code:


```

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

```

Abbildung 6. Beispiel für EJB-Anwendung (EJB) ARCHDEF

Das folgende Beispiel zeigt den EAR-Code:

```

LKED  J2EE0BJ
*
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

```

Abbildung 7. Beispiel für EAR-Anwendung (EAR) ARCHDEF

Format des J2EE-Ant-Erstellungsscripts

Das Format des J2EE-Ant-Erstellungsscripts hat den Typ J2EEBLD und die Sprache J2EEANT. Der Name kann bis zu acht Zeichen lang sein und die Variablen werden im Tag-Sprachformat definiert. Die Erstellungsscripts für JAR, WAR und EAR ähneln sich sehr. Die unten angegebene Syntax wird für ein WAR-Erstellungsscript angezeigt. Für JAR- und EAR-Erstellungsscripts sind die Variablen identisch, mit der Ausnahme, dass EAR_NAME und JAR_NAME anstelle von WAR_NAME verwendet werden.

Im folgenden Beispiel wird das Beispielerstellungsscript angezeigt:

```
<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>
```

Abbildung 8. J2EE-Ant-Erstellungsscript

Kundendefinierte Variablen werden bei Erstellungsanforderungen während der Ausführung des Ant-Erstellungsscripts dynamisch durch die SCLM-Erstellungsscripts überschrieben. Diese Variablen werden auf die unter Tabelle 3 angezeigten Werte gesetzt.

Tabelle 3. Kundendefinierte Variablen

Variable	Beschreibung
J2EE-Projektname	Java/J2EE-Projekttyp, der erstellt wird. Dabei handelt es sich um einen temporären Projektnamen, der im Erstellungsscript für Ant für die Verwendung während der Erstellung festgelegt wird. Für diesen werden folgende Werte festgelegt: <ul style="list-style-type: none">• J2EE EAR-Projekt• J2EE WAR-Projekt• J2EE EJB-Projekt• JAVA-Projekt Diese Variable muss nicht angepasst werden.
SCLM_ARCHDEF	ARCHDEF-Name oder die ARCHDEF, für die ein Build ausgeführt wird.
SCLM_ANTXML	Name der Entwurfs-Ant-XML, die für den Build verwendet wird.

Tabelle 3. Kundendefinierte Variablen (Forts.)

Variable	Beschreibung
SCLM_BLDMAP	Wert „Yes“ oder „No“. Wenn „Yes“ festgelegt werden soll, schließen Sie die SCLM-Buildzuordnung im Verzeichnis MANIFEST in JAR, WAR oder EAR ein. Stellt die Prüfung und die Buildzuordnung der eingeschlossenen Teile bereit.
JAVA_SOURCE	Der Wert 'Ja' oder 'Nein'. Bei 'Ja' schließen Sie die Java-Quelle in JAR, WAR oder EAR ein.
CLASSPATH_JARS	Klassenpfadverzeichnis von z/OS UNIX System Services, das zur Auflösung der Klassenpfadabhängigkeiten bei der Erstellung verwendet wird. Alle JAR-Dateien in diesem Verzeichnis werden im Klassenpfad verwendet.
CLASSPATH_JARS_FILES	Namen der einzelnen JAR- und Klassendateien, die bei der Erstellung eingeschlossen werden sollen. Dies kann in Form einer Liste auf folgende Weise geschehen: <property name="CLASSPATH_JARS_FILES" value="V2J4.jar,V2J3.jar" />
ENCODING	ASCII- oder EBCDIC-Codepage für JAVA. Dabei handelt es sich um die Codepage, in der der JAVA-Quellcode auf dem z/OS-Host gespeichert wird. Beispiel: <ul style="list-style-type: none"> • Für ASCII sollte die JAVA-Standard-Codepage ISO8859-1 sein. • Für EBCDIC sollte die JAVA-Standard-Codepage IBM-1047 sein.
JAR_FILE_NAME EJB_NAME WAR_NAME EAR_NAME	Name der JAR-, EJB-JAR-, WAR- oder EAR-Datei.
DEBUG_MODE	Auf „on“ gesetzt, um Developer Toolkit anzuweisen, keine Erstellungsdateien aus dem Verzeichnis WORKAREA zu löschen. Dies ist sinnvoll, wenn Sie die Struktur einer erstellten Java/J2EE-Anwendung prüfen müssen.

CLASSPATH-Abhängigkeiten: Java-Quellcode innerhalb einer ARCHDEF kann Klassenpfadabhängigkeiten mit anderen Java-Bibliotheken oder -klassen haben. Wenn diese Abhängigkeiten sich in Java-Komponenten befinden, die in derselben ARCHDEF-Struktur enthalten sind, werden diese Klassenpfadabhängigkeiten im Rahmen der ARCHDEF-Erstellung aufgelöst (unabhängig davon, ob der Erstellungsmodus bedingt oder erzwungen ist).

Eine J2EE-ARCHDEF-Komponente kann jedoch Klassenpfadabhängigkeiten mit externen JARs oder sogar Mitgliedern aus anderen ARCHDEFs haben. In diesem Fall kann das mit der ARCHDEF verknüpfte J2EE-Erstellungsscript die Klassenpfadabhängigkeiten mit folgenden Schlüsselwörtern steuern:

CLASSPATH_JARS

Dies ist der Name eines Verzeichnisses im Dateisystem von z/OS UNIX System Services, das alle externen, abhängigen JAR-Dateien und -klassen für diese bestimmte ARCHDEF-Erstellung enthalten kann.

Dieses Verzeichnis kann mit CLASSPATH-Dateien über die Client-Team-Funktion „JAR-Dateien hochladen“ aktualisiert werden, um JAR-Dateien vom Client in das Klassenpfadverzeichnis zu kopieren. Darüber hinaus steht die Funktion „Datei von SCLM in Klassenpfad kopieren“ zur Verfügung, um in SCLM gespeicherte JAR-Dateien in das Klassenpfadverzeichnis zu kopieren.

CLASSPATH_JARS_FILES

Dieses Schlüsselwort kann verwendet werden, um einzelne JAR- oder Klassendateien auszuwählen, die im Klassenpfad verwendet werden sollen. Wenn dieses Schlüsselwort verwendet wird, werden die aufgelisteten JAR-/Klassendateien aus SCLM abgerufen. Wenn diese nicht in SCLM ge-

speichert sind, wird das durch CLASSPATH_JARS referenzierte Verzeichnis bei der Abfrage durchsucht. Wenn dieses Schlüsselwort verwendet wird, werden nur die aufgelisteten Dateien im Klassenpfad verwendet.

J2EE-Beispielscripts: Das folgende Beispiel zeigt das 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>
```

Abbildung 9. JAR-Beispiel für J2EE-Erstellungsscript

Das folgende Beispiel zeigt das 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"/>
</path>
<fileset dir="${CLASSPATH_JARS}" includes="**/*.jar, **/*.zip"/>
</ANTXML>
```

Abbildung 10. WAR-Beispiel für J2EE-Erstellungsscript

Das folgende Beispiel zeigt das 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>

```

Abbildung 11. EJB-Beispiel für J2EE-Erstellungsscript

Das folgende Beispiel zeigt das 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>

```

Abbildung 12. EAR-Beispiel für J2EE-Erstellungsscript

JAVA/J2EE-Ant-XML-Erstellungsentwürfe

In diesem Abschnitt werden Beispiellentwürfe für die Ant-Erstellung aufgelistet, die in der Bibliothek FEK.SFEKSAMV bereitgestellt werden. Diese Beispielmuster können in den SCLM-Typ J2EEBLD in der SCLM-Hierarchie kopiert werden, um durch die JAVA/J2EE-Erstellungsscripts referenziert und verwendet zu werden. Die JAVA/J2EE-Erstellungsscripts sind Eigenschaftsvariablendateien, die die Ant-XML-Entwurfsdateien überschreiben.

Die angegebenen Beispiellentwürfe für die J2EE-Erstellung zur Erstellung einer einfachen JAR-Datei, eines SQLJ-Projekts, einer EJB-JAR-, WAR- oder EAR-Datei oder für die Implementierung können im Allgemeinen ohne Anpassung durch den Benutzer verwendet werden. Beachten Sie jedoch, dass manche J2EE-Projekte möglicherweise nicht dem Standardmodell entsprechen. In diesem Fall müssen die bereitgestellten Ant-XML-Entwürfe angepasst werden.

Anmerkung: JAVA/J2EE-Erstellungsscripts können mithilfe der Clientanwendung von SCLM Developer Toolkit erstellt werden. Diese Erstellungsscripts verwenden einen referenzierten Ant-XML-Entwurf (wie unten angegeben) und eine ARCHDEF im JAVA/J2EE-Erstellungsprozess.

Eine detaillierte Beschreibung der Erstellungsscripts, Ant-Entwürfe und Beispiele zum JAVA/J2EE-Erstellungsprozess ist im Benutzerhandbuch für SCLM Developer Toolkit enthalten, das mit dem Client-Plug-in geliefert wird.

BWBJAVAA

Beispielentwurf für Ant-XML-JAVA-Erstellung

Dieser Ant-Entwurf wird von einem Java-Build-Script verwendet, um mehrere Java-Programme zu kompilieren und optional eine JAR-Datei (Java Archive) zu erstellen, deren Struktur durch eine angegebene ARCHDEF festgelegt wird.

BWBEJBA

Beispielentwurf für Ant-XML-J2EE-EJB-Erstellung

Dieser Ant-Entwurf wird von einem J2EE-Erstellungsscript verwendet, um ein EJB-Projekt zu kompilieren oder zu erstellen. Dabei wird üblicherweise eine EJB-JAR-Datei erstellt, deren Struktur von einer angegebenen ARCHDEF bestimmt wird.

BWBWEBA

Beispielentwurf für Ant-XML-J2EE-WEB-Erstellung

Dieser Ant-Entwurf wird von einem J2EE-Erstellungsscript verwendet, um ein WEB-Projekt zu kompilieren oder zu erstellen, wobei üblicherweise eine WEB-Archivdatei (WAR) erstellt wird.

BWBEARA

Beispielentwurf für Ant-XML-J2EE-EAR-Assemblierung

Dieser Ant-Entwurf wird von einem J2EE-Erstellungsscript als Assemblierungsprozess in Vorbereitung für die J2EE-Anwendungsimplementierung verwendet. Bei diesem Prozess werden EAR-Dateien erstellt, die auf einem Webanwendungsserver implementiert werden können, z. B. auf einem WebSphere-Anwendungsserver.

BWBSQLB

Beispiel für Java-/SQLJ-Erstellungsscript

Dieser Ant-Entwurf wird von einem J2EE-Erstellungsscript zur Kompilierung oder Erstellung eines JAR-Projekts verwendet, das SQLJ verwendet.

BWBSQLBE

Beispiel für EJB-/SQLJ-Erstellungsscript

Dieser Ant-Entwurf wird von einem J2EE-Erstellungsscript zur Kompilierung oder Erstellung eines EJB-Projekts verwendet, das SQLJ verwendet.

BWBC9DTJ

Beispiel für die Konvertierung von Cloud 9 in SCLM DT

BWBDEPJ1

Beispiel zum Aktualisieren von SQLJ-SER-Dateien innerhalb einer JAR-Datei bei der Implementierung mit db2sqljcustomize

BWBDEPJ2

Beispiel für db2sqljcustomize, wobei der ausgeschriebene Name der Eigenschaft die angegebene JAR aus der angezeigten Gruppe kopiert und Positionen in SCLM in das durch "dest" angegebene Zielverzeichnis schreibt.

BWBDEPJ3

Diese Beispielroutine passt die in den ausgewählten JAR-Dateien enthaltenen SER-Dateien für die Implementierung mit db2sqljcustomize an.

BWBTRANX

Beispiel-SCLM-Erstellungsumsetzer für die Benachrichtigung bei SYSXML-Erstellungsfehlern für COBOL.

Zuordnen, J2EE-Projekte zu SCLM

IBM SCLM Developer Toolkit bietet die Möglichkeit, Projekte in SCLM zu verwalten, zu erstellen und zu implementieren. In diesem Abschnitt wird beschrieben, wie die SCLM-Projektstruktur konfiguriert werden muss, um die Entwicklung verteilter Anwendungen wie JAVA/J2EE zu unterstützen.

Bei vielen JAVA/J2EE-Projekten wird eine ausführbare EAR-Datei erstellt. Diese Anwendung ist eine Gruppe von Projekten, üblicherweise EJBs und Webanwendungen. Innerhalb der IDE-Umgebungen werden diese im Allgemeinen als einzelne Projektstrukturen entwickelt, die mit einem EAR-Projekt verbunden sind.

Diese Strukturform mit mehreren Projekten wird nicht direkt in SCLM abgebildet. Das bedeutet, dass ein SCLM-Projekt nicht mit einem anderen SCLM-Projekt verknüpft werden kann, um eine zusammengefasste Projektstruktur zu erhalten. SCLM bietet jedoch durch die Verwendung von Typen die Möglichkeit, eine Struktur mit mehreren Projekten innerhalb eines SCLM-Projekts zu erstellen.

SCLM-Projekte können mit mehreren Quellentypen definiert werden. Jeder Typ kann ein einzelnes IDE-Projekt enthalten. Wenn man mehrere Eclipse-IDE-Projekte in SCLM ohne eine Art der Trennung speichern würde, würden die Klassenpfad- und Projektdateien des Projekts bei der Hinzufügung der Projekte in SCLM überschrieben werden. Die Verwendung verschiedener Quellentypen ermöglicht die sichere Speicherung dieser und aller anderen mit dem Projekt verknüpften Dateien in SCLM.

Diese Zuordnung bewirkt, dass die IDE-Projekte in SCLM unabhängig gespeichert werden, wobei der Typ als wichtigstes Differenzierungsmerkmal dient. EJB1 wird z. B. im SCLM-Projekt SCLMPRJ1 unter Typ EJB1 gespeichert. Durch die Verwendung dieser Struktur ist es möglich, die IDE-Projektstruktur als unabhängige Typen innerhalb des SCLM-Projekts abzubilden.

Anmerkung:

1. Es ist nicht notwendig, einen Projektnamen in der IDE dem SCLM-Typnamen zuzuordnen. Diese Namen existieren unabhängig voneinander.
2. Typnamen dürfen höchstens acht Zeichen lang sein. Daher kann ein IDE-Projekt mit dem Namen „ProjectOne“ nicht den entsprechenden Typnamen „ProjectOne“ haben. Sie können stattdessen „Proj1“ verwenden.

Es ist daher wichtig, in der SCLM-Projektstruktur zu berücksichtigen, dass verschiedene IDE-basierte Projekte in einer einzelnen SCLM-Projektstruktur zugeordnet werden. Bei großen SCLM-Projekten kann es problematisch sein, zusätzliche

Projekttypen hinzuzufügen, da hierbei eine Änderung der SCLM-Projektdefinition, eine Neuerstellung der SCLM-Projektdefinition und die Dateizuordnung für die neuen Typen erforderlich ist.

Diese Struktur ist nicht auf Projekte im J2EE-Stil beschränkt, sondern kann auch in Situationen angewendet werden, in denen mehrere Projekte entwickelt werden, die eine Form der Abhängigkeit voneinander beinhalten.

Empfehlungen für die Zuordnung von J2EE-Projekten in SCLM

Die folgende Liste enthält Empfehlungen für die Zuordnung von J2EE-Projekten (und anderen) in SCLM:

- Identifizieren Sie die J2EE-Projektkomposition in Bezug auf EJBs, Webanwendungen usw. Diese Informationen werden für die Planung der SCLM-Projektstruktur benötigt.
- Erstellen Sie für jede J2EE-IDE-Projektkomponente einen zugehörigen Typ im SCLM-Projekt. Es ist sinnvoll, hierfür eine aussagefähige Namenskonvention zu verwenden. Es ist möglich, die IDE-Projekte unabhängig vom SCLM-Typ zu benennen, eine entsprechende Korrelation erleichtert jedoch die Verwaltung erheblich.
- Da die Projektanforderungen sich ändern können, sollten Sie zusätzliche Typdefinitionen erstellen, um eine problemlose Hinzufügung von anderen Komponenten zu ermöglichen, z. B. von zusätzlichen EJBs. Zusätzliche Services können durch die Typstruktur vorbereitet werden.
- Die Zuordnung mehrerer IDE-Projekte zu einem einzigen SCLM-Projekt wird durch die Verwendung von Typen unterstützt. Es ist außerdem sinnvoll, eine Paketierungsstruktur anzuwenden, die die Typdefinition für das jeweilige Projekt berücksichtigt.
- Paketierungskonventionen im Java-Stil können auch auf Projektebene definiert werden, um Quellennamenskollisionen zu vermeiden.
- Wenn die IDE-Struktur mehrere Projekte umfasst, empfiehlt es sich, diese Struktur in SCLM mit Typen zu replizieren.

Die Verwendung mehrerer SCLM-Typen zur Speicherung einzelner IDE-Projekte ist auch bei der Verarbeitung der ARCHDEF-Struktur für die Erstellung dieser IDE-Projekte zu berücksichtigen.

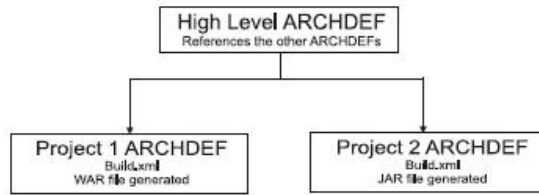


Abbildung 13. SCLM-Buildhierarchie

Die ARCHDEF-Datei enthält die Liste der Dateien, die zu einem Build gehören. In einem J2EE-Kontext kann bei einer Erstellung eine EAR-Datei erstellt werden, die aus mehreren WAR- und JAR-Dateien besteht. Diese Isolation von Projekten ähnelt der Typstruktur, die das Projekt in SCLM definiert. Durch die Verwendung einer übergeordneten ARCHDEF, die sich auf die zum Build gehörenden Teile bezieht, wird eine strukturierte Erstellungsumgebung bereitgestellt. Dies betrifft die effektive Definition der Projektstruktur beim Definieren der Typen in SCLM.

Eine strukturierte Definition des Projekts bietet folgende Vorteile:

- Migration von Dateien aus einem SCLM-Projekttyp oder einer ARCHDEF in ein IDE-Projekt, ohne dass die einzelnen Teile bekannt sein müssen.
- Die ARCHDEF-Struktur basierend auf der Typdefinition ermöglicht es auch, dass Projektabhängigkeiten effektiver zugeordnet werden können. Es ist allgemein üblich, dass IDE-Projekte auf andere IDE-Projekte im Arbeitsbereich verweisen. Die Verwendung des Schlüsselworts SCLM INCL in ARCHDEFs unterstützt diese Absicht, da andere IDE-Projekte, auf die von anderen ARCHDEFs verwiesen wird, durch Verschachtelung der ARCHDEFs innerhalb von ARCHDEFs höherer Ebene eingeschlossen werden können.

Beim Erstellen von Anwendungen mit Verweisen oder Abhängigkeiten auf andere Buildobjekte, z. B. JAR-Dateien, andere Projekte oder andere Klassen, können folgende Vorgehensweisen verwendet werden:

1. Schließen Sie den Verweis auf die JAR-Datei durch die INCLD-Anweisung in der ARCHDEF ein. Dadurch wird die Anwendung mit dem Bibliotheksverweis im finalen Buildpaket erstellt.
2. Schließen Sie das IDE-Projekt als verschachtelte INCL-SCLM-Projekt-ARCHDEF ein.
3. Schließen Sie die abhängige JAR-Datei im CLASSPATH-Verzeichnis ein.
 - Wenn das IDE-Projekt eine JAR-Datei referenziert, jedoch nicht Teil des finalen Buildpakets sein soll, kann die Bibliotheksdatei durch Verwendung des Services „JAR-Dateien hochladen“ in Developer Toolkit in den System-CLASSPATH kopiert werden. Dadurch kann dieser Service auch von anderen

SCLM-Projekten aus aufgerufen werden. Zum Zeitpunkt der Erstellung wird das IDE-Projekt aufgelöst, das auf die JAR-Datei verweist. Während der Laufzeit muss die JAR-Datei jedoch in einer PATH-Anweisung verfügbar sein.

- Sie können auf eine JAR-Datei im selben SCLM-Projekt verweisen, indem Sie die Eigenschaft `CLASSPATH_JARS_FILES` im Erstellungsscript verwenden.

SCLM Developer Toolkit - Implementierung

SCLM Developer Toolkit bietet mehrere Implementierungsfunktionen. Sie können Unternehmensarchivdateien (Enterprise Archive, EAR) auf jedem WebSphere Application Server (WAS) implementieren. Zusätzlich kann jede mit dem SCLM Developer Toolkit erstellte oder gesteuerte Komponente mit einem anpassbaren Implementierungsscript verteilt werden. Es werden Beispielscripts bereitgestellt, mit denen eine EAR-Datei unter Verwendung der Befehle für sicheres Kopieren (SCP) und sicheres FTP (SFTP) auf einen fernen Host kopiert werden kann.

Im Zentrum der Implementierung stehen im wesentlichen zwei Scripts. Das erste Script, das durch den Benutzer geändert wird, ist das Eigenschaftenscript. Es enthält eine Liste der Parameter für den Implementierungsvorgang. Das zweite ist das Aktionsscript, das die erforderlichen Schritte für die Ausführung des Implementierungsvorgangs enthält.

Die Implementierung wird vom Client-Plug-in von SCLM Developer Toolkit aus gestartet. Der Implementierungstyp wird durch Auswahl der entsprechenden Schaltfläche auf dem Implementierungsbildschirm ausgewählt. Abhängig von der ausgewählten Implementierungsaktion werden entsprechende Daten im Eigenschaftenscript ausgefüllt. In den meisten Scripts ist eine Eigenschaft mit dem Namen `SCLM_ANTXML` vorhanden, die den Membernamen des entsprechenden Aktionsscripts enthält. Developer Toolkit verwendet das generierte Eigenschaftenscript und überträgt es auf das Aktionsscript, bevor das resultierende Aktionsscript aufgerufen wird.

Im Folgenden wird eine Liste der Beispiele für Ant-Implementierungsaktionsscripts angezeigt, die in der Bibliothek `FEK.SFEKSAMV` bereitgestellt werden. Diese Beispielmembers können in den SCLM-Typ `J2EEBLD` in der SCLM-Hierarchie kopiert werden, um von den generierten Eigenschaftenscripts referenziert und verwendet zu werden. Die generierten Eigenschaftenscripts sind Eigenschaftenscriptdateien, mit denen die unten dargestellten Ant-XML-Implementierungsaktionsscripts überschrieben werden. Diese Scripts müssen mit einer Texttypsprache, z. B. `TEXT` oder `J2EEPART`, gespeichert werden.

Member	Beschreibung
BWBDEPLA	WAS-EAR-Implementierung.
BWBRDEPL	Ferne WAS-EAR-Implementierung.
BWBCOPY	Secure-Copy-Implementierung. Kopiert ein Erstellungsobjekt über SCP von einem Host auf einen anderen.
BWBSFTP	Sichere FTP-Implementierung. Kopiert ein Buildobjekt von einem Host auf einen anderen über SFTP.

Um diese Erstellungsscripts von mehreren Gruppen aus aufrufen zu können, muss der Administrator die Scripts erstellen und auf die höchste im Projekt verfügbare Gruppenebene umstufen.

Abhängig von den generierten Scripttypen unterscheidet sich die Vorgehensweise geringfügig.

WebSphere Application Server (WAS) - Implementierung

Bei der Implementierung von WebSphere Application Server (WAS) verweist die Eigenschaft SCLM_ANTXML nicht auf ein Ant-Aktionsscript, sondern stattdessen auf ein JACL-Aktionsscript. Alternativ können Sie das Tool „wsadmin“ verwenden, das im Lieferumfang von WAS unter z/OS enthalten ist.

Für das Tool „wsadmin“ wird ein JACL-Script als Leitfaden für den Implementierungsprozess benötigt. Wenn diese Implementierungsmethode verwendet wird, muss das JACL-Script als ASCII-Datei in einem z/OS UNIX-Verzeichnis erstellt werden, bevor der Implementierungsprozess aufgerufen werden kann.

Bei der Anpassung von Developer for System z wird ein Beispiel-(ASCII-)JACL-Script unter /etc/rdz/sclmdt/CONFIG/scripts/deploy.jacl bereitgestellt (wobei /etc/SCLMDT das Standard-etc-Verzeichnis für SCLM Developer Toolkit ist).

Zusätzliche JACL-Beispiele sind in der Dokumentation von WebSphere Application Server (WAS) enthalten.

Die Verzeichnispositionen des wsadmin-Tools (wsadmin.sh) und des JACL-Scripts (deploy.jacl) können auf der Vorgabenseite unter **Team > SCLM-Vorgaben > Build-Script-Optionen** konfiguriert werden. Der SCLMDT-Client wird verwendet, um ein Implementierungsscript zu generieren, das anschließend für die Erstellung verwendet werden kann. (Der Implementierungsprozess wird durch eine Implementierungsfunktionsanforderung für das Implementierungsscript ausgelöst, das im SCLM-Typ J2EEBLD gespeichert ist).

Die Beispiel-Aktionsscripts, die im SCLM-Typ J2EEBLD für die WAS-Implementierung oder ferne WAS-Implementierung gespeichert werden müssen, sind BWBDEPLA und BWRDEPL.

Implementierung von SCLM in UNIX System Services

SCLM Developer Toolkit bietet eine Möglichkeit zur Implementierung von Dateien, die im SCLM-Repository für das Dateisystem von z/OS UNIX System Services auf derselben logischen Partition gespeichert sind. Dadurch steht eine einfache Methode zur Verfügung, um Objekte, die von SCLM erstellt wurden, in einer Umgebung zu implementieren, in der diese ausgeführt oder über die sichere Implementierung auch auf einem fernen Host implementiert werden können. Diese Vorgehensweise wird unten beschrieben.

Es ist kein Beispielaktionsscript für diese Aktion vorhanden. Wählen Sie die entsprechenden Member aus SCLM aus und verwenden Sie die Schaltfläche „SCLM-Member einschließen“, um das benötigte Eigenschaftenscript zu generieren. Dadurch werden die Dateien von der ausgewählten SCLM-Speicherposition in ein angegebenes Verzeichnis im Dateisystem von z/OS UNIX System Services kopiert. Dieses Verzeichnis muss bereits vorher vorhanden sein. Andernfalls wird ein Fehler angezeigt.

Sichere Implementierung

Diese Option bietet eine Möglichkeit, implementierbare Objekte auf einen fernen Host zu kopieren, indem die Befehle für sicheres Kopieren (SCP) und sicheres FTP (SFTP) verwendet werden. Indem die Eigenschaftenscripts für sichere Implementierung gemeinsam mit den Include-SCLM-Membren verwendet werden, können die benötigten Dateien aus der SCLM-Hierarchie ausgewählt werden, an eine Speicher-

position im Dateisystem von z/OS UNIX System Services kopiert werden und anschließend unter Verwendung der Befehle für sicheres Kopieren (SCP) und sicheres FTP (SFTP) von diesem z/OS UNIX System Services-Dateisystem auf das Zielsystem kopiert werden.

Die Beispiel-Aktionsscripts, die für die sichere Implementierung im SCLM-Typ J2EEBLD gespeichert werden müssen, sind BWBSCOPY und BWBSFTP.

Anmerkung: Die IBM Ported Tools for z/OS müssen bestellt, installiert und konfiguriert werden, um eine sichere Implementierung zu unterstützen. Verwenden Sie den *IBM Rational Developer for System z Host Planning Guide* (IBM Form GI11-3123-01), um zu erfahren, wie Sie Ported Tools erhalten können. Die Installation und Anpassung dieses Produkts ist nicht in diesem Handbuch beschrieben.

IBM Ported Tools for z/OS stellt folgende Funktionen bereit:

- SCP für das Kopieren von Dateien zwischen Netzen. Dabei handelt es sich um eine Alternative zu RCP.
- SFTP für Dateiübertragungen über einen verschlüsselten SSH-Transport. Dabei handelt es sich um ein interaktives Dateiübertragungsprogramm, das FTP ähnelt.

Public-Key-Authentifizierung

Die Public-Key-Authentifizierung stellt eine Alternative zur interaktiven Anmeldung dar und kann im Rahmen des sicheren Implementierungsvorgangs von Developer Toolkit automatisiert werden.

Damit die Public-Key-Authentifizierung wie gewünscht arbeiten kann, können Sie entweder eine Ersatzbenutzer-ID für die Implementierung verwenden oder alle Benutzer konfigurieren, für die Sie Implementierungsfunktionen bereitstellen möchten.

Anweisungen dazu, wie die automatisierte, schlüsselbasierte Authentifizierung mithilfe von ssh-agent und ssh-add definiert wird, finden Sie im Benutzerhandbuch *IBM Ported Tools for z/OS User's Guide*. Informationen zur Verwendung der Ersatzbenutzer-ID für SCLM Developer Toolkit finden Sie unter Kapitel 4, „SCLM-Sicherheit“, auf Seite 51.

Weitere Implementierungsoptionen

Sie können auch eigene Ant-Scripts erstellen, um die Implementierung auf andere Weise auszuführen. In Ihren Scripts können Sie durch Verwendung des Ant-Tags `<exec>` ein beliebiges Programm aufrufen, das im Dateisystem z/OS UNIX System Services zur Verfügung steht. Wenn Sie diese Methode verwenden, können die Erstellungsscripts andere Programme, z. B. FTP aufrufen, um die Implementierung auszuführen. Weitere Informationen zur Erstellung von Ant-Scripts finden Sie in der Ant-Onlinedokumentation unter <http://ant.apache.org/>.

ASCII- oder EBCDIC-Speicheroptionen

Aus dem SCLM Developer Toolkit-Plug-in übertragene Quellendateien können in SCLM entweder als ASCII oder EBCDIC gespeichert werden.

Generell werden alle Quellen in SCLM in EBCDIC gespeichert, um direkt in ISPF/SCLM unter z/OS angezeigt und bearbeitet werden zu können. Wenn Sie den Code nicht direkt auf dem Host anzeigen oder bearbeiten möchten, können Sie den Code direkt (d. h. als übertragene Binärdatei) an der Position speichern, wo die Quelle in SCLM gespeichert wird. Dabei wird die ASCII/UNICODE-Codepage des ursprünglichen Clients verwendet. Da keine Umsetzung von ASCII in EBCDIC erforder-

derlich ist, erhalten Sie auf diese Weise ein besseres Leistungsverhalten bei der Speicherung und beim Import von großen Projekten aus SCLM sowie bei JAVA/J2EE-Erstellungen.

SCLM Developer Toolkit ermittelt, ob eine Datei binär übertragen wurde oder ob eine Umsetzung von ASCII in EBCDIC stattfindet, indem die SCLM-Sprache überprüft wird, die der Datei oder dem Member zugeordnet ist. Anschließend überprüft SCLM Developer Toolkit, ob diese SCLM-Sprache in der Datei TRANSLATE.conf mit einem TRANLANG-Schlüsselwort eingetragen ist.

ASCII/EBCDIC-Sprachumsetzer

Tabelle 4. SCLM-Sprachumsetzer und ASCII/EBCDIC

SCLM-Sprachumsetzer	Beschreibung
JAVA	Java-Quellcode-Member gespeichert als EBCDIC. Erstellt mit dem Beispiel BWBTRAN1.
SQLJ	SQLJ-Member gespeichert als EBCDIC. Erstellt durch Verwendung des Beispiels BWBTRANS.
JAVABIN	Java-Quellen-Member gespeichert als ASCII. Erstellt mit dem Beispiel BWBTRAN1.
J2EEPART	J2EE-Dateien, bei denen keine Syntaxanalyse erforderlich ist, gespeichert als EBCDIC. Erstellt mit dem Beispiel BWBTRAN1.
J2EEBIN	J2EE-Dateien, bei denen keine Syntaxanalyse erforderlich ist, gespeichert als Binär- oder ASCII-Dateien. Erstellt mit dem Beispiel BWBTRAN1.
SQLJ	SQLJ-Quellenmember gespeichert als EBCDIC. Erstellt durch Verwendung des Beispiels BWBTRANS.
SQLJBIN	SQLJ-Quellenmember, gespeichert als ASCII. Erstellt durch Verwendung des Beispiels BWBTRANS.
TEXT	Standard-TEXT-Umsetzer, wenn keine Syntaxanalyse erforderlich ist, gespeichert als EBCDIC. Erstellt mit dem Beispiel BWBTRAN1.
BINARY	Standard-Binärsprachumsetzer, wenn keine Syntaxanalyse erforderlich ist. Erstellt mit dem Beispiel BWBTRAN1.

Als Standardverwendung wird ASCII/EBCDIC-Konvertierung vorausgesetzt. Dies bedeutet, dass Dateien, die im Eclipse-Plug-in angezeigt und bearbeitet werden, auch direkt auf dem Host in ISPF/SCLM angezeigt und bearbeitet werden können.

Die Verwendung von ASCII (binär übertragen) empfiehlt sich bei der Projektmigration sowie zur Leistungsverbesserung beim Importieren und Erstellen, da diese Dateien keine Umsetzung erfordern. Dies gilt nur, wenn keine Bearbeitung in ISPF/SCLM erforderlich ist.

Abhängig vom verwendeten SCLM-Sprachumsetzer kann die Quelle entweder in ASCII oder EBCDIC erstellt werden.

Um die Nutzbarkeit auf verschiedenen Plattformen zu gewährleisten, werden alle implementierbaren Dateien erstellt, wie JAR-, WAR- und EAR-Dateien. Dadurch entsprechen alle enthaltenen Objekte dem Typ ASCII, unabhängig davon, ob ein Teil des Quellcodes als EBCDIC gespeichert ist.

Hinweis zur JAVA/J2EE-Erstellung: Wenn der Java-Quellcode in ASCII gespeichert wird, muss das Erstellungsscript die ASCII-Codepage mit der Eigenschaftsvariablen ENCODING angeben, damit der Java-Quellcode ordnungsgemäß kompiliert werden kann.

Beispiel:

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

Das aufgerufene Ant-Script wird den Java-Befehl mit dem Wert ENCODING=ISO8859-1 verwenden, um die ASCII-Quelle zu kompilieren. Die Standard-ENCODING-Codepage ist die EBCDIC-Codepage IBM-1047.

\$GLOBAL

Als Teil des JAVA/J2EE-Erstellungsprozesses werden einige zusätzliche Informationen benötigt, um die Erstellungen erfolgreich ausführen zu können. Wenn die Erstellungen in z/OS UNIX System Services ausgeführt werden, werden Informationen benötigt, z. B. die Speicherposition des Java-Produkts, des Ant-Produkts sowie die Speicherposition der Konfigurationsdateien und des Arbeitsbereichs von SCLM Developer Toolkit.

Darüber hinaus kann es erforderlich sein, verschiedene Versionen von Ant oder Java für verschiedene SCLM-Entwicklungsgruppen zu verwenden. Das Member \$GLOBAL kann daher gruppenspezifisch sein. Die Umgebungsvariablen, die in \$GLOBAL festgelegt sind, können durch spezifische Erstellungsscript-Variableneinstellungen überschrieben werden.

Anmerkung: Wenn die Konfigurationsdatei „ant.conf“ verwendet wird, überschreibt die Angabe JAVA_HOME die Angabe JAVA_BIN in \$GLOBAL bei Java/J2EE-Projektkompilierungen. Dies ist nicht wahr, wenn die Ant-Konfiguration in rsed.envvars erfolgt. Dies wird in *Rational Developer for System z Hostkonfiguration* (IBM Form SC12-4062-04) beschrieben.

Ein Beispielmember BWBGL0B wird in der Bibliothek FEK.SFEKSAMV bereitgestellt. Dieses Beispielmember muss im SCLM-Typ J2EEBLD in der SCLM-Hierarchie als Member \$GLOBAL kopiert und mit einer gültigen Nicht-Parsing-Sprache gespeichert werden, z. B. TEXT (wie im Sprachumsetzer FLM@TEXT in der Bibliothek SISPMACS bereitgestellt).

Das Member \$GLOBAL stellt dem JAVA/J2EE-Erstellungsprozess derzeit folgende Informationen zur Verfügung:

Tabelle 5. \$GLOBAL-Variablen

Variable	Beschreibung
ANT_BIN	Dateisystem-Verzeichnispfad von z/OS UNIX System Services für die Ant-Laufzeit Beispiel: <pre><property name="ANT_BIN" value="/usr/lpp/apache-Ant-1.6.0/bin/Ant"/></pre>
JAVA_BIN	z/OS UNIX System Services-Dateisystem-Verzeichnispfad für Java-Kompilierung/Laufzeit Beispiel: <pre><property name="JAVA_BIN" value="/usr/lpp/java/5.0/bin"/></pre>
_SCLMDT_WORK_HOME	Die Speicherposition des WORKAREA-Verzeichnisses von SCLM Developer Toolkit Beispiel: <pre><property name="_SCLMDT_WORK_HOME" value="/var/rdz"/></pre>

Tabelle 5. \$GLOBAL-Variablen (Forts.)

Variable	Beschreibung
_SCLMDT_CONF_HOME	Die Speicherposition des CONFIG-Verzeichnisses von SCLM Developer Toolkit Beispiel: <code><property name="_SCLMDT_CONF_HOME" value="/etc/rdz/sclmdt"/></code>
CLASSPATH_JARS	Klassenpfadverzeichnis des z/OS UNIX System Services-Dateisystems, das für JAVA-Kompilierungen verwendet wird. Alle JAR-Dateien in diesem Verzeichnis werden im Klassenpfad verwendet. Beispiel: <code><property name="CLASSPATH_JARS" value="/var/rdz/CLASSPATH"/></code>
TRANTABLE	VSAM-Datei, die die Namensumsetzungen für Lang-/Kurznamen enthält Beispiel: <code><property name="TRANTABLE" value="FEK.#CUST.LSTRANS.FILE"/></code>
DEBUG_MODE	Setzen Sie diese Einstellung auf „on“, wenn Developer Toolkit keine Java/J2EE-Builddateien aus dem z/OS UNIX System Services-Dateisystem löschen soll. Dies ist sinnvoll, wenn Sie die Struktur der erstellten Ausgaben im USS-Dateisystem für Fehlerbehebungs Zwecke anzeigen möchten.

Wenn diese Variablen für alle Gruppenebenen im SCLM-Projekt festgelegt werden sollen, empfiehlt es sich, ein einziges \$GLOBAL-Member auf der höchsten Hierarchieebene zu erstellen. Wenn der JAVA/J2EE-Buildumsetzer ausgeführt wird, sucht dieser die Hierarchie ausgehend von der Gruppenebene, die die Erstellung ausführt, und verwendet das erste \$GLOBAL-Member, das er im Typ J2EEBLD findet.

Anmerkung: Das \$GLOBAL-Member muss als gültiges SCLM-Member gespeichert werden, damit diese Hierarchiesuche ausgeführt werden kann.

Wenn verschiedene Einstellungen erforderlich sind, z. B. für verschiedene Entwicklungsgruppen, kann in jeder dieser Entwicklungsgruppen ein \$GLOBAL-Member erstellt werden.

TRANSLATE.conf

Die Datei TRANSLATE.conf stellt Schlüsselwörter bereit, um zu ermitteln, wie der Code in SCLM gespeichert ist. Die Konfigurationsdatei enthält Schlüsselwörter, die ermitteln, wie Dateien abhängig von ihrer Sprachdefinition auf den Host übertragen werden. Spezifische Schlüsselwörter bestimmen, ob Dateien mit einem bestimmten Sprachtyp binär sind, übertragen und gespeichert werden oder ob die textbasierte Quelle das ASCII-Format beibehält, statt standardmäßig von ASCII in EBCDIC konvertiert zu werden.

Darüber hinaus steuern die SCLM-Sprachdefinitionen, ob ausgeschriebene Dateinamen in passende gültige kurze Hostnamen konvertiert werden, die in SCLM gespeichert werden. Diese Zuordnung von Lang- zu Kurznamen wird durch die SCLM-VSAM für die Umsetzung von Lang- und Kurznamen gesteuert.

TRANSLATE.conf ist gespeichert in /etc/rdz/sclmdt/CONFIG. Sie können die Datei mit dem TSO-Befehl OEDIT bearbeiten.

Das folgende Beispiel zeigt den Code „TRANSLATE.conf“. Dieser muss angepasst werden, um Ihrer Systemumgebung zu entsprechen. Kommentarzeilen beginnen mit einem Stern (*).

```
*=====
* cross system sharing
TRANVRLS = NO
*=====
* codepage
CODEPAGE ASCII = IS08859-1
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
LOGLANG JAVA
LOGLANG SQLJ
LOGLANG J2EEPART
LOGLANG JAVABIN
LOGLANG J2EEBIN
LOGLANG J2EEOBJ
LOGLANG DOC
LOGLANG XLS
```

Abbildung 14. TRANSLATE.conf - SCLMDT ASCII/EBCDIC Umsetzungsconfigurationsdatei

Die folgenden Schlüsselwörter gelten innerhalb der Datei TRANSLATE.conf:

Codepage ASCII

Gibt die ASCII-Codepages an, die bei der Umsetzung verwendet werden sollen. Standardmäßig wird IS08859-1 verwendet.

Es muss eine Codepage-Anweisung für ASCII und EBCDIC für SCLM Developer Toolkit vorhanden sein, um zu ermitteln, wie übertragene Dateien konvertiert werden sollen.

Codepage EBCDIC

Gibt die EBCDIC-Codepages an, die bei der Umsetzung verwendet werden sollen. Standardmäßig wird IBM-1047 verwendet.

Es muss eine Codepage-Anweisung für ASCII und EBCDIC für SCLM Developer Toolkit vorhanden sein, um zu ermitteln, wie übertragene Dateien konvertiert werden sollen.

TRANVRLS

Gibt an, ob der VSAM-Datensatz für die Umsetzung von Lang- und Kurznamen auf mehreren Systemen gemeinsam genutzt werden kann. Der Standardwert ist NO. Gültige Werte sind YES und NO.

SCLM verwendet VSAM Record Level Sharing (RLS), um die gemeinsame Nutzung des VSAM-Datensatzes zu ermöglichen und die Integrität in einer gemeinsam genutzten Umgebung zu erhalten. Der VSAM-Datensatz muss mit der richtigen STORAGECLASS für die Verwendung von RLS definiert werden. Weitere Informationen zu RLS finden Sie im Dokument *DFSMS(TM) Using Data Sets (IBM Form SC26-7410-09)*.

TRANLANG

Gibt an, welche SCLM-Sprachtypen keine ASCII-/EBCDIC-Umsetzung erfordern (Dateien werden binär an den Host übertragen).

Beachten Sie, dass es kein Gleichheitszeichen (=) in dieser Anweisung gibt, um das TRANLANG-Schlüsselwort und den Namen des (Dummy-)Sprachumsetzers zu trennen.

LOGLANG

Ermittelt, für welche SCLM-Sprachtypen die Konvertierung von Langnamen in Kurznamen erforderlich ist. Die Konvertierung von ausgeschriebenen Namen in Kurznamen erfordert, dass die Datei mit dem ausgeschriebenen Namen auf dem Client (einschließlich Verzeichnispaketstruktur) zu einem gültigen Host-Membernamen mit 8 Zeichen zugeordnet wird und in SCLM mit diesem umgesetzten Host-Kurznamen gespeichert wird.

Wenn die SCLM-Sprache nicht im Schlüsselwort LOGLANG angegeben ist, wird vorausgesetzt, dass die Clientdatei bereits im Host-Kurznamenformat vorliegt (höchstens 8 Zeichen). Die Datei wird in dieser Form gespeichert.

Beachten Sie, dass es kein Gleichheitszeichen (=) in dieser Anweisung gibt, um das LOGLANG-Schlüsselwort und den Namen der SCLM-Sprache zu trennen.

Anmerkung: Es ist möglich, die in der Datei TRANSLATE.conf gesetzten Werte auf SITE- und SCLM-Projektebene zu überschreiben, wie unter „SITE- und projektspezifische Optionen“ beschrieben.

SITE- und projektspezifische Optionen

Es wurde eine Funktion bereitgestellt, um bestimmte Einstellungen auf SITE-Installationsebene oder auf einer bestimmten SCLM-Projektebene vornehmen zu können. Folgende Optionen können derzeit konfiguriert werden:

- Eintrag eines obligatorischen Änderungscode.
- Inaktivierung von Vordergrund-Builds und -Umstufungen.
- Bestimmung des Paketfreigabesystems. Derzeit wird das Freigabesystem IBM Breeze für SCLM unterstützt.
- Definition von Jobkarten für Batcherstellung, -umstufung und -migration.
- Einstellungen für Überschreibung in der Konfigurationsdatei TRANSLATE.conf.
- Projektlisten-Filtereinschränkung.
- Definition von Standardeinstellungen für bidirektional Sprachen (bidi).

Es können alle oder auch keine dieser Optionen festgelegt werden. Wenn diese Optionen nicht festgelegt werden, werden in den Programmen Standardwerte verwendet. Einige dieser Optionen können in der Datei SITE.conf festgelegt werden, andere auf projektspezifischer Ebene in SCLM. Alternativ kann auch auf eine SITE-spezifische Datei verzichtet werden. Die Optionen werden dann nur auf Projektebene in SCLM festgelegt. Jobkarteninformationen können überschrieben werden, indem Sie Ihre eigene angegebene Jobkarte verwenden, die in der IDE eingegeben wird.

Diese Funktion wird aktiviert, indem Sie die Datei SITE.conf im Verzeichnis z/OS UNIX /etc/rdz/sc1mdt/CONFIG/PROJECT/ erstellen (wobei /etc/rdz/sc1mdt das Standardverzeichnis etc für SCLM Developer Toolkit ist). Dieses Verzeichnis wird während der Anpassung von Developer for System z erstellt.

Eine Beispieldatei `SITE.conf` wird im Verzeichnis `/usr/lpp/rdz/samples/` bereitgestellt. Kopieren Sie dieses Verzeichnis und die Anweisungen Ihren Anforderungen entsprechend. Sie können die Datei mit dem TSO-Befehl `OEDIT` bearbeiten. Das folgende Beispiel zeigt die Konfigurationsdatei `SITE.conf`.

```

* 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
*
* 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
* Project list flag if set to N will stop users selecting * as project
* filter. This may avoid long catalog searches for all SCLM projects.
*
* 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
*
* Large temporary dataset allocations for users are set in the
* product. The maximum temporary dataset allocation is SPACE(15,75)
* tracks. To alter the maximum dataset allocation, uncomment and
* modify the primary and secondary extent values below to your own
* values. The SPACE values represent number of TRACKS.
*TEMPDSN=SPACE(15,75)

```

Abbildung 15. Beispiel für SITE-spezifische SCLM-Projekteinstellung

Es ist auch möglich, projektspezifische Konfigurationseinstellungen zu verwenden, die verwendet werden, um ein einzelnes SCLM-Projekt zu konfigurieren. Diese überschreiben die SITE-spezifischen Werte, wenn eine Datei des Typs SITE.conf vorhanden ist. Falls Sie projektspezifische Werte festlegen möchten, dann müssen Sie eine Datei mit dem Namen <project>.conf im Verzeichnis /PROJECT erstellen, wobei <project> der SCLM-Projektname ist (Groß- und Kleinschreibung spielt hierbei keine Rolle).

Eine Beispiel-Konfigurationsdatei wird im Verzeichnis /usr/lpp/rdz/samples/ als Datei SCLMproject.conf bereitgestellt. Kopieren Sie dieses Beispiel mit dem richtigen Zielnamen in das PROJECT-Verzeichnis und passen Sie die Anweisungen Ihren Anforderungen entsprechend an.

Sie können die Datei mit dem TSO-Befehl OEDIT bearbeiten. Das folgende Beispiel zeigt den Projektkonfigurationscode.

```
* 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
```

Abbildung 16. Beispiel für projektspezifische SCLM-Projekteinstellung

Alle aufgelisteten Optionen sind optional. Wenn in SITE.conf oder project.conf keine Angaben gemacht werden, werden Standardwerte verwendet.

Tabelle 6. SITE-/Projektoptionen

BUILDAPPROVER=approval product/NONE	Geben Sie den Namen des Freigabeprodukts an, das für den Erstellungsprozess verwendet wird. Derzeit wird nur IBM Breeze für SCLM unterstützt. Dieses Produkt wird mit dem Schlüsselwort BREEZE ausgewählt. Die Standardeinstellung ist NONE.
-------------------------------------	--

Tabelle 6. SITE-/Projektoptionen (Forts.)

PROMOTEAPPROVER=approval product/NONE	Geben Sie den Namen des Freigabeprodukts an, das für den Umstufungsprozess verwendet wird. Derzeit wird nur IBM Breeze für SCLM unterstützt. Wenn PROMOTEAPPROVER auf BREEZE gesetzt ist, werden bei einer Umstufung Breeze-spezifische Felder angezeigt. Die Standardeinstellung ist NONE.
CCODE=N/Y	Legen Sie Y fest, um den Änderungscode-Eintrag beim Check-In als Pflichtfeld zu definieren. Der Standardwert ist N, wobei der Änderungscode-Eintrag nicht obligatorisch ist.
FOREGROUNDBUILD=Y/N	Legen Sie N fest, um Vordergrund-Builds einzuschränken. Die Standardeinstellung ist Y, wobei Vordergrund-Builds zulässig sind.
FOREGROUNDPROMOTE=Y/N	Legen Sie N fest, um Vordergrund-Umstufungen einzuschränken. Die Standardeinstellung ist Y, wobei Vordergrund-Umstufungen zulässig sind.
BATCHBUILD1=Job card 1 BATCHBUILD2=Job Card 2 BATCHBUILD3=Job Card 3 BATCHBUILD4=Job Card 4	Legen Sie eine Standard-Batch-Jobkarte für den Erstellungsprozess fest. Verschiedene Projekte können verschiedene Account-Codes oder Jobklassen verwenden. Mit der Option für die Angabe von projektspezifischen Jobkarten ist dieses Szenario möglich.
BATCHPROMOTE1=Job card 1 BATCHPROMOTE2=Job card 2 BATCHPROMOTE3=Job card 3 BATCHPROMOTE4=Job card 4	Legen Sie einen Standard-Batch-Job für den Umstufungsprozess fest. Verschiedene Projekte können verschiedene Account-Codes oder Jobklassen verwenden. Mit der Option für die Angabe von projektspezifischen Jobkarten ist dieses Szenario möglich.
BATCHMIGRATE1=Job card 1 BATCHMIGRATE2=Job card 2 BATCHMIGRATE3=Job card 3 BATCHMIGRATE4=Job card 4	Legen Sie einen Standard-Batch-Job für den Migrationsprozess fest. Verschiedene Projekte können verschiedene Account-Codes oder Jobklassen verwenden. Mit der Option für die Angabe von projektspezifischen Jobkarten ist dieses Szenario möglich.
BUILDSECURITY=Y/N	Geben Sie Y an, um den SAF/RACF-Sicherheitsaufruf für den Erstellungsschritt aufzurufen und eventuell einen Wechsel zur Ersatz-ID auszuführen. Weitere Informationen finden Sie unter Kapitel 4, „SCLM-Sicherheit“, auf Seite 51.
PROMOTSECURITY=Y/N	Geben Sie Y an, um den SAF/RACF-Sicherheitsaufruf für den Umstufungsschritt aufzurufen und eventuell einen Wechsel zur Ersatz-ID auszuführen. Weitere Informationen finden Sie unter Kapitel 4, „SCLM-Sicherheit“, auf Seite 51.
DEPLOYSECURITY=Y/N	Geben Sie Y an, um den SAF/RACF-Sicherheitsaufruf für den Implementierungsschritt aufzurufen und eventuell einen Wechsel zur Ersatz-ID auszuführen. Weitere Informationen finden Sie unter Kapitel 4, „SCLM-Sicherheit“, auf Seite 51.
ASCII=ASCII codepage	Geben Sie die ASCII-Codepage an, mit der die in TRANSLATE.conf angegebene ASCII-Codepage überschrieben werden soll. Beispiel: ASCII=UTF-8
EBCDIC=EBCDIC codepage	Geben Sie die EBCDIC-Codepage an, mit der die in TRANSLATE.conf angegebene EBCDIC-Codepage überschrieben werden soll. Beispiel: EBCDIC=IBM-420
TRANLANG=SCLM Language	Geben Sie einen TRANLANG-Parameter an, der zur Liste der TRANLANG-Parameter hinzugefügt werden soll, die in TRANSLATE.conf angegeben sind. Beispiel: TRANLANG=DOC

Tabelle 6. SITE-/Projektoptionen (Forts.)

NOTRANLANG=SCLM Language	Verwenden Sie das Schlüsselwort NOTRANLANG, um bereits angegebene TRANLANG-Parameter aus der für dieses SCLM-Projekt zulässigen Liste zu entfernen, wie in TRANSLATE.conf angegeben. Beispiel: NOTRANLANG=JAVA
LOGLANG=SCLM Language	Geben Sie einen LOGLANG-Parameter an, der zur Liste der LOGLANG-Parameter hinzugefügt werden soll, die in TRANSLATE.conf angegeben sind. Beispiel: LOGLANG=DOC
NOLONGLANG=SCLM Language	Verwenden Sie das Schlüsselwort NOLONGLANG, um bereits angegebene LOGLANG-Parameter aus der für dieses Projekt zulässigen SCLM-Liste, wie in TRANSLATE.conf angegeben, zu entfernen. Beispiel: NOLONGLANG=COBOL
BIDIPROP=LANG=SCLM Language/* TextOrient=LTR/RTL TextType=Visual/Logical SymetricSwap=On/Off NumericSwap=On/Off	Verwenden Sie das Schlüsselwort BIDIPROP, um die Standardwerte für bidirektionale Sprachen auf SCLM-Sprachen zu setzen. LANG= kann entweder auf alle SCLM-Sprachen oder auf bestimmte SCLM-Sprachen gesetzt werden. Bidirektionale Unterstützung ist nur unter Developer for System z möglich.
PROJECTLISTALL=Y	Das Flag der Projektliste ist auf N gesetzt und verhindert, dass Benutzer * als Projektfilter auswählen. Dadurch werden zeitaufwändige Benutzerkatalogsuchen nach allen SCLM-Projekten vermieden.

Beispiel für die Verwendung von Kombinationen der TRANSLATE.conf-Überschreibungen

Die Datei TRANSLATE.conf legt die Standardeinstellungen für die Codepageunterstützung und Standard-SCLM-Sprachunterstützung fest, die in SCLM Developer Toolkit zugewiesen werden sollen. In diesem Beispiel hat TRANSLATE.conf die unten angegebenen Werte.

```

CODEPAGE ASCII = ISO8859-1
CODEPAGE EBCDIC = IBM-1047
*
TRANLANG JAVABIN
TRANLANG J2EEBIN
TRANLANG J2EEOBJ
TRANLANG TEXTBIN
TRANLANG BINARY
TRANLANG DOC
TRANLANG XLS
*
LOGLANG JAVA
LOGLANG SQLJ
LOGLANG DOC
LOGLANG XLS
LOGLANG J2EEPART
LOGLANG JAVABIN
LOGLANG J2EEBIN
LOGLANG J2EEOBJ

```

Verschiedene SCLM-Projekte, die verschiedene Datentypen speichern, möglicherweise in verschiedenen Landessprachen, können diese Standardeinstellungen überschreiben. Daher kann die Projektkonfigurationsdatei SCLMPRJ1.conf für das SCLM-Projekt SCLMPRJ1 folgende Überschreibungseinstellungen haben:

```

* Überschreibungen für arabische Codepages
*
ASCII=UTF-8
EBCDIC=IBM-420
*
* Projektspezifische TRANLANG- und LONGLANG-Einträge
*
TRANLANG=DOC
LONGLANG=DOC

```

Dadurch werden die Codepages für Quellenumsetzungen auf das arabische Codepage-Paar gesetzt. Darüber hinaus wird eine SCLM-Sprache des Typs DOC zu den Standardwerten aus TRANSLATE.conf hinzugefügt.

Das SCLM-Projekt SCLMPRJ2 kann andere Überschreibungseinstellungen in SCLMPRJ2.conf haben:

```

* Überschreibungen für hebräische Codepages
*
ASCII=UTF-8
EBCDIC=IBM-424
*
* Projektspezifische TRANLANG- und LONGLANG-Einträge
*
TRANLANG=DOC
TRANLANG=XLS
NOTRANLANG=JAVABIN
NOTRANLANG=J2EEBIN
NOTRANLANG=J2EEOBJ
LONGLANG=DOC
LONGLANG=XLS
NOLONGLANG=COBOL
NOLONGLANG=J2EEPART
NOLONGLANG=JAVABIN
NOLONGLANG=J2EEBIN
NOLONGLANG=J2EEOBJ

```

Dadurch werden die Codepages für Quellenumsetzungen auf das hebräische Codepage-Paar gesetzt. Darüber hinaus werden SCLM-Sprachen des Typs DOC und XLS zu den Standardwerten aus TRANSLATE.conf hinzugefügt. In diesem Fall werden jedoch die in TRANSLATE.conf festgelegten Standardwerte gelöscht. Dies ist nicht unbedingt erforderlich, da es nicht problematisch ist, zusätzliche Einstellungen zu verwenden. Auf diese Weise wird jedoch veranschaulicht, wie ein Projekt so eingerichtet werden kann, dass nur die erforderlichen SCLM-Sprachen für ein bestimmtes SCLM-Projekt vorhanden sind.

Beispiel für die Verwendung von Kombinationen der BIDIPROP-Überschreibungen

Die BIDIPROP-Werte, die in SITE.conf angegeben sind, können von beliebigen BIDIPROP-Werten überschrieben werden, die in den projektspezifischen Dateien <project>.conf angegeben sind. In SITE.conf wird beispielsweise Folgendes festgelegt:

```

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

```

Dadurch werden alle SCLM-Sprachen auf die angegebenen Einstellungen gesetzt. Nun kann Folgendes in der Datei ADMIN10.conf festgelegt werden:

```
*  
* 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
```

Diese Einstellungen überschreiben die Einstellungen in SITE.conf für die JAVA- und COBOL-Sprachdefinitionen. Alle anderen Sprachen erhalten die Standardeinstellungen, wie in SITE.conf angegeben.

Kapitel 3. SQLJ-Unterstützung

SQLJ ist eine Spracherweiterung für Java. Es handelt sich dabei um eine Technologie, mit der Java-Programmierer Datenbankkommunikation in Programmen integrieren können. SQLJ bietet eine Möglichkeit, statisches, eingebettetes SQL zu erzeugen, das im Allgemeinen eine bessere Leistung bietet als dynamische Äquivalente wie JDBC.

SCLM Developer Toolkit wird mit Beispielskripts geliefert, die Ihnen ermöglichen, SQLJ-fähige Java-Programme unter Verwendung von DB2 zu erstellen.

In diesem Kapitel werden die Grundlagen von SQLJ erläutert und es wird beschrieben, wie diese bei der Verwendung von SCLM Developer Toolkit angewendet werden.

Was ist SQL?

SQL ist ein Akronym für *Structured Query Language*. Dabei handelt es sich um eine offene Sprache, die für das Abfragen, Hinzufügen, Löschen und Ändern von Daten in einem Managementsystem für relationale Datenbanken (RDMS) verwendet wird.

Diese Sprache wurde erstmals in den 1970er Jahren in einem frühen IBM Datenbankprodukt implementiert: System R. Seither wurde SQL weiterentwickelt, standardisiert (durch ANSI und ISO) und wird in vielen Varianten bei unterschiedlichen Datenbanksystemen verwendet.

Was ist DB2?

DB2 ist ein vielfach eingesetztes Datenbanksystem, das ursprünglich für die Mainframeplattform entwickelt und seitdem für viele andere Plattformen erweitert wurde. Es ist das Standardmanagementsystem für relationale Datenbanken unter z/OS.

DB2 UDB Version 8 ist die Version, auf der die Erstellungsskripts von SCLM Developer Toolkit basieren. Verweise auf DB2 in diesem Kapitel beziehen sich speziell auf DB2 UDB Version 8.

Was ist JDBC?

JDBC steht für *Java Database Connectivity*. Auf dem Gebiet der Java-Entwicklung ist dies eine bekannte und weit verbreitete Technologie für die Implementierung von Datenbankinteraktionen. JDBC ist eine API auf Aufrufebene, d. h. SQL-Anweisungen werden als Zeichenfolgen an die API übergeben, die diese dann auf dem RDMS ausführt. Daher kann der Wert dieser Zeichenfolgen während der Laufzeit geändert werden, wodurch JDBC dynamisch wird.

JDBC-Programme werden langsamer ausgeführt als funktional entsprechende SQLJ-Programme. Ein Vorteil dieser Lösung ist jedoch das als „Write once, call anywhere“ (Einmal schreiben, überall aufrufen) bekannte Konzept. Da keine Interaktion bis zur Laufzeit erforderlich ist, können JDBC-Programme zwischen verschiedenen Systemen mit minimalem Aufwand übertragen werden.

Was ist SQLJ?

SQLJ ist eine Spracherweiterung, die für Datenbanktransaktionen in Java-Anwendungen verwendet wird. Dabei wird statischer, integrierter SQLJ-Code erzeugt. Dieser Begriff setzt sich zusammen aus „SQL“, was für *Structured Query Language* steht, und „J“, was für *Java* steht.

SQLJ ist *statisch*, da die SQL-Anweisungen, die während der Laufzeit ausgeführt werden, bekannt sind, wenn das Programm assembliert wird. Hierin besteht ein Unterschied zu JDBC. In JDBC können die ausgeführten Abfragen jederzeit geändert werden.

SQLJ ist *integriert*, denn während des Bindens wird ein serialisiertes Formular der SQL-Anweisungen des Programms an die Datenbank übergeben. Die Datenbank verwendet diese serialisierten Daten, um optimierte Zugriffspfade zu den referenzierten Tabellen zu ermitteln. In JDBC kann die Datenbank erst ermitteln, welche Anweisungen ausgeführt werden sollen, wenn sie diese während der Laufzeit von der Anwendung erhält. Daher muss die Datenbank die Zugriffspfade während der Laufzeit ermitteln. Dadurch entsteht ein Systemaufwand, der bei der Verwendung von SQLJ vermieden wird.

Vergleich von JDBC und SQLJ

Diese Tabelle basiert auf Materialien, die im Kapitel 5.2 des Redbooks *DB2 UDB for z/OS Version 8: Everything You Ever Wanted to Know, ... and More* enthalten sind.

Tabelle 7. Vergleich von JDBC und SQLJ

	SQLJ (statisch)	JDBC (dynamisch)
PERFORMANCE	Meistens ist statisches SQL schneller als dynamisches SQL, da während der Laufzeit nur die Berechtigung für Pakete und Pläne vor der Ausführung des Programms überprüft werden muss.	Bei dynamischen SQL-Anweisungen muss eine Syntaxanalyse der SQL-Anweisungen ausgeführt werden, die Berechtigung für die Tabellen/Anzeige muss geprüft werden und der Optimierungspfad muss ermittelt werden.
AUTORISIERUNG	In SQLJ gewährt der Anwendungsinhaber die EXECUTE-Berechtigung für den Plan oder das Paket und der Empfänger dieses GRANT muss die Anwendung wie geschrieben ausführen.	In JDBC gewährt der Anwendungsinhaber Berechtigungen für alle zugrunde liegenden Tabellen, die von der Anwendung verwendet werden. Der Empfänger dieser Berechtigungen kann alle Vorgänge ausführen, die diese Berechtigungen zulassen, z. B. diese außerhalb der Anwendung verwenden, für die die Berechtigungen ursprünglich gewährt wurden. Die Anwendung kann nicht steuern, welche Vorgänge der Benutzer ausführen kann.

Tabelle 7. Vergleich von JDBC und SQLJ (Forts.)

	SQLJ (statisch)	JDBC (dynamisch)
DEBUGGING	SQLJ ist keine Anwendungsprogrammierschnittstelle, sondern eine Spracherweiterung. Dies bedeutet, dass die SQLJ-Tools die SQL-Anweisungen in Ihrem Programm kennen und diese während des Programmentwicklungsprozesses auf korrekte Syntax und Berechtigungen prüfen.	JDBC ist eine reine Anwendungsprogrammierschnittstelle auf Aufrufebene. Dies bedeutet, dass der Java-Compiler keinerlei Informationen zu den SQL-Anweisungen hat. Diese erscheinen nur als Argumente für Methodenaufrufe. Wenn eine Ihrer Anweisungen fehlerhaft ist, kann dieser Fehler erst während der Laufzeit abgefangen werden, wenn die Datenbank den Fehler meldet.
MONITORING	Mit SQLJ erhalten Sie eine wesentlich bessere Systemüberwachung und Leistungsberichterstattung. Statische SQL-Pakete liefern Ihnen die Namen der Programme, die zu einem beliebigen Zeitpunkt ausgeführt werden. Dies ist überaus nützlich für die Überprüfung der CPU-Belegung durch die verschiedenen Anwendungen, bei Sperrproblemen (wie Deadlock oder Zeitlimitüberschreitung) usw.	Während Sie mit SQLJ den Namen des aktuell ausgeführten Programms ermitteln können, werden mit JDBC alle Vorgänge durch dasselbe Programm ausgeführt. Dadurch wird die Überwachung und Lokalisierung von Problembereichen erschwert.
VERBOSITY	Da SQLJ-Anweisungen in reiner SQL-Syntax codiert werden, ohne dass diese in eine Java-Methode eingeschlossen werden müssen, sind die Programme leichter zu lesen, wodurch sie einfacher verwaltet werden können. Da außerdem ein Teil des Standardcodes, der explizit in JDBC codiert werden muss, automatisch in SQLJ generiert wird, sind in SQLJ geschriebene Programme meistens kürzer als funktional entsprechende JDBC-Programme.	In JDBC müssen alle SQL-Anweisungen in API-Aufrufe eingeschlossen werden, wodurch im Allgemeinen ein undeutlicher und ausführlicher Code entsteht.

Was ist ein serialisiertes Profil?

Ein serialisiertes Profil ist ein Code, der in SQLJ geschrieben wurde und mit der Erweiterung `.sqlj` in eine Datei eingefügt wird. Im ersten Schritt der Programmierung (später ausführlicher erläutert) wird die Datei `.sqlj` dem SQLJ-Umsetzer zugeführt.

Der Umsetzer erstellt zwei Typen von Ausgaben. Der erste Typ ist Java-Quellcode (`.java`). Dieser Quellcode ist die Java-Implementierung des Codes innerhalb der SQLJ-Datei.

Der zweite Ausgabetypp ist ein serialisiertes Profil (`.ser`). Diese Datei enthält alle SQL-Anweisungen aus der `.sqlj`-Datei in serialisierter Form. Dieses Profil muss dem Programm während der Laufzeit zur Verfügung stehen und kann auch verwendet werden, um eine Bindung mit dem RDMS herzustellen.

Was ist ein Datenbankankforderungsmodul (DBRM)?

DBRM steht für *Database Request Module* (Datenbankanforderungsmodul). Dabei handelt es sich um die konventionelle serialisierte DB2-Darstellung der SQL-Anweisungen in einem Programm. Ein Programm kann z. B. in COBOL geschrieben sein. Dieses Programm wird durch DB2 vorverarbeitet. Dabei wird ein Datenbankankforderungsmodul erstellt, das für die Bindung mit einem bestimmten DB2-Subsystem verwendet wird.

Dieser Prozess unterscheidet sich in Bezug auf SQLJ geringfügig. Im Zusammenhang mit DB2 UDB Version 8 wird dieser Vorgang als *Kompatibilitätsmodus* bezeichnet. Das Dienstprogramm *db2sqljcustomize* kann mit optionalen Befehlszeilenargumenten bereitgestellt werden, die die Erstellung eines Datenbankankforderungsmoduls veranlassen. Dieses Datenbankankforderungsmodul kann anschließend durch Verwendung konventioneller Vorgehensweisen, z. B. durch ein REXX-Script, das durch einen SCLM-Benutzerexit aufgerufen wird, an DB2 gebunden werden.

SQLJ-Programmerstellung

Bevor erläutert wird, wie SCLM Developer Toolkit zum Erstellen von SQLJ-Programmen verwendet werden kann, wird an dieser Stelle zunächst der manuelle Prozess beschrieben. Dieser Prozess bezieht sich auf die DB2-Implementierung von SQLJ. Dabei werden drei Befehle mit den Bezeichnungen *sqlj*, *db2sqljcustomize* und *db2bind* verwendet. Beachten Sie, dass der Bindungsschritt optional in *db2sqljcustomize* ausgeführt werden kann, daher wird *db2bind* nicht in allen Fällen benötigt.

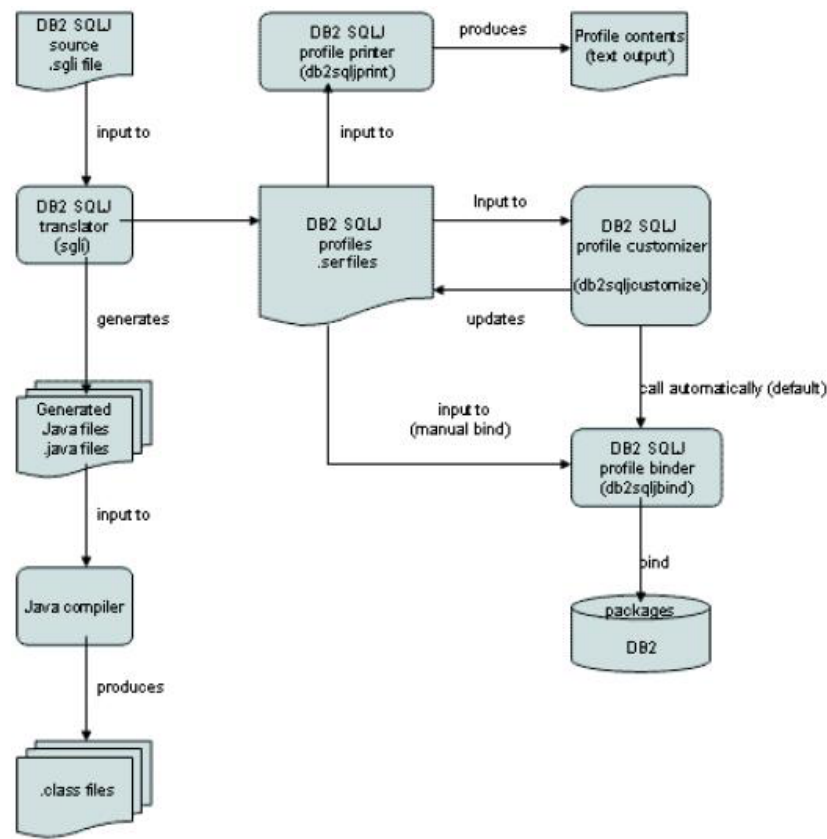


Abbildung 17. SQLJ-Programmerstellung

Umsetzung

Der SQLJ-Umsetzer (nicht zu verwechseln mit einem *SCLM-Sprachumsetzer*) verwendet SQLJ-Quellendateien als Eingabe und erstellt Java-Quellcode (Dateien des Typs .java) und serialisierte Profile (Dateien des Typs .ser).

Die SQLJ-Sprache selbst wird in diesem Handbuch nicht erläutert. Nutzen Sie die Website <http://www.sql.org>, um Hinweise zur Entwicklung von SQLJ-Code zu erhalten.

Die Anzahl der pro .sqlj-Datei generierten serialisierten Profile ist abhängig von der Anzahl der *Verbindungskontext*-Klassen, die im SQLJ-Code referenziert werden. Ein serialisiertes Profil wird für jede Klasse generiert.

Viele SQLJ-Quellendateien verweisen nur auf eine bestimmte Verbindungskontextklasse und generieren daher nur ein serialisiertes Profil. Die serialisierten Profile werden anhand der Reihenfolge benannt, in der sie in der Quellendatei angegeben werden. Für den Namen wird das folgende Format verwendet:

progname_SJProfileX.ser

Dabei gilt:

- *Programmname* repräsentiert den Namen des Programms. Dieser wird durch Löschen der Erweiterung .sqlj aus dem Eingabequellen-Dateinamen ermittelt.

- *X* repräsentiert eine ganze Zahl, die den Index der laufenden Klasse angibt. Basis für die Indexierung ist null. Die erste referenzierte Verbindungskontextklasse erzeugt das Profil 0, die zweite das Profil 1 usw.

Beispiel:

Eingabe: Customer.sqlj (referenziert eine Verbindungskontextklasse)

Ausgabe: Customer.java
Customer_SJProfile0.ser

Und optional, wenn das Argument `-compile=true` für sqlj bereitgestellt wird:
Customer.class

Anpassung

Nach dem Generieren der serialisierten Profile müssen diese angepasst werden. Der hierfür in DB2 Version 8 verwendete Befehl ist *db2sqljcustomize*. In früheren Versionen wurde der Befehl *db2prof* verwendet. Jeder Aufruf des Customizers sollte einem Aufruf mit dem SQLJ-Umsetzer entsprechen. Das heißt, wenn ein Aufruf des Umsetzers fünf Profile generiert hat, müssen diese fünf Profile einem Aufruf der Profilanpassungsfunktion als Eingabe zugeführt werden. Anders ausgedrückt bedeutet dies, dass jeder einzelne Programmname mit einem Aufruf für jedes einzelne Dienstprogramm verknüpft wird. Beachten Sie, dass der Programmname dem Eingabequellen-Dateinamen ohne die Erweiterung `.sqlj` entspricht.

Bei der Anpassung werden dem serialisierten Profil DB2-spezifische Informationen hinzugefügt, die während der Laufzeit verwendet werden. Andere Optionen, wie automatisches Binden, können durch Befehlszeilenswitches konfiguriert werden. Wenn Sie eine ältere Version von DB2 verwenden oder wenn Sie die Attribute *gendbrm* und *dbrmdir* für *db2sqljcustomize* festlegen, wird eine DBRM-Datei generiert. Diese Datei wird später für die Bindung an die Datenbank verwendet. Mit dem universellen Treiber von DB2 UDB 8+ können Sie auf die Erstellung eines Datenbankankforderungsmoduls verzichten und die Bindung stattdessen mit den serialisierten Profilen herstellen, die mit dem SQLJ-Umsetzer generiert wurden.

Bindung

Binden ist der letzte Schritt im Prozess der SQLJ-Programmerstellung. In DB2 Version 8 wird für das Binden der Befehl *db2sqljbind* verwendet. Sie können auch automatisches Binden festlegen, indem Sie den Befehl *db2sqljcustomize* ausführen. Beim Binden wird ein Zugriffspfad zu den DB2-Tabellen für Ihre serialisierten SQL-Anweisungen erstellt. Diese Anweisungen sind entweder in Form eines Datenbankankforderungsmoduls oder eines serialisierten Profils verfügbar.

Standardmäßig werden vier Pakete erstellt, eins für jede Isolationsstufe. Sie können entweder mit der konventionellen Methode binden, bei der ein Datenbankankforderungsmodul verwendet wird, oder mit der neuen universellen Methode, bei der stattdessen serialisierte Profile verwendet werden.

SCLM DT-Typen und -Umsetzer

Bevor die SCLM-Typen und -Umsetzer erläutert werden, sollen die Unterschiede zwischen einem *SCLM-Sprachumsetzer* (kurz *SCLM-Umsetzer*) und dem SQLJ-Umsetzer *sqlj* beschrieben werden, der in DB2 bereitgestellt wird.

In SCLM muss jede definierte Sprache einen Umsetzer haben, damit bekannt ist, wie diese Sprache verarbeitet werden kann. Diese Umsetzer unterscheiden sich von

dem SQLJ-Umsetzer *sqlj*: Bei letzterem handelt es sich um ein Befehlszeilendienstprogramm, das eine SQLJ-Quellendatei aufruft und serialisierte Profile und Java-Quellcode erstellt.

Nachdem diese Unterscheidung verdeutlicht wurde, sollen die *SCLM-Typen* und *SCLM-Umsetzer* erläutert werden, die mit dem SQLJ-Erstellungsprozess verbunden sind.

Ein SCLM-Umsetzer für SQLJ wird bereitgestellt und sollte als Sprachtyp für den gesamten SQLJ-Quellcode zugewiesen werden, der in SCLM gespeichert wird. Für diesen neuen Umsetzer müssen zusätzliche SCLM-Typen definiert werden. Der SCLM-Umsetzer für SQLJ ist ähnlich aufgebaut wie der JAVA-Umsetzer, enthält jedoch weitere IOTYPE-Definitionen für die SCLM-Ausgabetypen SQLJSER und DBRMLIB. Wenn Sie keine DBRM-Dateien während des Anpassungsvorgangs generieren möchten, kann dieser DBRMLIB IOTYPE aus der SQLJ-Sprachdefinition entfernt werden.

Innerhalb der Projektdefinition muss der Administrator den neuen SCLM-Umsetzer und zusätzliche Typen definieren.

Tabelle 8. SCLM-Umsetzertypen für SQLJ

SQLJSER	Dies ist erforderlich, um die generierten, serialisierten Profildateien (.ser-Dateien) zu speichern, die in den Umsetzungs- und Anpassungsschritten erstellt oder angepasst wurden. Es wird empfohlen, diesen Datensatz vom Typ SCLM als recfm=VB, lrecl= 256 zu definieren.
DBRMLIB	Ein Typ, der die generierten DBRM-Dateien enthalten muss, die im Anpassungsschritt erstellt werden. Dieser Typ wird nur für Kunden benötigt, die generierte DBRM-Dateien im Rahmen des DB2-Bindeprozesses verwenden.

Anpassung des Erstellungsprozesses

Um hohe Flexibilität zu gewährleisten, kann der SQLJ-Erstellungsprozess angepasst werden. Auf diese Weise können unterschiedliche Standortkonfigurationen und eine beliebige Kombination von Parametern berücksichtigt werden, die an *sqlj* und *db2sqljcustomize* übergeben werden müssen.

In diesem Abschnitt werden die Konzepte der Implementierung von SQLJ mit SCLM Developer Toolkit beschrieben. Hier erfahren Sie, wie Sie den Erstellungsprozess anpassen können, um die Anforderungen Ihres Standorts zu berücksichtigen.

Während der SQLJ-Umsetzung und Profilanpassung ruft SCLM Developer Toolkit dieselben Java-Klassen direkt auf, die von den Befehlen *sqlj* und *db2sqljcustomize* verwendet werden. Beachten Sie, dass die Argumente, die an den SCLM DT-Umsetzungs- und Anpassungsprozess übergeben werden, exakt übereinstimmen. Ausführliche Informationen zu allen Befehlszeilenargumenten für jeden Befehl finden Sie im Handbuch *DB2 Universal Database Benutzerhandbuch*.

Anpassung des Erstellungsscripts

Angenommen, Sie haben den Assistenten 'Zu SCLM hinzufügen' verwendet, dann erhält das Erstellungsscript für Ihr SQLJ-Programm denselben Membernamen wie die ARCHDEF. Falls die ARCHDEF für Ihr SQLJ-Projekt SCLM10.DEV1.ARCHDEF(SQLJ01) ist, dann finden Sie das Erstellungsscript unter SCLM10.DEV1.J2EEBLD(SQLJ01).

In diesem Erstellungsscript ist ein Verweis auf das Master-Erstellungsscript vorhanden. Dieser befindet sich in der Eigenschaft. Der größte Teil der Konfiguration, die für die Umsetzungs- und Anpassungsschritte aufgeführt ist, wird in dieser Datei ausgeführt.

Anmerkung: Das in Developer Toolkit enthaltene Erstellungsscript für JAR-Projekte ist BWBSQLB, für EJB-Projekte wird BWBSQLBE verwendet. Üblicherweise muss dieser Wert nicht geändert werden.

sqlj.*-Eigenschaften

Jede in Tabelle 9 aufgeführte Eigenschaft wird im BWBSQLB-Erstellungsscript angezeigt. Die Eigenschaften liegen im XML-Format vor und lauten wie folgt:

Die Konfiguration des Scripts umfasst die Änderung des Werts für alle relevanten Eigenschaften.

Tabelle 9. sqlj.-Eigenschaften*

Name	Wert	Beschreibung
sqlj.exec	usr/lpp/rdz/bin/bwbsqlc.rex	Gibt die Position von SQLJ (Structured Query Language for Java) und der db2sqljcustomize-Exec-Routine bwbsqlc.rex an, die sich im Installationsverzeichnis Developer for System z befindet.
sqlj.class	sqlj.tools.Sqlj	Geben Sie den SQLJ-Klassennamen an. Dies ist der Name der Klasse, die durch das SQLJ-Dienstprogramm aufgerufen wird. Üblicherweise muss dieser Wert nicht geändert werden.
sqlj.bin	/db2path/bin	Geben Sie die Speicherposition des DB"-SQLJ-Verzeichnisses „bin“ an, wo das SQLJ-Script gespeichert ist.
sqlj.cp	/db2path/jcc/classes/sqlj.zip	Geben Sie die Speicherposition von sqlj.zip für das Einschließen des Klassenpfads an.
sqlj.arg	-compile=false status linemap=NO db2optimize	Geben Sie unten globale Eigenschaftsargumente für die SQLJ-Verarbeitung an.

db2sqljcustomize.*-Eigenschaften

Jede in Tabelle 10 aufgeführte Eigenschaft wird im BWBSQLB-Erstellungsscript angezeigt. Die Eigenschaften im XML-Format sind folgende:

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

Die Konfiguration des Scripts umfasst die Änderung des Werts für alle relevanten Eigenschaften.

Tabelle 10. db2sqljcustomize.-Eigenschaften*

Name	Wert	Beschreibung
sqljdb2cust.class	com.ibm.db2.jcc.sqlj.Customizer	Geben Sie den Anpassungsklassennamen für sqlj db2 an. Üblicherweise muss dieser Wert nicht geändert werden.

Tabelle 10. *db2sqljcustomize.*-Eigenschaften (Forts.)*

Name	Wert	Beschreibung
db2sqljcust.cp	/db2path/jcc/classes/db2jcc.jar: ./SRC: /db2path/jcc/classes/db2jcc_license_cisuz.jar	Klassenpfadeinstellungen für das Anpassungsdienstprogramm. In XML müssen vollständig qualifizierte Namen bereitgestellt werden.
db2sqljcust.arg	-automaticbind NO -onlinecheck YES -staticpositioned YES -bindoptions "ISOLATION(CS)" genDBRM	Allgemeine Argumente für das Anpassungsdienstprogramm.
db2sqljcust.propfile	user.properties	Temporärer Eigenschaftendateiname, der an ein Benutzereigenschafts-Ermittlungsscript für dynamische Eigenschaftswerte übergeben wird. Behalten Sie die Standardeinstellung bei.
db2sqljcust.userpgm	NONE, wenn Sie das Script umgehen möchten. Andernfalls müssen Sie den vollständig qualifizierten Pfad- und Dateinamen des Benutzerscripts angeben.	Dieses Script wird direkt vor dem Anpassungsdienstprogramm ausgeführt. Dadurch wird eine Eigenschaftendatei dynamisch aktualisiert, die als Eingabe für das Anpassungsdienstprogramm verwendet wird.

Angepasstes Benutzerscript

Das SQLJ-Erstellungsscript, das in SCLM Developer Toolkit enthalten ist, wurde entwickelt, um im DB2 UDB v8-Kompatibilitätsmodus zu arbeiten. Dieser Modus unterstützt das DB2-Konzept der Datenbankanforderungsmodule anstelle des Bindens durch serialisierte Profile. Um die serialisierten Profile verwenden zu können, müssen Änderungen in BWBSQLB vorgenommen werden. Dies wird im Unterabschnitt „Bindung [Serialisiertes Profil]“ auf Seite 48 erläutert.

Abgesehen von der Bindungsmethode müssen Sie bei der Anpassung des Erstellungsprozesses für Ihren Standort wahrscheinlich die Argumente für *sqlj* und *db2sqljcustomize* anpassen, um Ihre Datenbankumgebung, Isolationsrichtlinie und andere Faktoren zu berücksichtigen. Vielleicht möchten Sie auch Ihre eigenen Scripts verwenden, um dynamische Eigenschaften für diese Argumente festzulegen, um z. B. einen Paketnamen, der mit dem Eingabedateinamen verbunden ist, auf effiziente Weise zu erstellen.

In SCLM Developer Toolkit können Sie Ihre eigenen Anpassungsscripts angeben. Alle Angaben im ANT-XML-Erstellungsprozess verwenden das Konzept der „Eigenschaften“. Dabei handelt es sich um XML-*Property*-Elemente, die ein Name-Wert-Paar angeben. So werden z. B. im *db2sqljcustomize*-Schritt im Erstellungsscript BWBSQLB die globalen Befehlszeilenargumente, die an *db2sqljcustomize* weitergegeben werden sollen, in einem Eigenschaftselement mit dem Namen *db2sqljcust.arg* und dem Standardwert *-automaticbind NO -onlinecheck YES -staticpositioned YES -bindoptions "ISOLATION(CS)" genDBRM lang=EN-AU* definiert.

Wenn Sie die angegebenen Argumente ändern möchten, können Sie das Erstellungsript bearbeiten, um den Wert der Eigenschaft zu ändern. Dabei werden die Eigenschaften global geändert. Andernfalls können Sie Ihr eigenes Anpassungsript in den Prozess einbinden.

Um Ihr eigenes angepasstes Eigenschaftssript einzubinden, geben Sie den Namen Ihres Scripts in `db2sqljcust.userpgm` sowie den Namen der Eigenschaftendatei, in die geschrieben werden soll, in `db2sqljcust.propfile` an.

Das in `db2sqljcust.userpgm` angegebene Script wird direkt vor dem `db2sqljcustomize`-Prozess ausgeführt. Ihr Script aktualisiert dynamisch die in `db2sqljcust.userpgm` angegebene Eigenschaftendatei. Diese Eigenschaftendatei wird als Eingabe für den `db2sqljcustomize`-Prozess verwendet, da der Erstellungsprozess beide Eigenschaften in der dynamisch aktualisierten Eigenschaftendatei sowie die bereits im Erstellungsript definierten Eigenschaften verkettet.

Das in `db2sqljcust.userpgm` angegebene Script wird durch die folgenden Argumente bereitgestellt, wenn es ausgeführt wird:

Argument	Beschreibung
Basedir	Basisverzeichnis (Arbeitsbereichsverzeichnis)
Propfile	Der Name der Eigenschaftendatei, die erstellt und aktualisiert wird Anmerkung: Die erstellte Eigenschaftendatei muss <code>basedir'/'propfile</code> sein.
Sqljf	Eine Liste von Dateinamen, die die serialisierten Profile (.ser) repräsentieren, die durch „db2sqljcustomize“ verarbeitet werden.

Die Eigenschaften sollten in der Datei im folgenden Format festgelegt werden, wobei eine Eigenschaftsdeklaration pro Zeile angegeben wird:

```
argument=value
Beispiel:
singlepkgname= pkgname
```

Beispiel:

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

Die angepasste Routine wird einmal pro Datei aufgerufen. Schließlich werden die Argumenteigenschaften verwendet, um die erforderliche Argumentenfolge für den Aufruf von „db2sqljcustomize“ zu erstellen. Beispiel:

```
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}
```

Bindung [DBRM]

DB2 verwendet traditionell ein *Datenbankanforderungsmodul* (DBRM) für diesen Zweck. Das Datenbankanforderungsmodul wird durch den Befehl `db2sqljcustomize` erstellt, wenn das Attribut `gendbrm` bereitgestellt wird. Ohne dieses Flag, geht der Befehl davon aus, dass Sie eine Bindung über serialisierte Profile ausführen möchten und generiert kein DBRM (Datenbankanforderungsmodul).

Wenn Sie diesen Parameter angeben, berücksichtigt SCLM Developer Toolkit die generierten Datenbankanforderungsmodule und speichert diese in SCLM für die zukünftige Verwendung. Ein Vorteil dieses Verfahrens ist, dass Sie eine DB2-Bindung einfach in einem SCLM-Benutzerexit ausführen können, z. B. dem Build-/Copy-Exit.

Da der Benutzerexit für Build und Kopie automatisch mit einer Liste aktualisierter Objekte zur Verfügung gestellt wird, können Sie nur die Module selektiv erneut binden, die sich geändert haben. Dadurch wird Ineffizienz durch redundante Bindungen vermieden.

Um die Bindung von Datenbankanforderungsmodulen zu konfigurieren, müssen die folgenden vier Schritte ausgeführt werden:

1. Legen Sie die entsprechenden Argumente für *sqlj* wie folgt fest:

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

Argument	Beschreibung
compile=false	Wenn diese Option auf „false“ gesetzt wird, wird vermieden, dass der SQLJ-Umsetzer automatisch den zu erstellenden Java-Quellcode kompiliert. SCLM Developer Toolkit verwendet die generierte Quelle in seinem eigenen Erstellungsprozess. Daher wird empfohlen, die Option immer auf „false“ zu setzen.
linemap=no	Gibt an, ob die Zeilennummern in Java-Ausnahmebedingungen den Zeilennummern in den SQLJ-Quellendateien (.sqlj-Dateien) entsprechen oder den Zeilennummern in der Java-Quellendatei, die durch die SQLJ-Umsetzerdateien generiert wird (.java-Datei). Dafür ist eine Datei des Typs .class erforderlich. Deshalb muss <i>no</i> festgelegt werden, wenn das Argument in Verbindung mit <i>compile=false</i> verwendet wird.
status	Gibt eine sofortige Statusanzeige der SQLJ-Verarbeitung aus.

2. Legen Sie die entsprechenden Argumente für *db2sqljcustomize*, einschließlich *gendbrm*, wie folgt fest:

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

Argument	Beschreibung
automaticbind no	Wenn auf „no“ gesetzt, führt der Customizer keine Bindung aus, wenn die Anpassung abgeschlossen ist.
onlinecheck yes	Führen Sie eine Onlineprüfung des mit dem Parameter <i>url</i> angegebenen Systems aus. Der Standardwert lautet 'Ja', wenn <i>url</i> angegeben ist. Andernfalls lautet der Standardwert 'Nein'.
Bindoptions ISOLATION(CS)	Weist den Binder an, ein einzelnes Paket zu erstellen (Cursorstabilität). Wird in Verbindung mit <i>singlepkgname</i> verwendet (dynamisch festgelegt).
gendbrm	Weist den Customizer an, DBRM-Dateien zu generieren.

3. Konfigurieren Sie das Benutzerscript.

Legen Sie die Speicherposition Ihres Benutzerprogramms in BWBSQLB fest. Dadurch erhält der Erstellungsprozess die Information, wo sich das Routenerweiterungsscript befindet, das zum Berechnen der dynamischen Eigenschaften verwendet wird.

Die entscheidende Eigenschaft, die dynamisch konfiguriert werden soll, ist *singlepkgname*. Dies ist der Name des Pakets, mit dem eine Bindung erstellt wer-

den soll. Jedes Programm erhält einen eindeutigen Paketnamen. In diesem einfachen Beispiel sind dies die ersten acht Buchstaben des Programmnamens.

4. Schreiben Sie einen Erstellungs-Exit, um das Datenbankanforderungsmodul zu binden. Es wird empfohlen, den Build-/Copy-Exit zu verwenden.

Da im Anpassungsschritt *singlepkgname* verwendet wird, stimmt der Name des Pakets mit dem Namen des Datenbankanforderungsmoduls überein.

Bindung [Serialisiertes Profil]

Als neue Lösung für das Binden von SQLJ-Programmen wird die Verwendung von serialisierten Profilen (Dateien des Typs *.ser*) empfohlen. Diese neue Lösung musste eingeführt werden, da das serialisierte Profil dieselbe Funktion ausführt wie das Datenbankanforderungsmodul und ein serialisiertes Bild der Anweisungen im Programm bereitstellt.

Mit einigen kleinen Änderungen am Erstellungsscript BWBSQLB können Sie SCLM Developer Toolkit für die Verwendung der neuen Methode konfigurieren. Dabei müssen nur die für „db2sqljcustomize“ bereitgestellten Argumente geändert werden, indem der Befehlszeilenwechsel „gendbrm“ gelöscht und „automaticbind“ auf „YES“ gesetzt wird.

Um die Bindung für serialisierte Profile zu konfigurieren, müssen die folgenden drei Schritte ausgeführt werden:

1. Legen Sie die entsprechenden Argumente für *sqlj* wie folgt fest:

Es gibt keine Befehlszeilenparameter für den sqlj-Umsetzer, die nur für die Bindung mit serialisierten Profilen gelten. Es werden jedoch die für dieses bestimmte Beispiel festgelegten Argumente gezeigt.

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

Argument	Beschreibung
compile=false	Wenn diese Option auf „false“ gesetzt wird, wird vermieden, dass der SQLJ-Umsetzer automatisch den zu erstellenden Java-Quellcode kompiliert. SCLM Developer Toolkit verwendet die generierte Quelle in seinem eigenen Erstellungsprozess. Daher wird empfohlen, die Option immer auf „false“ zu setzen.
linemap=no	Gibt an, ob die Zeilennummern in Java-Ausnahmebedingungen den Zeilennummern in den SQLJ-Quelldateien (.sqlj-Dateien) entsprechen oder den Zeilennummern in der Java-Quelldatei, die durch die SQLJ-Umsetzerdateien generiert wird (.java-Datei). Dafür ist eine Datei des Typs <i>.class</i> erforderlich. Deshalb muss <i>no</i> festgelegt werden, wenn das Argument in Verbindung mit <i>compile=false</i> verwendet wird.
status	Gibt eine sofortige Statusanzeige der SQLJ-Verarbeitung aus.

2. Legen Sie die entsprechenden Argumente für *db2sqljcustomize* wie folgt fest:

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

Argument	Beschreibung
automaticbind yes	Wenn auf „yes“ gesetzt, führt der Customizer auch dann eine Bindung aus, wenn die Anpassung abgeschlossen ist. Wenn auf „no“ gesetzt, muss die Bindung gesondert mit dem Befehl <i>db2bind</i> ausgeführt werden.
onlinecheck yes	Führen Sie eine Onlineprüfung des mit dem Parameter <i>url</i> angegebenen Systems aus. Der Standardwert lautet 'Ja', wenn <i>url</i> angegeben ist. Andernfalls lautet der Standardwert 'Nein'.

3. Konfigurieren Sie das Benutzerscript.

Legen Sie die Speicherposition Ihres Benutzerprogramms in BWBSQLB fest. Dadurch erhält der Erstellungsprozess die Information, wo sich das Routenerweiterungsscript befindet, das zum Berechnen der dynamischen Eigenschaften verwendet wird.

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

Kapitel 4. SCLM-Sicherheit

SCLM Developer Toolkit bietet optionale Sicherheitsfunktionen für die Builderstellung, Umstufung und Implementierung.

- Die Sicherheitsfunktionalität wird durch die Sicherheitsattribute gesteuert, die in den Konfigurationsdateien von SCLM Developer Toolkit und durch Profile in Ihrem Sicherheitsprodukt festgelegt sind.
- Wenn das Sicherheitsattribut für eine Funktion festgelegt ist und die anfordernde Benutzer-ID die Berechtigungsprüfung Ihres Sicherheitsprodukts besteht, kann die Funktion fortgesetzt werden. Wenn der Anforderer die Berechtigungsprüfung nicht besteht, endet die Funktion mit RC=8 und einer Sicherheitsfehlernachricht.
- Wenn die Funktion in dieser Form in Ihrem Sicherheitsprodukt definiert ist, wird der Vorgang mit einer anderen (Ersatz-)Benutzer-ID fortgesetzt.

Sicherheitsflag

Sie können ein Sicherheitsattribut für die Erstellung, Umstufung und Implementierung in den SITE-/PROJECT-Konfigurationsdateien festlegen. Weitere Informationen zu den SITE-/PROJECT-Konfigurationsdateien finden Sie unter „SITE- und projektspezifische Optionen“ auf Seite 28. Die folgenden Anweisungen steuern das jeweilige Sicherheitsattribut der Erstellungs-, Umstufungs- und Implementierungsfunktion:

- BUILDSECURITY = Y
- PROMOTSECURITY = Y
- DEPLOYSECURITY = Y

Einrichtung in Ihrem Sicherheitsprodukt

Die Erstellungs-, Umstufungs- und Implementierungsfunktion in SCLM verwendet die Sicherheitsschnittstelle SAF/RACROUTE, welche von allen gängigen Sicherheitsprodukten unterstützt wird.

Die aufgeführten Beispielbefehle gelten für RACF. Wenn Sie ein anderes Sicherheitsprodukt verwenden, ziehen Sie die entsprechende Dokumentation zu Rate.

- Verwenden Sie den Befehl RDEFINE, um für RACF alle Ressourcen zu definieren, die zu Klassen gehören, die in der Klassendeskriptortabelle angegeben sind. Der Befehl RDEFINE fügt ein Profil für die Ressource zur RACF-Datenbank hinzu, um den Zugriff auf die Ressource zu steuern.
- Der Befehl PERMIT wird verwendet, um die Listen von Benutzern und Gruppen zu verwalten, die berechtigt sind, auf eine bestimmte Ressource zuzugreifen. Die Standardzugriffsliste enthält Benutzer-IDs und Gruppennamen, die berechtigt sind, auf die Ressource und die jeweils gewährte Zugriffsebene zuzugreifen.
- Definieren Sie Funktionsprofile für SCLM-Funktionen für die XFACILIT-Klasse.
- Der Sicherheitsadministrator definiert die erforderlichen RACF-Profile mit dem RDEFINE-Befehl und erlaubt Ihnen den Zugriff mit dem Befehl PERMIT.

Sicherheitsprofile

Der Sicherheitsadministrator definiert die erforderlichen Profile in der XFACILIT-Klasse mit dem Befehl **RDEFINE** sowie die Zugriffsberechtigungen mit dem Befehl **PERMIT**. In Tabelle 11 erfahren Sie, welche Profile definiert werden müssen.

Tabelle 11. SCLMDT Developer Toolkit-Sicherheitsprofile

Funktion	XFACILIT-Profil	Erforderlicher Zugriff
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

In der unten stehenden Liste wird die Bedeutung der verschiedenen Profiltailnamen beschrieben:

SCLM	Konstante, gibt ein SCLM-Funktionsprofil an
BUILD	Konstante, gibt die BUILD-Funktion an
PROMOTE	Konstante, gibt die PROMOTE-Funktion an
DEPLOY	Konstante, gibt die DEPLOY-Funktion an
project	SCLM-Projektname oder * für alle Projekte
projdef	Alternativer Projektname (standardmäßig übereinstimmend mit dem Projektnamen) oder * für alle alternativen Projekte
Group	SCLM-Gruppe, die erstellt/umgestuft/implementiert werden soll, oder * für alle Gruppen
Type	SCLM-Typ oder * für alle Typen
Member	SCLM-Member, das erstellt/umgestuft/implementiert werden soll, oder * für alle Member
Server	Ziel-Implementierungsserver (SERVER_NAME im Ant-Implementierungsscript) oder * für alle Server
Application	Name der Ziel-WAS-Anwendung (APPLICATION_NAME im Ant-Implementierungsscript) oder * für alle Anwendungen
Node	Name des Ziel-WAS-Knotens (NODE_NAME im Ant-Implementierungsscript) oder * für alle Knoten
Cell	Name der Ziel-WAS-Zelle (CELL_NAME im Ant-Implementierungsscript) oder * für alle Zellen

Ersatzbenutzer-ID

Die Erstellungs-, Umstufungs- und Implementierungsfunktionen unterstützen die Verwendung einer Ersatzbenutzer-ID zum Ausführen der Funktion. Wenn diese aktiviert ist, bewirken alle autorisierten Aufrufe der Funktion, dass die Funktion mit den Berechtigungen der Ersatzbenutzer-ID anstelle der anfordernden Benutzer-ID ausgeführt wird.

Die Aktivierung der Ersatzbenutzer-ID ist profilspezifisch und wird gesteuert durch die Zeichenfolge "SUID=userid" im Feld APPLDATA im Sicherheitsprofil, wobei userid die Ersatzbenutzer-ID ist. Wenn die Zeichenfolge vorhanden ist, wird die Ersatzbenutzer-ID verwendet, wenn nicht, wird die Benutzer-ID des Anforderers verwendet.

Beispiel: Erstellung

In diesem Beispiel werden die Sicherheitsproduktdefinitionen aufgelistet, die benötigt werden, um die Erstellungsfunktion für ein bestimmtes Projekt zu schützen.

Dasselbe Beispiel kann verwendet werden, um die Umstufungsfunktion zu schützen, indem die Regel SCLM.BUILD.* durch die Regel SCLM.PROMOTE.* ersetzt wird.

Die folgende Profildefinition schützt alle Member für die Erstellung im Projekt =TESTPROJ auf der Gruppenebene PROD. Darüber hinaus wird eine Ersatzbenutzer-ID definiert.

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

In diesem Beispiel wird ein SCLM-Erstellungsprofil definiert. Dabei gilt Folgendes:

- Projekt = TESTPROJ
- Alternative Projektdefinition = TESTPROJ
- SCLM-Gruppe = PROD
- SCLM-Typ = alle Typen
- Member = alle Member
- Umfassende Erreichbarkeit = NONE (standardmäßig ist niemand für das Profil berechtigt)
- Ersatzbenutzer-ID = IBMUSER

Anmerkung: Der Befehl SETROPTS aktualisiert die speicherinternen Tabellen von RACF und aktiviert dadurch die Definition.

Das folgende Beispiel zeigt Sicherheitsberechtigungen, die für einzelne Benutzer (oder Benutzergruppen) für das TESTPROJ-Projekt des vorherigen Beispiels definiert wurden:

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

Der Wert PERMIT entspricht dem ursprünglichen RDEFINE-Profil und erlaubt dem Benutzer USERID, ein beliebiges Member aus dem Projekt TESTPROJ und der Gruppe PROD zu erstellen. Da eine Ersatzbenutzer-ID im Anwendungsdatenfeld (APPLDATA) des entsprechenden Profils gespeichert ist, wird der BUILD unter dieser Ersatzbenutzer-ID verarbeitet (in diesem Fall IBMUSER).

Beispiel: Implementierung

Im Folgenden sehen Sie ein Beispiel für die Erstellung eines Implementierungsprofils.

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

WAS-Details:

- Servername = SERVERY
- Anwendungsname = TESTAPP
- Knotenname = NODE1
- Zellename = CELL1

SCLM-Projektdetails:

- Projekt = TESTPROJ
- Alternative Projektdefinition = TESTPROJ
- SCLM-Gruppe = PROD

- SCLM-Typ = J2EEDEP

Weitere Informationen:

- Umfassende Erreichbarkeit = NONE (standardmäßig ist niemand für das Profil berechtigt)
- Keine Ersatzbenutzer-ID

Das folgende Beispiel zeigt Sicherheitsberechtigungen, die für eine Benutzergruppe des zuvor definierten Implementierungsprofils definiert wurden:

```
PERMIT SCLM.DEPLOY.SERVERY.TESTAPP.NODE1.CELL1.TESTPROJ.TESTPROJ.PROD.J2EEDEP
      CLASS(XFACILIT) ID(J2EEGRP) ACCESS(READ)
```

Dies entspricht dem ursprünglichen RDEFINE-Profil und erlaubt allen Benutzern, die zur RACF-Gruppe J2EEGRP gehören, auf dem obigen Server und von denselben SCLM-Projektetails aus zu implementieren.

Kapitel 5. CRON-aktivierte Erstellungen und Umstufungen

Auch wenn die meisten Erstellungen und Umstufungen über den Developer Toolkit-Client initialisiert werden, besteht außerdem die Möglichkeit, Erstellungs- und Umstufungskonfigurationsdateien im z/OS UNIX System Services-Dateisystem einzurichten und diese Erstellungen oder Umstufungen durch den CRON-(Zeit-)Service in UNIX System Services zu initialisieren.

Wenn Sie diese Methode verwenden, wird der SCLM Developer Toolkit-Client nicht benötigt, da die relevanten Erstellungs- und Umstufungsparameter aus einer z/OS UNIX System Services-Dateisystem-Konfigurationsdatei gelesen werden und an die Hostkomponente von Developer Toolkit für die Verarbeitung in SCLM übermittelt werden.

Unten finden Sie eine Beschreibung der Beispiele von SCLM Developer Toolkit, die über CRON initialisierte Erstellungen und Umstufungen bereitstellen. Diese Beispiele sind im Beispielverzeichnis von Developer for System z unter `/usr/lpp/rdz/samples/` verfügbar. Eine Kopie, die Ihren Anforderungen angepasst werden kann, wurde während der Anpassung von Developer for System z unter `/etc/rdz/sclmdt/script/` abgelegt.

BWBCRON1

Dieses REXX-Beispiel ruft die Hostschnittstelle von SCLM Developer Toolkit auf und übergibt die Funktionsparameter. Die Ausgabe aus dem Funktionsprozess wird in STDOUT angezeigt, kann jedoch eventuell in eine z/OS UNIX-Datei oder ein Protokoll umgeleitet werden. REXX muss angepasst werden, wie im Beispiel angegeben. Dieses REXX-Beispiel muss bei Erstellungen in Verbindung mit einer Eingabe aus dem Beispiel BWBCRONB und bei Umstufungen mit einer Eingabe aus dem Beispiel BWBCRONP ausgeführt werden.

BWBCRONB

Dieses REXX-Beispiel richtet die Erstellungsparameter-Eingabezeichenfolge ein, die an das Modul BWBCRON1 übermittelt wird. Für dieses Beispiel ist eine Benutzeranpassung erforderlich, um alle benötigten Erstellungsparameter zu aktualisieren.

BWBCRONP

Dieses REXX-Beispiel richtet die Umstufungsparameter-Eingabezeichenfolge ein, die an das Modul BWBCRON1 übermittelt wird. Das Beispiel erfordert Benutzeranpassung, um alle erforderlichen Umstufungsparameter zu aktualisieren.

bwbsqld.rex

Beispiel-ANT-XML-SQLJ-JAVA-Erstellungsscript

STEPLIB- und PATH-Anforderungen

Die Variablen PATH und STEPLIB im systemweiten Profil (`/etc/profile`) oder Benutzerprofil (`/u/userid/.profile`) müssen festgelegt werden, um die CRON-Jobs (\$PATH) zu lokalisieren und die SCLM Developer Toolkit-Lademodule (\$STEPLIB) zu lokalisieren, wenn die SCLM Developer Toolkit-Lademodule sich nicht in der LINKLIST befinden. Beispiel:

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

CRON-Erstellungsjob, Ausführung

Nach dem Hinzufügen der CRON-Jobs zur Pfadvariable, können diese durch Verkettungen der Ausgabe von `parameter_exec` mit `processing_exec` ausgeführt werden. Die Ausgabe kann dann in eine Ausgabeprotokolldatei umgeleitet werden.

Syntax

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

Das Zeichen "|" ist das Verkettungszeichen von z/OS UNIX.

Beispiel für den Aufruf

Wenn Sie die angegebenen Beinamen verwenden, kann die CRON-Erstellungs-Exec wie folgt aufgerufen werden (\$ ist die z/OS UNIX-Eingabeaufforderung):

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

Weitere Informationen zu den verfügbaren CRON-Services und dem Format CRONTAB finden Sie in den Dokumenten *UNIX System Services Command Reference* (IBM FORM SA22-7802-11) und *UNIX System Services Planning* (IBM FORM GA22-7800-16).

Alternativ können Sie den Online-z/OS UNIX-Befehl **man** verwenden:

- man cron
- man crontab
- man at

CRON-Erstellungsjob, Beispiel

In diesem Abschnitt werden die Jobmuster BWBCRON1, BWBCRONB, BWBCRONP angezeigt, die in der SFEKSAMV-Bibliothek bereitgestellt werden.

Das folgende Beispiel zeigt den 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 USAGE */
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

```

Abbildung 18. BWBCRON1 - CRON Build exec

Das folgende Beispiel zeigt den 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

```

Abbildung 19. BWBCRONB - Erstellungsparameterdatei

Das folgende Beispiel zeigt den 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 */
PRMSGDS = '' /* 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||',
        '&PRMSGDS='PRMSGDS'&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

```

Abbildung 20. BWBCRONP - Promote parameter file

Anhang A. SCLM-Übersicht

SCLM Developer Toolkit, eine Funktion von IBM Rational Developer for System z, bietet eine Möglichkeit, mit der verteilte Anwendungen, die in Eclipse geschrieben sind, mit Software Configuration and Library Manager (SCLM) verwaltet und erstellt werden können. Dies ist das Managementsystem für IBM z/OS-Quellcode.

Die Sprache und Tools, die von verteilten und Mainframe-Benutzern verwendet werden, unterscheiden sich ebenso wie die verwendeten Umgebungen. Durch Kenntnis der entscheidenden Konzepte beider Umgebungen ist es möglich, diese erfolgreich in einer kohäsiven Struktur zu integrieren.

Im Hinblick auf die Anwendungsstruktur besteht SCLM Developer Toolkit aus einer Serie von Eclipse-Plug-ins mit entsprechendem z/OS-Host-Code, der die Verwendung von HTTP- und RSE-Transporten ermöglicht. Für den Betrieb registriert der Eclipse-Entwickler ein Arbeitsbereichsprojekt in SCLM. Die Dateien im Projekt können zu einem SCLM-Projekt hinzugefügt werden, ein- und ausgecheckt und optional erstellt und implementiert werden. Alle dieses Services werden über das Team-Aktionsmenü gesteuert. Aus Sicht des SCLM-Administrators können Projekte, Typen, Sprachen und zugeordnete erstellte Umsetzer erstellt werden. Die Funktionen, wie Änderungs- und Berechtigungscode, sind abhängig von den Anforderungen.

SCLM-Konzepte

Aus dem Blickwinkel eines Java/J2EE-Entwicklers sind folgende Konzepte zur Beschreibung von SCLM hilfreich:

Dateibenennung

Das z/OS-Dateisystem unterstützt nur Dateinamenlängen von acht Zeichen. Die Schnittstelle von SCLM Developer Toolkit bietet einen Umsetzungsservice, der die Unterstützung von normalen Java-Langnamenskonventionen ermöglicht. Bestimmte SCLM-Dateien müssen den Einschränkungen für die Benennung entsprechen. Diese beziehen sich vor allem auf die in „J2EE ARCHDEF-Format“ auf Seite 9 beschriebene ARCHDEF-Struktur.

Typ

Jede Datei (in der SCLM-Terminologie als „Member“ bezeichnet), die in einem SCLM-Projekt gespeichert wird, wird in einer Dateigruppe gespeichert. Die Dateigruppe, in der die Datei gespeichert ist, wird durch den SCLM-Typ identifiziert. Der Typ ist Teil des Dateinamens. In Bezug auf SCLM hat der Name das Format SCLM-Projekt.Gruppe.Typ mit einer zugewiesenen Sprache. In einem SCLM-Projekt können zahlreiche Typen definiert werden. Mithilfe dieser Typen können zwei Dateien mit demselben Namen innerhalb desselben SCLM-Projekts gespeichert werden. Jedes Projekt kann zahlreiche Typen enthalten. Durch die Verwendung von Typen ist es möglich, mehrere Eclipse-Projekte in einem SCLM-Projekt zu speichern, auch wenn jedes IDE-Projekt möglicherweise Dateien mit der Bezeichnung „project“ und „classpath“ enthält. Wenn diese Dateien nicht durch die Verwendung von Typen unterschieden werden, wird nur eine Kopie der Dateien in SCLM gespeichert.

Der SCLM-Administrator ist verantwortlich für die Definition der SCLM-Typen. Wenn Sie ein Projekt in SCLM gemeinsam nutzen, muss Ihnen bekannt sein, welchen Typ Sie beim Speichern der Objekte in SCLM verwenden müssen.

Sprache

Wenn Sie eine Datei zu SCLM hinzufügen, müssen Sie diese mit einer bestimmten Sprachdefinition speichern. Auch hier ist der SCLM-Administrator verantwortlich für die Definition der Sprachen. Diese Definition steuert das Verhalten von SCLM, wenn Dateien in und aus dem Hostsystem übertragen werden. Durch Verwendung der Sprachdefinition ist es möglich, zu definieren, ob ein bestimmter Dateityp aus einem ausgeschriebenen Namen umgesetzt wird, als binäres Objekt gespeichert wird oder in ASCII oder EBCDIC umgesetzt wird (native z/OS-Codierung). Eine Sprachdefinition des Typs JAVABIN kann z. B. zu einem Binärobjekt mit einem umgesetzten Langnamen gehören. Alternativ kann WEBHTML so definiert werden, dass eine Textdatei mit umgesetztem Langnamen repräsentiert wird, die in ASCII gespeichert wird. Die Zahl der Sprachdefinitionen wird objektweise definiert. Es muss bekannt sein, welche Sprache verwendet werden soll, um sicherzustellen, dass die Datei gespeichert wird und ordnungsgemäß aus SCLM abgerufen werden kann. Die Sprache definiert auch, wie SCLM ein Objekt erstellt.

SCLM-Eigenschaften

Jeder Datei, die durch SCLM gesteuert wird, ist eine bestimmte Zahl von Eigenschaften zugeordnet. Diese Eigenschaften werden für die Zuordnung zwischen der IDE-Datei und den entsprechenden SCLM-Eigenschaften verwendet. Wenn ein Serviceaufruf an SCLM gerichtet wird, werden diese Daten gelesen, um die entsprechenden Serviceparameter zu formulieren. Sie können diese im Eigenschaftmenü anzeigen, wenn Sie ein durch SCLM gesteuertes Member aus Eclipse markieren.

SCLM-Projektstruktur

Wenn Sie ein Projekt in SCLM gemeinsam nutzen, müssen Sie auch angeben, welcher Entwicklungsgruppe Sie angehören. SCLM-Projektstrukturen sind hierarchisch aufgebaut.

Der Code wird zunächst auf DEVELOPMENT-Ebene gespeichert. Nachdem dieser erfolgreich erstellt und getestet wurde, kann der Code auf die TEST-Ebene umgestuft werden. Nach erfolgreichen Tests kann der Code anschließend auf die PRODUCTION-Ebene umgestuft werden. Dabei handelt es sich im Allgemeinen um das entwickelte Produkt. Wenn ein Fehler im Code auf Produktionsebene gefunden wird, werden die entsprechenden Dateien, die bearbeitet werden müssen, um den Fehler zu beheben, in die Entwicklungsebene herunterkopiert und der Erstellungsprozess wird erneut gestartet. Der Code, aus dem die Anwendung besteht, wird nicht in die Entwicklungsebene herunterkopiert. SCLM verfolgt die Elemente, die den Build bilden, sowie die Ebene, auf der diese gespeichert sind.

Innerhalb der Entwicklungsebene können mehrere Gruppen vorhanden sein. Dadurch kann Code, der auf der Entwicklungsebene gespeichert wird, getrennt verwaltet werden. SCLM bietet außerdem Steuerelemente, die das Verhalten von Codes, die in verschiedenen Entwicklungsgruppen gespeichert sind, im Hinblick auf die Umstufungsmöglichkeiten bestimmen.

ARCHDEF

Die Struktur eines IDE-Projekts besteht im Allgemeinen aus mindestens einem IDE-Projekt. Durch die Speicherung jedes IDE-Projekts in einem anderen SCLM-Typ wird diese Struktur erhalten. Die ARCHDEF-Datei definiert dann die Dateien,

aus denen ein IDE-Projekt besteht. Jedes SCLM-Projekt kann mehrere ARCHDEFs haben. Es ist möglich, dass eine ARCHDEF andere ARCHDEFs referenziert. Daher kann eine Struktur mit mehreren IDE-Projekten für den Erstellungsprozess definiert werden. Die ARCHDEF ist dabei die wichtigste Methode für die Definition einer Erstellungsliste für SCLM. Dies lässt sich am ehesten mit einem make-Prozess vergleichen. Die ARCHDEF listet die Dateien auf, die den Build bilden, und gibt ein Erstellungsscript an, mit dem Sie die Speicherposition von externen JARs oder Klassen angeben können. Weitere Informationen hierzu finden Sie im Bereich „Benutzerhandbuch“ des Onlinehilfesystems.

JAVA/J2EE-Konzepte

Wenn ein IDE-Projekt im Arbeitsbereich erstellt wird, wird automatisch eine Beschreibungsdatei generiert und unter dem Namen **.project** gespeichert. Dieses XML-Dokument enthält Beschreibungen sämtlicher Builder und Spezifiken, die dem Projekt zugeordnet sind. „Builder“ sind Programme zur inkrementellen Projekterstellung, die abhängig vom Projekteinhalt einen bestimmten Buildstatus erstellen. Wenn sich der Inhalt des Projekts ändert, wird diese Datei aktualisiert. „Spezifiken“ definieren und verwalten die Zuordnung zwischen einem angegebenen Projekt und einem bestimmten Plug-in oder einer Komponente.

Die **.classpath**-Datei beschreibt den Pfad, der zum Auffinden externer JARs und Klassen verwendet wird, die durch den Quellcode in Ihrem IDE-Projekt referenziert werden. Die funktional entsprechende Funktion während einer Erstellung durch SCLM Developer wird mit der Anweisung **CLASSPATH_JARS** in den Ant-Erstellungsscripts definiert. Diese Anweisung beschreibt den Pfad auf dem z/OS-Host, der zum Auffinden von externen JARs und Klassen verwendet wird, die durch den Quellcode in Ihrem IDE-Projekt referenziert werden.

Sowohl **.classpath** als auch **.project** werden verwendet, um Ihre IDE-Projektkonfiguration beizubehalten, damit diese in einem anderen Arbeitsbereich neu erstellt werden kann. Aus diesem Grund wird empfohlen, dass beide als Teil des IDE-Projekts in SCLM eingecheckt werden.

Ein wichtiger Aspekt bei der Projektentwicklung, insbesondere bei J2EE-Projekten, sind die verschiedenen ausführbaren Anwendungsdateien, die erstellt werden können. Ausführbare Java-Projektdateien werden häufig als JAR-, WAR-, RAR- oder EAR-Dateien paketi.

JAR-Dateien beziehen sich üblicherweise auf Standard-Java-Anwendungen oder Enterprise Java Beans (EJB).

WAR-Dateien werden für Webanwendungen erstellt. Üblicherweise setzen sich diese aus Java-Servlets, JSPs und HTML-Dateien zusammen. WAR-Anwendungen stellen oft das Front-End von webbasierten Anwendungen dar. Diese Anwendungen können sich auf andere JAR-Dateien beziehen, z. B. EJB-Dateien für bestimmte Services. Jede WAR-Datei enthält eine Datei mit dem Namen **web.xml**. Diese beschreibt die Zusammensetzung der WAR-Anwendung im Hinblick auf Java, HTML und die verwendeten Bibliotheksservices.

Die Entwicklung von RAR-Dateien wird derzeit nicht in SCLM Developer Toolkit unterstützt.

EAR-Dateien repräsentieren Unternehmensanwendungen. Diese Anwendungen setzen sich aus JAR- und WAR-Dateien zusammen. In J2EE-Sprache ist die Erstellung der EAR-Datei die Assemblierung der konstituierenden JAR- und WAR-Dateien.

Diese Assemblierungsmethode ermöglicht die Erstellung von EAR-Anwendungen, die sich effektiv aus bestimmten Komponenten zusammensetzen (JAR/WAR). Die tatsächliche Zusammensetzung der Anwendung wird in der Datei „application.xml“ beschrieben. Die EAR-Datei selbst ist kein eigenständiges ausführbares Objekt. Die EAR-Datei muss in einem J2EE-Container wie Websphere Application Server (WAS) installiert werden. Die Installation der EAR-Datei wird als „Implementierung“ bezeichnet. Eine implementierte EAR-Anwendung kann über die WAS-Umgebung aufgerufen werden. Der Prozess der Implementierung ist ein separater Vorgang, der unabhängig vom Erstellungsprozess ausgeführt wird. Die Implementierung beinhaltet die physische Installation der EAR-Anwendung.

Für die Entwicklung von J2EE-Anwendungen kann daher die Entwicklung mehrerer separater Komponenten wie WAR- und JAR-Dateien erforderlich sein. Diese Dateien werden gemeinsam in einer EAR-Datei assembliert. Die EAR-Datei kann dann in einem J2EE-Container (z. B. WAS) für die Verwendung implementiert werden.

Im Eclipse-Arbeitsbereich liegen die Projekte nah beieinander, d. h. sie können in der IDE-Umgebung hinsichtlich der Paketierung auf andere IDE-Projekte verweisen. In SCLM muss jedes dieser IDE-Projekte (z. B. WAR-, JAR- und EAR-Projekte) einem einzelnen SCLM-Projekt zugewiesen werden, wobei die Projekte durch die Verwendung verschiedener SCLM-Typen unterschieden werden. Der Grund hierfür ist, dass es allgemeine Dateinamen gibt, die in vielen IDE-Projekten verwendet werden, z. B. „project“, „classpath“, „web.xml“ und „application.xml“. Durch die Verwendung unterschiedlicher Typen können mehrere Teile mit demselben Namen innerhalb eines SCLM-Projekts vorhanden sein. Weitere Informationen zur Zuordnung finden Sie unter „Zuordnen, J2EE-Projekte zu SCLM“ auf Seite 18.

Aus der SCLM-Perspektive lässt sich die Entwicklung der EAR-Anwendung am besten durch die Verwendung einer ARCHDEF-Struktur mit mehreren Ebenen darstellen. In SCLM bilden die übergeordneten ARCHDEFs, in vielen SCLM-Projekten als *Paket* bezeichnet, den Scheitelpunkt der ARCHDEF-Struktur, gefolgt von der EAR-Anwendung und Referenzen auf niedrigerer Ebene (WAR- und JAR-Dateien), aus denen die EAR-Anwendung besteht. Diese Struktur ermöglicht die Verwendung von Builds auf hoher und niedriger Ebene sowie die Verwendung von vollständigen oder bedingten Builds. Die ARCHDEFs bieten somit eine Möglichkeit, mit der es auch möglich ist, die J2EE-Projektelemente innerhalb des SCLM-Projekts zu definieren.

Anhang B. Umsetztabelle für Lang-/Kurznamen

Derzeit unterstützt der zentrale Bestandteil von SCLM keine Speicherung von Codes mit Datei- bzw. Membernamen, die länger als acht Zeichen sind.

Codes, z. B. Java- und andere PC-Client-Codes, haben üblicherweise viel längere Namen und enthalten auch Pfadinformationen (Paketierung) als Teil des Namens. Codes mit benannten Teilen, die länger als acht Zeichen sind, müssen daher mit einem Dienstprogramm für die Umsetzung von Lang- in Kurznamen bearbeitet werden, damit diese Teile in SCLM mit einer Namenslänge von höchstens acht Zeichen gespeichert werden können.

In einer Tabelle für die Umsetzung von ausgeschriebenen Namen in Kurznamen werden die ausgeschriebenen Namen (echten Namen) und die entsprechenden Kurznamen (wie in SCLM gespeichert) erfasst. Diese Tabellen werden über SCLM gespeichert und aufgerufen und in einem VSAM-Datensatz gespeichert. Diese Funktionalität wurde in SCLM mit der vorläufigen Programmkorrektur für APAR OA11426 für z/OS 1.4 und höher eingeführt. Für z/OS 1.8 und höher wird diese vorläufige Programmkorrektur nicht benötigt.

Der Konvertierungsalgorithmus führt folgende Schritte aus:

1. Das Umsetzungspräfix besteht aus den ersten beiden Zeichen (Großbuchstaben) des ausgeschriebenen Programm-/Dateinamens (d. h. des letzten Dateinamens nach dem Zeichen "/" im mehrfach paketierte Format). Wenn die ersten beiden Zeichen nicht als Präfix für einen Host-Membernamen gültig sind (weil sie ungültige Sonderzeichen enthalten), dann lautet das Präfix „XX“. Für Spezialfälle wie z. B. alphabetische Namen mit einem einzelnen Zeichen (/u/test/A oder /u/test/A.java) wird ebenfalls das Präfix „XX“ zugewiesen.
2. Die letzten sechs Zeichen werden numerisch der nächsten in der Tabelle verfügbaren fortlaufenden Zahl zugeordnet.

Beispiel

Ausgeschriebener Name	Kurzname in SCLM PDS oder PDSE
com/ibm/workbench/testprogram.java	TE000001
source/plugins/Phantom/.classpath	XX000001

Technische Zusammenfassung des SCLM-Umsetzungsprogramms

Das SCLM-Programm FLMLSTRN wurde zum Lesen und Aktualisieren der VSAM-Umsetztabelle erstellt. SCLM Developer Toolkit verwendet dieses Programm zum Lesen und Aktualisieren von korrelierenden Lang- und Kurznamen.

Die VSAM-Datei, die zum Speichern der Umsetztabelle verwendet wird, ist eine Datei in Schlüsselfolge mit variabler Länge mit einem definierten Alternativindex und -pfad. Ein Beispieljob wird in SCLM bereitgestellt, um diese VSAM-Datei zuzuordnen.

Die interne Struktur des VSAM-Clusters ist wie folgt gestaltet:

```

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

```

Eine Beispiel-Jobsteuersprache für die Zuordnung der VSAM-Datei für die Umsetzung von Lang-/Kurznamen sehen Sie in Schritt 6: Konfigurieren der VSAM-Datei für Lang-/Kurznamentabellen.

Anmerkung: Die folgenden technischen Informationen zu SCLM-Umsetztabellen-Funktionsaufrufen werden allein zu Informationszwecken angegeben und werden nicht benötigt, um Funktionen von SCLM Developer Toolkit zu aktivieren.

Das Programm FLMLSTRN wird mit dem ISPF-SELECT-Service mit einem der in Tabelle 12 aufgelisteten Parameter aufgerufen.

Syntax:

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

Beispiel für einen Aufruf:

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

Tabelle 12. Parameter für die Umsetzung von Lang-/Kurznamen.

Schlüsselwortdatensatz	Verarbeitung	Beschreibung
FINDLONG	Einzeln	Findet einen ausgeschriebenen Namen für einen bestimmten Kurznamen
FINDSHORT	Einzeln	Findet einen Kurznamen für einen bestimmten ausgeschriebenen Namen
TRANSLATE	Einzeln	Findet einen Kurznamen, falls vorhanden, oder ordnet einen neuen Kurznamen zu, falls keiner vorhanden ist
MIGRATE	Mehrfach	Sucht nach mehreren ausgeschriebenen Namen
IMPORT	Mehrfach	Sucht nach mehreren Kurznamen

Verarbeitung einzelner Lang-/Kurznamensätze

FINDLONG-Verarbeitung

- Der VSAM-Cluster, der DD LSTRANS zugeordnet ist, wird im Lesemodus geöffnet.
- Der Kurzname wird aus der ISPF-Variablen FLMLSSHR abgerufen und zum Lesen der VSAM-Datei verwendet.
- Wenn der Datensatz nicht gefunden wird, wird eine Nachricht über die ISPF-Variable FLMLSERR zurückgegeben, dass der ausgeschriebene Name nicht gefunden wurde.
- Wenn der ausgeschriebene Name gefunden wurde, wird dieser in der ISPF-Variablen FLMLSLNG zurückgegeben.
- Der VSAM-Cluster wird geschlossen.

FINDSHORT-Verarbeitung

- Der VSAM-Pfad, der DD LSTRNPTH zugeordnet ist, wird im Lesemodus geöffnet.
- Der ausgeschriebene Name wird aus der ISPF-Variablen FLMLSLNG abgerufen.
- Die letzten 50 Byte des ausgeschriebenen Namens werden zum Lesen des Pfads verwendet.
- Wenn kein Datensatz zurückgegeben wird, wird eine Nachricht über die ISPF-Variable FLMLSERR zurückgegeben, dass der Kurzname nicht gefunden wurde.
- Wenn ein Datensatz zurückgegeben wird, wird der ausgeschriebene Name im VSAM-Datensatz mit dem ausgeschriebenen Namen in der ISPF-Variablen FLMLSLNG abgeglichen.
- Wenn keine Übereinstimmung vorhanden ist, werden die VSAM-Datensätze gelesen und verglichen, bis `ls_lngname_key` nicht übereinstimmt oder der ausgeschriebene Name gefunden wird.

Anmerkung: `ls_lngname_key` lässt Duplikate zu, da ein VSAM-Datensatz mit demselben Schlüssel, aber einem anderen ausgeschriebenen Namen vorhanden sein kann.

- Wenn der Kurzname gefunden wurde, wird dieser in der ISPF-Variablen FLMLSHR zurückgegeben.
- Der VSAM-Pfad wird geschlossen.

TRANSLATE-Verarbeitung

Die Verarbeitung erfolgt in gleicher Weise wie bei FINDSHORT, und zwar folgendermaßen:

- Wenn der Kurzname gefunden wird, wird keine weitere Verarbeitung ausgeführt.
- Wenn der Kurzname nicht gefunden wird, wird der VSAM-Cluster, der DD LSTRANS zugewiesen ist, im Aktualisierungsmodus geöffnet.
- Der Dateiname wird durch Suche des letzten '/' oder '\' im ausgeschriebenen Namen ermittelt.
- Die ersten 2 Byte des Dateinamens werden verwendet, um den VSAM-Dateipräfixsatz zu suchen, der eine Zahl enthält.
- Das Dateipräfix und die Zahl werden verwendet, um den Kurznamen zu generieren (z. B. PR000123).
- Der generierte Kurzname (PR000123) wird verwendet, um in der VSAM-Datei zu überprüfen, ob der Kurzname verwendet wird.
- Wenn dies der Fall ist, wird die Präfixzahl erhöht und der Kurzname wird erneut überprüft.
- Dieser Prozess wird fortgesetzt, bis ein Kurzname gefunden wird, der nicht verwendet wird.
- Der Präfixsatz wird aktualisiert und anschließend wird der neue Umsetzungssatz hinzugefügt.
- Der Kurzname wird in der ISPF-Variablen FLMLSSHR zurückgegeben.
- Der VSAM-Cluster wird geschlossen.

Verarbeitung mehrerer Lang-/Kurznamensätze

Die Funktionen MIGRATE und IMPORT wurden eingeführt, um das Leistungsverhalten bei einer großen Zahl von umzusetzenden Langnamen (MIGRATE) bzw. bei einer großen Zahl von durchsuchten Kurznamen (IMPORT) zu verbessern.

Beide Funktionen, MIGRATE und IMPORT, lesen eine durch Variablen blockierte sequenzielle Datei mit LRECL=1036, die als DD LSTRNPRC zugeordnet wird.

Vor dem Aufruf enthält diese Datei abhängig von der aufgerufenen Funktion Kurznamen oder ausgeschriebener Name im richtigen Format und in der richtigen Spalte.

Nach dem Aufruf, enthält LSTRNPRC sowohl den Kurznamen als auch den entsprechenden ausgeschriebenen Namen.

Die Datei hat folgendes Format:

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

IMPORT-Verarbeitung

- Der VSAM-Cluster, der DD LSTRANS zugewiesen ist, wird im Lesemodus geöffnet und die Verarbeitungsdatei, die DD LSTRNPRC zugewiesen ist, wird zur Aktualisierung geöffnet.
- Der Kurzname der einzelnen Datensätze in der Verarbeitungsdatei wird verwendet, um die VSAM-Umsetzungsdatei zu lesen. Wenn ein Datensatz gefunden wird, wird die Verarbeitungsdatei mit dem ausgeschriebenen Namen aktualisiert.
- Der VSAM-Cluster/die Verarbeitungsdateien werden geschlossen.

MIGRATE-Verarbeitung

- Der VSAM-Cluster, der DD LSTRANS zugewiesen ist, wird im Lesemodus geöffnet und die Verarbeitungsdatei, die DD LSTRNPRC zugewiesen ist, wird zur Aktualisierung geöffnet.
- Der ausgeschriebene Name der einzelnen Datensätze in der Verarbeitungsdatei wird zum Lesen der VSAM-Datei verwendet. Falls ein Datensatz gefunden wurde, wird der Datensatz der Verarbeitung mit dem entsprechenden Kurznamen aktualisiert. Andernfalls wird LSTRANS im Aktualisierungsmodus geöffnet, um neue Einträge für ausgeschriebener Namen/Kurznamen hinzuzufügen und der neue generierte Kurzname wird in die Datei LSTRNPRC zurückgeschrieben.
- Der VSAM-Cluster/die Verarbeitungsdateien werden geschlossen.

Das folgende Beispiel zeigt einen Beispiel-REXX-Code für den Aufruf des Umsetzungsprozesses für Lang-/Kurznamen.


```

/* 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)"

```

Abbildung 21. Beispiel REXX für den Aufruf des Umsetzungsmoduls

Anhang C. SCLM Developer Toolkit-API

In diesem Anhang wird die Anwendungsprogrammierschnittstelle (API) für die Host-Services von SCLM Developer Toolkit erläutert. Die API verwendet ein XML-basiertes Anforderungs- und Antwortformat und muss über z/OS UNIX Systems Services aufgerufen werden.

Mit der API können Benutzer die Host-Services von SCLM Developer Toolkit mit ihrem eigenen Client/Plug-in und ihrer eigenen Transportschicht verwenden, falls gewünscht.

Ein Beispiel-Java-Programm (mit Eingabe und Ausgabe) wird bereitgestellt. Dabei wird die SCLMDT-Host-Services-API mit einem HTTP-Server als Transportmechanismus aufgerufen.

Viele der Funktionsanforderungen basieren auf den aktuellen SCLM-Host-Services. Weitere Informationen zu ähnlichen Parametereinstellungen für allgemeine Funktionen finden Sie im SCLM-Handbuch sowie in der Dokumentation für das relevante z/OS-Release.

Anmerkung: Diese Anwendungsprogrammierschnittstelle dokumentiert die derzeit vom SCLM Developer Toolkit-Client verwendeten Host-Services. Daher sind viele der Funktionen möglicherweise weniger geeignet für die Verwendung durch Kunden bzw. erfordern weitere Überprüfung auf der Clientseite.

Aufrufformat

Der folgende Beispielfehl veranschaulicht, wie die SCLMDT-Host-Services aufgerufen werden können:

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

cat	z/OS UNIX-Befehl zum Anzeigen von Textdateien
sclmdt_request.xml	Die XML-Eingabedatei mit der Benutzeranforderung
	z/OS UNIX-Befehl zum Umleiten der Ausgabe des vorherigen Befehls über eine Pipe als Eingabe für den nächsten Befehl
BWBXML	Das XML-Konvertierungsscript, das sich in Developer for System z im Verzeichnis /bin befindet, ruft die Serviceschnittstelle SCLMDT auf.
>	z/OS UNIX-Befehl zum Umleiten der Ausgabe des vorherigen Befehls in eine Datei
sclmdt_response.xml	Die XML-Ausgabedatei, die die Serviceantwort enthält

Anmerkung:

1. Die Umgebungsvariable PATH muss die Verzeichnisposition von BWBXML enthalten, zum Beispiel:
export PATH=/usr/lpp/rdz/bin:\$PATH
2. Wenn HTTP als Transportmechanismus verwendet wird:
 - Das Schnittstellenscript BWBXML kann als CGI-Script aufgerufen werden (Typ EXEC in den HTTP-Serveranweisungen).

- Die Anforderung wird als STDIN durch BWBXML gelesen und kann daher als POST-Anforderung an das CGI-Script übermittelt werden (siehe Beispielprogramm).

XML-Schema für SCLMDT-Befehle

Das folgende Beispiel zeigt das XML-Schema für SCLMDT-Befehle, die in der XML-Eingabedatei referenziert werden. Dieses Beispiel ist auch als Member BWBXSD1 in der Beispielbibliothek FEK.SFEKSAMV verfügbar.

```

<?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,
                  prmmmsg,prmmmsgds,prmmextds,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:restriction>
              </xs:simpleType>
            </xs:element>
          </xs:all>
        </xs:complexType>
      </xs:element>
    </xs:all>
  </xs:complexType>
</xs:element>

```

Abbildung 22. XML-Schema für SCLMDT-Befehle

```

<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 -->
<!-- Deploy parms : project,projdef,group,repdgrp,type,member,
      depmode,depsec,groudpdy -->

      <xs:enumeration value="REPORT"/> <!-- SCLM project list -->
<!-- Report parms : project,projdef,reptype,repdgrp,repccode,replang,
      repacode,repmem,repgrp,repmode,repbmap -->

      <xs:enumeration value="MIGDSN"/> <!-- List non-sclm datasets
      & members -->
<!-- Migdsn parms : project,projdef,groupdev,migmem,migdsn -->

      <xs:enumeration value="MIGPDS"/> <!-- Migrate NON-SCLM
      datasets to SCLM -->
<!-- Migpds parms : project,projdef,group,type,migfile,migmode,
      authcode,ccode,migonly -->

      <xs:enumeration value="INFO"/>    <!-- SCLM member status
      information -->
<!-- Info parms : project,projdef,group,repdgrp,type,member,lang -->

      <xs:enumeration value="UPDATE"/> <!-- update SCLM member
      information -->
<!-- Update parms : project,projdef,group,repdgrp,type,member,
      lang,ccode,authcode -->

      <xs:enumeration value="AUTHUPD"/> <!-- Change SCLM member
      authcode -->
<!-- Authupd parms : project,projdef,group,type,member,authcode -->

      <xs:enumeration value="VERLIST"/> <!-- SCLM list versions -->
<!-- Verlist parms : project,projdef,group,repdgrp,type,member -->

      <xs:enumeration value="VERBROW"/> <!-- SCLM browse versions -->
<!-- Verbrow parms : project,projdef,group,repdgrp,type,member -->

      <xs:enumeration value="VERREC"/> <!-- SCLM recover versions -->
<!-- Verbrow parms : project,projdef,group,repdgrp,type,member -->

      <xs:enumeration value="VERDEL"/> <!-- SCLM delete versions -->
<!-- Verdel parms : project,projdef,group,repdgrp,type,member -->

      <xs:enumeration value="VERHIST"/> <!-- SCLM version history -->
<!-- Verhist parms : project,projdef,group,repdgrp,type,member -->

      <xs:enumeration value="REPUTIL"/> <!-- SCLM DBUTIL report -->
<!-- Reputil parms : project,projdef,group,repdgrp,type,member -->

      <xs:enumeration value="PROJGRPS"/> <!-- Retrieve SCLM groups for
      a project -->
<!-- Projgrps parms : project,projdef -->

```

Abbildung 23. XML-Schema für SCLMDT-Befehle (Fortsetzung)

```

        <xs:enumeration value="J2EEMIG"/> <!-- Migrate project into SCLM -->
<!-- J2eemig parms : project,projdef,group,type,member,lang,migmode,
        projarch,archtype,authcode,ccode,j2eetype,j2eesinc,J2eefile,
        sclmrefs,archac,archcc,submit -->

        <xs:enumeration value="J2EEMIGB"/> <!-- Batch migrate project
        into SCLM -->
<!-- J2eemigb parms : project,projdef,group,type,member,lang,migmode,
        authcode,ccode -->

        <xs:enumeration value="J2EEIMP"/> <!-- Import SCLM project -->
<!-- J2eeimp parms : project,projdef,group,repdgrp,type,member,repacode,
        repccode,replang,reparchm,reparcht,reparchg,replang -->

        <xs:enumeration value="PROJINFO"/> <!-- Retrieve SCLM project
        information -->
<!-- Projinfo parms : project,projdef,repdgrp -->

        <xs:enumeration value="LRECL"/> <!-- Retrieve LRECL of SCLM
        dataset -->
<!-- Reputil parms : project,projdef,group,repdgrp,type -->

        <xs:enumeration value="JOBSTAT"/> <!-- Retrieve status of batch
        job -->
<!-- Jobstat parms : project, jobfunc,jobname,jobid -->

        <xs:enumeration value="JARCOPY"/> <!-- Copies SCLM JAR into cpath
        directory -->
<!-- Jarcopy parms : project,projdef,group,repdgrp,type,classdir,jarname -->

</xs:restriction>

</xs:simpleType>
</xs:element>

<!-- List of common function parameters -->

<!-- project : SCLM project -->
<xs:element name="project" type="xs:string" minOccurs="0"/>
<!-- projdef : SCLM project definition -->
<xs:element name="projdef" type="xs:string" minOccurs="0"/>
<!-- group : SCLM group selected -->
<xs:element name="group" type="xs:string" minOccurs="0"/>
<!-- repdgrp : Users SCLM development group -->
<xs:element name="repdgrp" type="xs:string" minOccurs="0"/>
<!-- type : SCLM type -->
<xs:element name="type" type="xs:string" minOccurs="0"/>
<!-- member : SCLM member -->
<xs:element name="member" type="xs:string" minOccurs="0"/>
<!-- lang : SCLM language for the member -->
<xs:element name="lang" type="xs:string" minOccurs="0"/>
<!-- submit : submission type either online or batch -->
<xs:element name="submit" type="xs:string" minOccurs="0"/>

```

Abbildung 24. XML-Schema für SCLMDT-Befehle (Fortsetzung)

```

<!-- sclmfunc=build : additional parameter options -->
  <!-- bldscope : Build scope -->
  <xs:element name="bldscope" type="xs:string" minOccurs="0"/>
  <!-- bldmode : Build mode -->
  <xs:element name="bldmode" type="xs:string" minOccurs="0"/>
  <!-- bldlist : Translator listing only if error -->
  <xs:element name="bldlist" type="xs:string" minOccurs="0"/>
  <!-- bldrept : build report generated -->
  <xs:element name="bldrept" type="xs:string" minOccurs="0"/>
  <!-- bldmsg : build messages generated -->
  <xs:element name="bldmsg" type="xs:string" minOccurs="0"/>
  <!-- bldmsgds: build messages dataset name -->
  <xs:element name="bldmsgds" type="xs:string" minOccurs="0"/>
  <!-- bldrptds : build report dataset name -->
  <xs:element name="bldrptds" type="xs:string" minOccurs="0"/>
  <!-- bldlstds : build list dataset name -->
  <xs:element name="bldlstds" type="xs:string" minOccurs="0"/>
  <!-- bldextds : build exit dataset name -->
  <xs:element name="bldextds" type="xs:string" minOccurs="0"/>
  <!-- groupbld : SCLM group to build against -->
  <xs:element name="groupbld" type="xs:string" minOccurs="0"/>

  <!-- sclmfunc=promote : additional parameter options -->
  <!-- prmscope : Promote scope -->
  <xs:element name="prmscope" type="xs:string" minOccurs="0"/>
  <!-- prmmode : Promote mode -->
  <xs:element name="prmmode" type="xs:string" minOccurs="0"/>
  <!-- prmrept : Promote report generated -->
  <xs:element name="prmrept" type="xs:string" minOccurs="0"/>
  <!-- prmmsg : Promote messages generated -->
  <xs:element name="prmmsg" type="xs:string" minOccurs="0"/>
  <!-- prmmsgds: Promote messages dataset name -->
  <xs:element name="prmmsgds" type="xs:string" minOccurs="0"/>
  <!-- prm rptds : Promote report dataset name -->
  <xs:element name="prm rptds" type="xs:string" minOccurs="0"/>
  <!-- prmextds : Promote exit dataset name -->
  <xs:element name="prmextds" type="xs:string" minOccurs="0"/>
  <!-- groupprm : SCLM group to promote from -->
  <xs:element name="groupprm" type="xs:string" minOccurs="0"/>

  <!-- sclmfunc=delete : additional parameter option -->
  <!-- delflag : either text|acct|txbm|bmap|all -->
  <xs:element name="delflag" type="xs:string" minOccurs="0"/>

  <!-- sclmfunc=deploy : additional parameter options -->
  <!-- depmode : If set to 'R' then deploy report only -->
  <xs:element name="depmode" type="xs:string" minOccurs="0"/>
  <!-- depsec : set to Y if using surrogate userids -->
  <xs:element name="depsec" type="xs:string" minOccurs="0"/>
  <!-- groupdpy : SCLM group deploying from -->
  <xs:element name="groupdpy" type="xs:string" minOccurs="0"/>

```

Abbildung 25. XML-Schema für SCLMDT-Befehle (Fortsetzung)


```

<!-- sclmfunc=report : additional parameter options -->
  <!-- reptype : type|* - SCLM type to report on -->
  <xs:element name="reptype" type="xs:string" minOccurs="0"/>
  <!-- repmem : member|*|filter - SCLM member to report on -->
  <xs:element name="repmem" type="xs:string" minOccurs="0"/>
  <!-- replang : lang|* - SCLM language to filter with -->
  <xs:element name="replang" type="xs:string" minOccurs="0"/>
  <!-- repgrp : group|* - SCLM group to filter on -->
  <xs:element name="repgrp" type="xs:string" minOccurs="0"/>
  <!-- repccode : SCLM changecode to filter on -->
  <xs:element name="repccode" type="xs:string" minOccurs="0"/>
  <!-- repmode : D|E developer mode or explorer mode -->
  <xs:element name="repmode" type="xs:string" minOccurs="0"/>
  <!-- repbmap : If ON then report on build maps -->
  <xs:element name="repbmap" type="xs:string" minOccurs="0"/>

  </xs:all>
</xs:complexType>
</xs:element>

</xs:all>
</xs:complexType>
</xs:element>

</xs:schema>

```

Abbildung 26. XML-Schema für SCLMDT-Befehle (Fortsetzung)

Funktionen und Parameter anfordern

Funktionsformat

Das folgende Beispiel zeigt die Grundstruktur der XML-Eingabedatei, wobei #function die aufgerufene Funktion repräsentiert und #parameter und #value einen Parameter und dessen Wert repräsentieren.

```

<?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>

```

Abbildung 27. Grundstruktur der XML-Eingabedatei

Anmerkung: Wenn ein Parameter mehrfach angegeben ist, werden die Definitionen in der letzten Instanz verwendet.

Die folgende Liste enthält allgemeine Anmerkungen zu den Parametern und der Art und Weise, wie diese in der vorliegenden Veröffentlichung dokumentiert werden:

- Parameter sind nicht positionsgebunden.
- Eckige Klammern ([]) kennzeichnen einen optionalen Parameter für die Funktion.

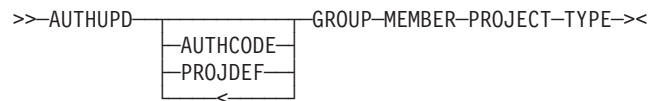
- Geschweifte Klammern ({}) kennzeichnen eine Liste gültiger Werte für den Parameter.
- Ein senkrechter Balken (|) trennt verschiedene Werte in einer Liste. Der unterstrichene Wert gibt, falls vorhanden, den Standardwert an.
- Schlüsselwörter in Großbuchstaben repräsentieren Konstanten, die wie angegeben codiert werden müssen. Schlüsselwörter in Kleinbuchstaben repräsentieren Platzhalter, die mit angepassten Werten ersetzt werden müssen.
- Jeder Parameter erfordert einen Wert. Es werden jedoch nur die Werte dokumentiert, die zu einer Liste gehören.
- Referenzen auf WORKAREA beziehen sich auf das Verzeichnis z/OS UNIX WORKAREA, das sich standardmäßig in /var/rdz/sc1mdt/ befindet.
- Verweise auf member erfordern die Verwendung des kurzen (SCLM-)Membernamens, sofern nicht explizit abweichend angegeben.

Funktionsliste

- „AUTHUPD – SCLM-Berechtigungscode ändern“
- „BROWSE – SCLM-Member durchsuchen“ auf Seite 79
- „BUILD – SCLM-Member erstellen“ auf Seite 79
- „DELETE – SCLM-Member löschen“ auf Seite 81
- „DEPLOY – J2EE-EAR-Datei implementieren“ auf Seite 81
- „EDIT – SCLM-Member bearbeiten“ auf Seite 82
- „INFO – Statusinformationen zum SCLM-Member“ auf Seite 83
- „J2EEIMP – Projekt aus SCLM importieren“ auf Seite 83
- „J2EEMIG – Projekt in SCLM migrieren“ auf Seite 85
- „J2EEMIGB – Projekt als Batch in SCLM migrieren“ auf Seite 86
- „JARCOPY – JAR-Datei kopieren“ auf Seite 87
- „JOBSTAT – Status des Batch-Jobs abrufen“ auf Seite 87
- „LRECL – LRECL aus SCLM-Dateigruppe abrufen“ auf Seite 88
- „MIGDSN – Nicht-SCLM-Dateigruppen und -Member auflisten“ auf Seite 88
- „MIGPDS – Nicht-SCLM-Dateigruppen und -Member in SCLM migrieren“ auf Seite 88
- „PROJGRPS – SCLM-Gruppen für ein Projekt abrufen“ auf Seite 89
- „PROJINFO – SCLM-Projektinformationen abrufen“ auf Seite 89
- „PROMOTE – SCLM-Member umstufen“ auf Seite 89
- „REPORT – Projektbericht erstellen“ auf Seite 91
- „REPUTIL – SCLM-DBUTIL-Bericht“ auf Seite 92
- „SAVE – SCLM-Member speichern“ auf Seite 92
- „UNLOCK – SCLM-Member freigeben“ auf Seite 93
- „UPDATE – SCLM-Memberinformationen aktualisieren“ auf Seite 93
- „VERBROW – SCLM-Versionen anzeigen“ auf Seite 94
- „VERDEL – SCLM-Versionen löschen“ auf Seite 94
- „VERHIST – SCLM-Versionsprotokoll“ auf Seite 95
- „VERLIST – SCLM-Versionen auflisten“ auf Seite 95
- „VERREC – SCLM-Versionen wiederherstellen“ auf Seite 96

AUTHUPD – SCLM-Berechtigungscode ändern

Diese Funktion ändert den Berechtigungscode eines Members.



Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

MEMBER

SCLM-Member - Ausgewähltes SCLM-Member.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

[AUTHCODE]

SCLM-Berechtigungscode - Neuer Berechtigungscode, der dem Member zugewiesen werden soll.

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

BROWSE – SCLM-Member durchsuchen

Diese Funktion kopiert ein Member aus SCLM in das z/OS UNIX-Verzeichnis WORKAREA/userid/EDIT/. Es wird keine Bearbeitungssperre in SCLM ausgeführt.



Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

MEMBER

SCLM-Member - Ausgewähltes SCLM-Member.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

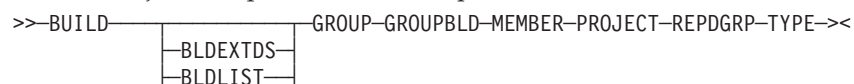
Optionale Parameter:

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

BUILD – SCLM-Member erstellen

Diese Funktion weist SCLM an, Softwarekomponenten den Architekturdefinitionen für das Projekt entsprechend zu kompilieren, zu verlinken und zu integrieren.



BLDLSTDS
BLDMODE
BLDMSGDS
BLDREPT
BLDRPTDS
BLDScope
PROJDEF
SUBMIT
<

Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

GROUPBLD

SCLM-Gruppe - Die ausgewählte SCLM-Gruppe, in der das erforderliche Member erstellt werden soll.

MEMBER

SCLM-Member - Ausgewähltes SCLM-Member.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

REPDGRP

SCLM-Entwicklungsgruppe - Die Entwicklungsgruppe des Benutzers.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

[BLDEXTDS [dsn | NONE]]

Build-Exit-Datei - NONE oder der Name der Datei, die die Build-Exit-Ausgabe enthält, wenn ein Build-Exit verwendet wird. Wenn die Dateigruppe nicht vorhanden ist, wird eine neue Dateigruppe zugeordnet. Die Standardeinstellung ist NONE.

[BLDLIST [Y | N]]

Bilddauflistung - Gibt an, ob die Buildumsetzerlisten nur bei Fehlern in die Listendatei kopiert werden sollen. Die Standardeinstellung ist Y.

[BLDLSTDS [dsn | NONE]]

Buildlistendatei - NONE oder der Name der Datei, die die Buildlisten enthält. Wenn die Dateigruppe nicht vorhanden ist, wird eine neue Dateigruppe zugeordnet. Die Buildlisten werden außerdem in der XML-Antwortdatei zurückgegeben, unabhängig von dieser Einstellung. Die Standardeinstellung ist NONE.

[BLDMODE [C | F | R | U]]

Erstellungsmodus - Gibt den Erstellungsmodus an (C=conditional, F=forced, R=report, U=unconditional). Die Standardeinstellung ist C.

[BLDMSGDS [dsn | NONE]]

Build-Nachrichtendatei - NONE oder der Name der Datei, die die Erstellungsnachrichten enthält. Wenn die Dateigruppe nicht vorhanden ist, wird eine neue Dateigruppe zugeordnet. Die Erstellungsnachrichten werden außerdem in der XML-Antwortdatei zurückgegeben, unabhängig von dieser Einstellung. Die Standardeinstellung ist NONE.

[BLDREPT [Y | N]]

Erstellungsbericht - Gibt an, ob ein Erstellungsbericht erstellt werden soll. Die Standardeinstellung ist Y.

[BLDRPTDS [dsn | NONE]]

Erstellungsberichtdatei - NONE oder der Name der Datei, die den Erstellungsbericht enthält. Wenn die Dateigruppe nicht vorhanden ist, wird eine neue Dateigruppe zugeordnet. Der Erstellungsbericht wird außerdem in der XML-Antwortdatei zurückgegeben, unabhängig von dieser Einstellung. Die Standardeinstellung ist NONE.

[BLDSCOPE [E | L | N | S]]

Eingliederungsbereich - Gibt den Eingliederungsbereich an (E=extended, L=limited, N=normal, S=subunit). Die Standardeinstellung ist N.

[PROJDEF]

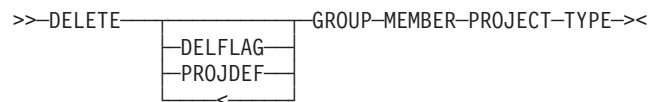
Alternativer SCLM-Projektnamen - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

[SUBMIT [BATCH | ONLINE]]

Übergabemethode - Der Build wird entweder online oder als Batch übergeben. Die Standardeinstellung ist ONLINE.

DELETE – SCLM-Member löschen

Diese Funktion löscht ein SCLM-Member.



Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

MEMBER

SCLM-Member - Ausgewähltes SCLM-Member.

PROJECT

SCLM-Projektnamen - Der Name des SCLM-Projekts.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

[DELFLAG [TEXT | ACCT | TXBM | BMAP | ALL]]

Attribut löschen - Gibt an, dass entweder Text, Account, Buildzuordnung, eine Kombination von Buildzuordnung und Text oder alle Attribute für ein bestimmtes Member gelöscht werden sollen. Die Standardeinstellung ist ALL.

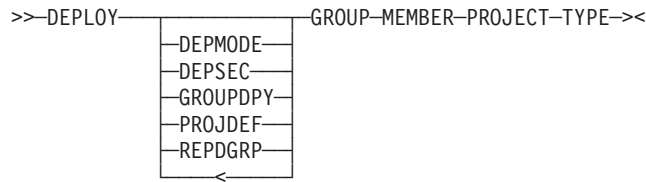
[PROJDEF]

Alternativer SCLM-Projektnamen - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

DEPLOY – J2EE-EAR-Datei implementieren

Die DEPLOY-Funktion führt das Implementierungsscript aus, das durch das Member referenziert wird, um eine J2EE-Unternehmensarchivdatei (EAR) aus USS oder SCLM auf einem Websphere Application Server (WAS) zu implementieren. Weitere

Informationen zum Erstellen eines Implementierungsscriptmembers finden Sie im Benutzerhandbuch zu SCLM Developer Toolkit.



Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

MEMBER

SCLM-Member - Ausgewähltes SCLM-Member.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

[DEPSEC {Y | N}]

Sichere Implementierung - Markieren Sie diese Einstellung, um anzugeben, ob eine Überprüfung der Sicherheitsregeln und ein möglicher Ersatzbenutzer-ID-Wechsel für diese Implementierung ausgeführt werden soll.

[DEPMODE {R}]

Implementierungsmodus - Wenn auf 'R' gesetzt, nur Berichtmodus. Es wird keine Implementierung ausgeführt.

[GROUPDPY]

Implementierungsgruppe - Die SCLM-Gruppe, aus der die EAR-Datei implementiert wird (wenn nicht gefunden, wird die Gruppenshierarchie durchsucht). Wenn auf 'USS' gesetzt, wird die EAR-Datei direkt aus dem z/OS UNIX-Verzeichnis implementiert, das als Teil des EAR-Namens in der Variablen EAR_FILE_NAME angegeben ist. Diese Variable wird im Implementierungsscript-Member als MEMBER referenziert.

[PROJDEF]

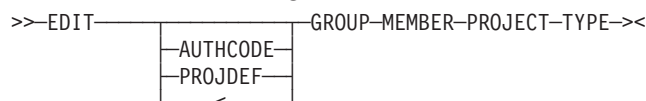
Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

[REPDGRP]

SCLM-Entwicklungsgruppe - Die Entwicklungsgruppe des Benutzers.

EDIT – SCLM-Member bearbeiten

Diese Funktion kopiert ein Member aus SCLM in das z/OS UNIX-Verzeichnis WORKAREA/userid/EDIT/. Außerdem wird eine SCLM-Sperre für das Member mit der Benutzer-ID als Zugriffsschlüssel erstellt.



Erforderliche Parameter:

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

SCLM-Member - Ausgewähltes SCLM-Member.

SCLM-Projektname - Der Name des SCLM-Projekts.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

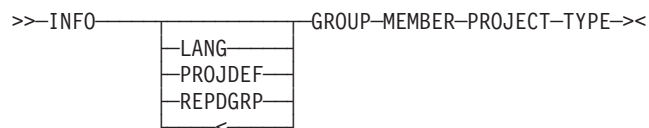
Optionale Parameter:

SCLM-Berechtigungscode - Neuer Berechtigungscode, der dem Member zugewiesen werden soll.

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

INFO – Statusinformationen zum SCLM-Member

Diese Funktion stellt Informationen zum Status des SCLM-Members bereit.



Erforderliche Parameter:

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

SCLM-Member - Ausgewähltes SCLM-Member.

SCLM-Projektname - Der Name des SCLM-Projekts.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

SCLM-Sprache - Die SCLM-Sprache des ausgewählten Members.

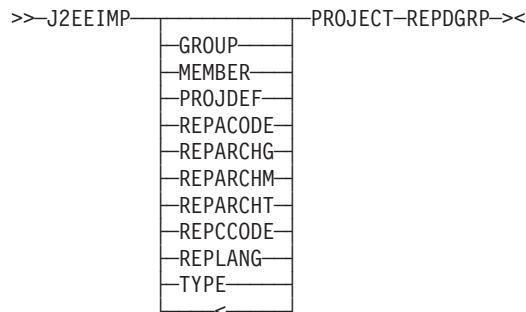
Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

SCLM-Entwicklungsgruppe - Die Entwicklungsgruppe des Benutzers.

J2EEIMP – Projekt aus SCLM importieren

Diese Funktion importiert ein Projekt im JAR-Format (komprimiert) aus SCLM in das z/OS UNIX /var/rdz/WORKAREA/userid-Verzeichnis /var/rdz/WORKAREA/

userid. Die JAR-Projektdatei kann anschließend auf den Client kopiert werden. Der Name der Projektdatei wird im Schlüsselwort J2EEFILE der (XML-)Funktionsausgabe zurückgegeben.



Erforderliche Parameter:

[PROJECT]

SCLM-Projektname - Der Name des SCLM-Projekts.

[REPDGRP]

SCLM-Entwicklungsgruppe - Die Entwicklungsgruppe des Benutzers.

Optionale Parameter:

[GROUP {group | group* | *}]

Gruppenhierarchie - Die Gruppenhierarchie für die Memberauswahl. Falls REPARCHM definiert ist, befinden sich die zugehörigen ARCHDEFs in group*.

[MEMBER {member | member* | *}]

SCLM-Member - Ausgewählte SCLM-Member.

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

[REPACODE]

Berechtigungscode - Berechtigungscode für die Filterung.

[REPARCHG]

ARCHDEF-Gruppe - Die SCLM-Gruppe, in der sich die ARCHDEF-Member befinden.

[REPARCHM]

ARCHDEF-Member - Der Name der ARCHDEF-Member, die für den Import ausgewählt werden sollen.

[REPARCHT]

ARCHDEF-Typ - Der SCLM-Typ, in dem sich das ARCHDEF-Member befindet.

[REPCODE]

Änderungscode - Änderungscode für Filterung.

[REPLANG]

SCLM-Sprache - Sprache für Filterung.

[TYPE {type | type* | *}]

SCLM-Typ - SCLM-Typ(en) für die Memberauswahl.

J2EEMIG – Projekt in SCLM migrieren

Diese Funktion nimmt eine komprimierte Datei (JAR-Format) aus dem USS-Verzeichnis, extrahiert diese und migriert alle darin enthaltenen Member in SCLM, wobei ausgeschriebene Namen bei Bedarf in Kurznamen umgesetzt werden.

```
>>-J2EEMIG-          GROUP-J2EEFILE-LANG-MEMBER-PROJECT-TYPE-><
|
|  ARCHAC
|  ARCHCC
|  ARCHTYPE
|  AUTHCODE
|  CCODE
|  J2EESINC
|  J2EETYPE
|  MIGMODE
|  PROJARCH
|  PROJDEF
|  SCLMREFS
|  SUBMIT
|
|  <
```

Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

J2EEFILE

Eingabedateiname - Der Dateiname des (gezippten) JAR-Projekts, das in SCLM importiert werden soll. Die JAR-Datei muss sich im Verzeichnis z/OS UNIX /var/rdz/WORKAREA/userid befinden.

LANG

SCLM-Sprache - Die SCLM-Sprache des ausgewählten Members.

MEMBER

SCLM-Member - Ausgewähltes SCLM-Member.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

[ARCHAC]

ARCHDEF-Berechtigungscode - ARCHDEF-Berechtigungscode festlegen.

[ARCHCC]

ARCHDEF-Änderungscode - ARCHDEF-Änderungscode festlegen.

[ARCHTYPE [ARCHDEF | archtype]]

ARCHDEF-SCLM-Typ - ARCHDEF-Typ festlegen.

[AUTHCODE]

Memberberechtigungscode - Berechtigungscode von ARCHDEF-Membren festlegen.

[CCODE]

Memberänderungscode - Änderungscode von ARCHDEF-Membren festlegen.

[J2EESINC]

J2EE-SINC-Erstellungsscript.- Name des Erstellungsscripts in TYPE J2EEBLD, auf das mit dem SINC-Schlüsselwort im ARCHDEF-Member verwiesen wird.

[J2EETYPE {JAR | WAR | EAR}]

J2EE-Typ - Geben Sie die JAR-, WAR- oder EAR-Datei für das J2EE-ARCHDEF-Projekt an.

[MIGMODE {FORCE}]

Migrationsmodus – Mit FORCE werden vorhandene Member ersetzt. Die Standardeinstellung sieht eine bedingte Migration vor.

[PROJARCH]

ARCHDEF-Projekt - Der Name der ARCHDEF, die aktualisiert oder erstellt werden soll.

[SCLMREFS]

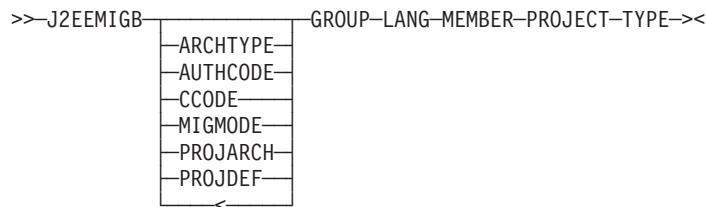
SCLM-Verweise - Weitere ARCHDEFs oder Teile.

[SUBMIT [BATCH | ONLINE]]

Übergabemethode - Die Migration wird entweder online oder als Batch übergeben. Die Standardeinstellung ist ONLINE.

J2EEMIGB – Projekt als Batch in SCLM migrieren

Mit dieser Funktion wird der BATCH-Job für die Migration eingerichtet und ausgeführt.



Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

LANG

SCLM-Sprache - Die SCLM-Sprache des ausgewählten Members.

MEMBER

SCLM-Member - Ausgewähltes SCLM-Member.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

[ARCHTYPE]

ARCHDEF-SCLM-Typ - ARCHDEF-Typ festlegen.

[AUTHCODE]

Memberberechtigungscode - Berechtigungscode von ARCHDEF-Memberrn festlegen.

[CCODE]

Memberänderungscode - Änderungscode von ARCHDEF-Memberr festlegen.

[MIGMODE {FORCE}]

Migrationsmodus – Mit FORCE werden vorhandene Member ersetzt. Die Standardeinstellung sieht eine bedingte Migration vor.

[PROJARCH]

ARCHDEF-Projekt - Der Name der ARCHDEF, die aktualisiert oder erstellt werden soll.

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

JARCOPY – JAR-Datei kopieren

Diese Funktion kopiert eine in SCLM gespeicherte JAR-Datei in ein z/OS UNIX-Verzeichnis, das als CLASSPATH-Verzeichnis verwendet werden kann.

```
>>JARCOPY—CLASSDIR—GROUP—JARNAME—PROJECT—REPDGRP—TYPE—><
```

Erforderliche Parameter:

CLASSDIR

Klassenpfadverzeichnis - Name des Ziel-z/OS UNIX-Verzeichnisses.

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

JARNAME

JAR-Dateiname - ausgeschriebener Name der Ziel-JAR-Datei. Der SCLM-Kurzname wird automatisch gesucht.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

REPDGRP

SCLM-Entwicklungsgruppe - Die Entwicklungsgruppe des Benutzers.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

JOBSTAT – Status des Batch-Jobs abrufen

Diese Funktion ruft den Status eines bestimmten Batch-Jobs ab.

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

Parameter:

JOBFUNC {JOBSTAT | JOBRETR}

Funktionsauswahl - Jobstatus (JOBSTAT) oder Jobausgabe (JOBRETR) abrufen.

JOBID

Job-ID - Jobnummer des Batch-Jobs.

JOBNAME

Jobname - Name des Batch-Jobs.

PROJECT

SCLM-Projektnamen - Der Name des SCLM-Projekts.

LRECL – LRECL aus SCLM-Dateigruppe abrufen

Diese Funktion ruft die Länge eines logischen Satzes einer SCLM-Dateigruppe ab.

```
>>-LRECL-_____GROUP-PROJECT-REPDGRP-TYPE-><
      |_____|
      | PROJDEF |
```

Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

PROJECT

SCLM-Projektnamen - Der Name des SCLM-Projekts.

REPDGRP

SCLM-Entwicklungsgruppe - Die Entwicklungsgruppe des Benutzers.

Optionale Parameter:

[PROJDEF]

Alternativer SCLM-Projektnamen - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

MIGDSN – Nicht-SCLM-Dateigruppen und -Member auflisten

Diese Funktion listet ausgewählte Dateigruppen und Member auf, die nicht in SCLM verwaltet werden (für anschließende Migration nach SCLM).

```
>>-MIGDSN-_____MIGDSN-MIGMEM-PROJECT-><
      |_____|
      | PROJDEF |
```

Erforderliche Parameter:

MIGDSN [dsn | *]

Dateifilter - Dateifilter für Liste. Die Standardeinstellung ist *.

MIGMEM [member | *]

Memberfilter - Memberfilter für Liste. Die Standardeinstellung ist *.

PROJECT

SCLM-Projektnamen - Der Name des SCLM-Projekts.

Optionale Parameter:

[PROJDEF]

Alternativer SCLM-Projektnamen - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

MIGPDS – Nicht-SCLM-Dateigruppen und -Member in SCLM migrieren

Diese Funktion migriert ausgewählte Dateigruppen und Member, die nicht SCLM-verwaltet sind, in SCLM.

```
>>-MIGPDS-_____GROUP-PROJECT-TYPE-><
      |_____|
      | PROJDEF |
```

Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

PROJGRPS – SCLM-Gruppen für ein Projekt abrufen

Diese Funktion ruft die SCLM-Gruppen für ein ausgewähltes Projekt ab.

>>-PROJGRPS-PROJECT-><
└-PROJDEF-

Erforderliche Parameter:

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

Optionale Parameter:

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

PROJINFO – SCLM-Projektinformationen abrufen

Diese Funktion ruft die SCLM-Projektinformationen für ein ausgewähltes Projekt ab.

>>-PROJINFO-PROJECT-REPDGRP-><
└-PROJDEF-

Erforderliche Parameter:

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

REPDGRP

SCLM-Entwicklungsgruppe - Die Entwicklungsgruppe des Benutzers.

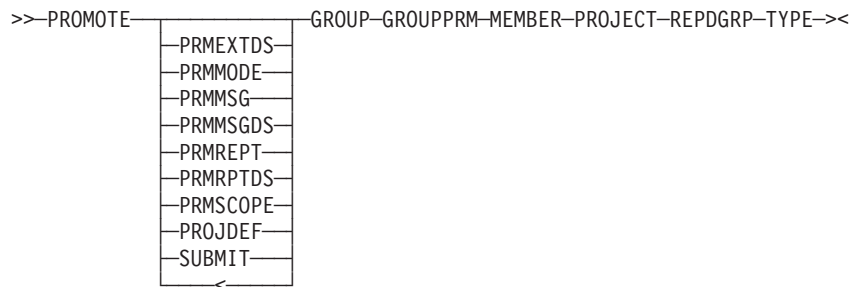
Optionale Parameter:

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

PROMOTE – SCLM-Member umstufen

Diese Funktion stuft ein SCLM-Member (oder eine ARCHDEF) in der SCLM-Gruppenhierarchie entsprechend der Architekturdefinition des Projekts und der Projektdefinition um.



Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

GROUPPRM

SCLM-Gruppe - Die ausgewählte SCLM-Gruppe, aus der das erforderliche Member umgestuft werden soll.

MEMBER

SCLM-Member - Ausgewähltes SCLM-Member.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

REPDGRP

SCLM-Entwicklungsgruppe - Die Entwicklungsgruppe des Benutzers.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

[PRMEXTDS [dsn | NONE]]

Umstufungs-Exit-Datei - NONE oder der Dateiname, unter dem die Umstufungs-Exit-Ausgabe gespeichert ist, falls ein Umstufungs-Exit verwendet wird. Wenn die Dateigruppe nicht vorhanden ist, wird eine neue Dateigruppe zugeordnet. Die Standardeinstellung ist NONE.

[PRMMODE [C | R | U]]

Umstufungsmodus - Gibt den Umstufungsmodus an (C=conditional, R=report, U=unconditional). Die Standardeinstellung ist C.

[PRMMSG [Y | N]]

Umstufungsnachrichten - Auf „Y“ setzen, um Umstufungsnachrichten einzuschließen. Die Standardeinstellung ist N.

[PRMMSGDS [dsn | NONE]]

Umstufungsnachrichtendatei - NONE oder der Dateiname, unter dem die Umstufungsnachrichten gespeichert werden. Wenn die Dateigruppe nicht vorhanden ist, wird eine neue Dateigruppe zugeordnet. Die Umstufungsnachrichten werden ebenfalls in der XML-Antwortdatei zurückgegeben, falls für PRMMSG der Wert Y angegeben ist. Der Standardwert lautet NONE.

[PRMREPT [Y | N]]

Umstufungsbericht - Auf „Y“ setzen, um den Umstufungsbericht einzuschließen. Die Standardeinstellung ist N.

[PRMRPTDS [dsn | NONE]]

Umstufungsberichtdatei - NONE oder der Name der Datei, unter dem der Umstufungsbericht gespeichert wird. Wenn die Dateigruppe nicht vorhanden ist, wird eine neue Dateigruppe zugeordnet. Die Umstufungsnachricht-

ten werden ebenfalls in der XML-Antwortdatei zurückgegeben, falls für PRMSG der Wert Y angegeben ist. Der Standardwert lautet NONE.

[PRMSCOPE [E | N | S]]

Umstufungsbereich - Gibt den Umstufungsbereich an (E=extended, N=normal, S=subunit). Die Standardeinstellung ist N.

[PROJDEF]

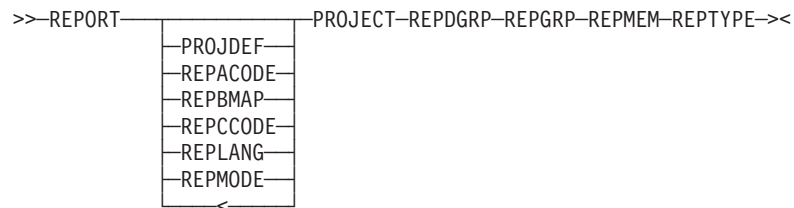
Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

[SUBMIT [BATCH | ONLINE]]

Übergabemethode - Die Umstufung wird entweder online oder als Batch übergeben. Die Standardeinstellung ist ONLINE.

REPORT – Projektbericht erstellen

Die Funktion zur Berichterstellung stellt einen DBUTIL-Hierarchiebericht für das Projekt bereit, entsprechend den verwendeten Berichtparametern und Filtern.



Erforderliche Parameter:

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

REPDGRP

SCLM-Entwicklungsgruppe - Die Entwicklungsgruppe des Benutzers.

REPGRP {group | group* | * }

SCLM-Gruppe - SCLM-Gruppen für die Filterung.

REPMEM {member | mem* | * }

SCLM-Member - SCLM-Member für die Filterung.

REPTYPE {type | type* | * }

SCLM-Typ - SCLM-Typen für die Filterung.

Optionale Parameter:

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

[REPACODE]

Berechtigungscode - Berechtigungscode für die Filterung.

[REPBMAP [Y | N]]

Build-Maps - Attribut zum Einschließen von Build-Maps im DBUTIL-Bericht. Die Standardeinstellung ist N.

[REPCODE]

Änderungscode - Änderungscode für Filterung.

[REPLANG]


Sprache - SCLM-Sprachtyp für Filterung.

[REPMODE [D | E]]

Modus - Entwicklermodus (D) oder Explorermodus (E) für den Bericht.
Die Standardeinstellung ist D.

REPUTIL – SCLM-DBUTIL-Bericht

Der DBUTIL-Service ruft Informationen aus der Projektdatenbank ab und erstellt eine angepasste Ausgabe sowie einen Bericht. Dieser beschreibt den Inhalt der Projektdatenbank basierend auf den Auswahlkriterien, die Sie angeben.

>>REPUTIL  GROUP-MEMBER-PROJECT-REPDGRP-TYPE-><

Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

MEMBER

SCLM-Member - Ausgewähltes SCLM-Member.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

REPDGRP

SCLM-Entwicklungsgruppe - Die Entwicklungsgruppe des Benutzers.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

SAVE – SCLM-Member speichern

Diese Funktion kopiert ein Member aus dem Verzeichnis WORKAREA/userid/EDIT/ in die Entwicklungsgruppe eines Benutzers in SCLM. Ein neues Member wird erstellt, wenn das Member nicht in der SCLM-Projekthierarchie vorhanden ist.

>>SAVE  GROUP-MEMBER-PROJECT-TYPE-><

Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

MEMBER

SCLM-Member - Ausgewähltes SCLM-Member.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

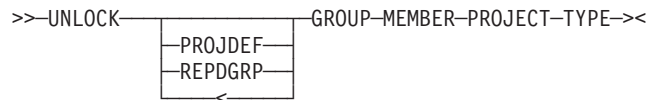
Optionale Parameter:

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

UNLOCK – SCLM-Member freigeben

Diese Funktion gibt ein SCLM-Member frei, das mit der EDIT-Funktion gesperrt wurde.



Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

MEMBER

SCLM-Member - Ausgewähltes SCLM-Member.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

[PROJDEF]

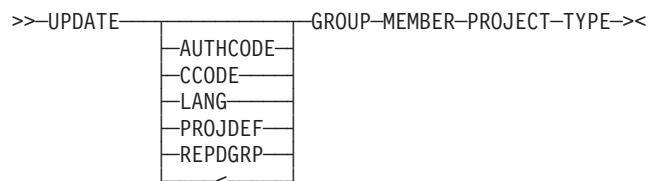
Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

[REPDGRP]

SCLM-Entwicklungsgruppe - Die Entwicklungsgruppe des Benutzers.

UPDATE – SCLM-Memberinformationen aktualisieren

Diese Funktion aktualisiert die Memberinformationen in SCLM.



Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

MEMBER

SCLM-Member - Ausgewähltes SCLM-Member.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

[AUTHCODE]

Berechtigungscode - Berechtigungscode für die Aktualisierung des Members.

[CCODE]

Änderungscode - Änderungscode zum Aktualisieren des Members.

[LANG]

SCLM-Sprache - Die SCLM-Sprache des ausgewählten Members.

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

[REPDGRP]

SCLM-Entwicklungsgruppe - Die Entwicklungsgruppe des Benutzers.

VERBROW – SCLM-Versionen anzeigen

Diese Funktion zeigt eine Version eines Members aus der Dateigruppe der Versionen an.

```
>>-VERBROW-PROJDEF-GROUP-MEMBER-PROJECT-REPDGRP-TYPE-><
```

Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

MEMBER

SCLM-Member - Ausgewähltes SCLM-Member.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

REPDGRP

SCLM-Entwicklungsgruppe - Die Entwicklungsgruppe des Benutzers.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

VERDEL – SCLM-Versionen löschen

Diese Funktion löscht die Informationen über ein versionsgesteuertes oder geprüftes Member aus SCLM.

```
>>-VERDEL-PROJDEF-GROUP-MEMBER-PROJECT-REPDGRP-TYPE-><
```

Diese Funktion löscht alle älteren Versionen eines Members in SCLM.

Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

MEMBER

SCLM-Member - Ausgewähltes SCLM-Member.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

REPDGRP

SCLM-Entwicklungsgruppe - Die Entwicklungsgruppe des Benutzers.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

VERHIST – SCLM-Versionenprotokoll

Diese Funktion zeigt das Versionsprotokoll einer ausgewählten Memberversion in SCLM.

```
>>-VERHIST-  
      |  
      |_____|  
      | PROJDEF |  
      |_____|  
      GROUP-MEMBER-PROJECT-REPDGRP-TYPE-><
```

Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

MEMBER

SCLM-Member - Ausgewähltes SCLM-Member.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

REPDGRP

SCLM-Entwicklungsgruppe - Die Entwicklungsgruppe des Benutzers.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

VERLIST – SCLM-Versionen auflisten

Diese Funktion listet die verschiedenen Versionen für ein bestimmtes Member auf.

```
>>-VERLIST-  
      |  
      |_____|  
      | PROJDEF |  
      |_____|  
      GROUP-MEMBER-PROJECT-REPDGRP-TYPE-><
```

Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

MEMBER

SCLM-Member - Ausgewähltes SCLM-Member.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

REPDGRP

SCLM-Entwicklungsgruppe - Die Entwicklungsgruppe des Benutzers.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

VERREC – SCLM-Versionen wiederherstellen

Diese Funktion stellt eine ausgewählte Version eines Members in der Entwicklungsgruppe wieder her.

```
>>-VERREC-  
      |  
      | PROJDEF |  
      |  
      | GROUP-MEMBER-PROJECT-REPDGRP-TYPE-><
```

Erforderliche Parameter:

GROUP

SCLM-Gruppe - Die SCLM-Gruppe, in der sich das ausgewählte Member befindet.

MEMBER

SCLM-Member - Ausgewähltes SCLM-Member.

PROJECT

SCLM-Projektname - Der Name des SCLM-Projekts.

REPDGRP

SCLM-Entwicklungsgruppe - Die Entwicklungsgruppe des Benutzers.

TYPE SCLM-Typ - Der SCLM-Typ, der das ausgewählte Member enthält.

Optionale Parameter:

[PROJDEF]

Alternativer SCLM-Projektname - Projektdefinitionsname (standardmäßig wird Projekt verwendet).

Beispiel

Ein Beispiel-Java-Programm (mit Eingabe und Ausgabe) wird bereitgestellt. Dabei wird die SCLMDT-Host-Services-API mit einem HTTP-Server als Transportmechanismus aufgerufen.

sclmdt_request.xml

Die Beispielanforderung im folgenden Beispiel ruft die BUILD-Funktion auf, um das ALLOEXT-Member im SCLMVCM-Projekt zu kompilieren und zu verknüpfen.

```

<?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>

```

Abbildung 28. *sclmdt_request.xml* – Beispiel-XML-Befehl-Eingabedatei

xmlbld.java

Das folgende Beispiel zeigt ein Beispiel-Java-Programm (in Eclipse) unter Verwendung der oben angegebenen XML-Eingabeanforderung. Bei der Ausführung stellt dieses Programm eine URL-Verbindung mit dem HTTP-Server unter z/OS (CDF-MVS08) her, wobei der XML-Eingabedatenstrom als POST-Anforderung an die BWBXML-CGI-Routine übergeben wird.

```

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" );
    }
}

```

Abbildung 29. xmlbld.java – Beispiel-Java-Programm

```

        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>" );
out.println( "</SCLMDT-INPUT>" );
out.flush();
out.close();
}

public static void doGet(URL url, HttpURLConnection con) throws IOException
{
    BufferedReader in;
    try
    {
        System.out.println("About to accept response from Server");
        in = new BufferedReader(new InputStreamReader(con.getInputStream()));
        System.out.println("Response from Server received");
    }
    catch (FileNotFoundException exception)
    {
        InputStream err = ((HttpURLConnection)con).getErrorStream();
        if (err == null) throw exception ;
        in = new BufferedReader(new InputStreamReader(err));
    }

    String line;
    while ((line = in.readLine()) != null)
        System.out.println(line);

    in.close();
}
}

```

Abbildung 30. *xmlbld.java* – Beispiel-Java-Programm (Fortsetzung)

Anmerkung:

1. Abhängig von Base64Encoder.class für Benutzer-ID-Verschlüsselung.
2. Die Datei C:\logon.txt enthält USERID:PASSWORD.

sclmdt_response.xml

Das folgende Beispiel zeigt die Beispielausgabe der BUILD-Funktion, die mit dem Beispiel-Java-Programm aufgerufen wurde.

Abb. 31 zeigt die Beispielausgabe der BUILD-Funktion, die mit dem Beispiel-Java-Programm aufgerufen wurde.

Abbildung 31. *sclmdt_response.xml* – Beispiel-XML-Ausgabedatei

```

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
      MEMBER      TYPE      VERSION      (REASON FOR REBUILD)
      -----

```



```

          ***** 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

MEMBER      TYPE      VERSION      (REASON FOR DELETE)
-----
***** 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 /BWFXML
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
Timecheck1sa: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 PARS = 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_TRANSTABLE=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 VCMTMP
Allocating data set SCLMVCM.ISPF.VCMISPF.ID075178 to the VCMTMP DD
1024 bytes of ISPF SELECT CMD(BWBBLD ID35577 SCLMVCM.IBMUSER SCLMVCM
SCLMVCM TEST DEV1 SOURCE ALLOCEXT /) NEST LANG(CREX)
written to VCMTMP
1024 bytes of /etc/rdz;/var/rdz written to VCMTMP
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 VCMTMP
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: isplib=isp.sispmenu
Allocation successful for ISPLIB
REC: isptlib=isp.sisptenu
Allocation successful for ISPTLIB
REC: isplib=isp.sisppenu
Allocation successful for ISPLIB
REC: ispslib=bzz.sbzzenus,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
<ISPF>
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 FLMSGSGS alloc is 0
PARMS=SCLMVCM,SCLMVCM,DEV1,SOURCE,ALLOCEXT,,N,C,Y,Y,,tmpmsg,
    tmpcpt,tmp1st,BLDEXIT
BWBBLD CHECK: PARMS = SCLMVCM,SCLMVCM,DEV1,SOURCE,ALLOCEXT,,
    N,C,Y,Y,,tmpmsg,tmpcpt,tmp1st,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
</ISPFINFO>
ISPRC = 0

```

```

***** ISPLLOG 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,tmpcpt,tmp1st,BLD
09:53 EXIT
09:53 End of ISPF Log - - - Session # 1 -----

```

```

***** END ISPLLOG 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
```

Anhang D. Rational Application Developer for WebSphere Software - Erstellungsdienstprogramm

Übersicht über das Erstellungsdienstprogramm von Rational Application Developer for WebSphere Software

In diesem Abschnitt werden die Erweiterungen des Java/J2EE-Erstellungsprozesses unter Verwendung des Erstellungsdienstprogramms von Rational Application Developer for WebSphere Software (früher AST: Application Server Toolkit) beschrieben. Das Erstellungsdienstprogramm von Rational Application Developer for WebSphere Software wird als Teil von Rational Application Developer (RAD) bereitgestellt. RAD muss separat angefordert, installiert und konfiguriert werden. Die Installation und Anpassung dieses Produkts wird in diesem Handbuch nicht beschrieben. Verwenden Sie die im Lieferumfang von RAD enthaltene Dokumentation, wenn Sie Anweisungen zur Installation und Anpassung der Funktionen des Erstellungsdienstprogramms benötigen. In diesem Abschnitt wird im Folgenden das Erstellungsdienstprogramm von Rational Application Developer for WebSphere als "Erstellungsdienstprogramm" bezeichnet.

Mit dem Erstellungsdienstprogramm ist es möglich, replizierte Projektarbeitsbereiche aus der IDE, die in SCLM gespeichert wurden, unter Verwendung des automatischen Modus von Eclipse unter z/OS mit SCLM zu erstellen.

Dieser Abschnitt sollte in Verbindung mit dem Onlinebenutzerhandbuch für SCLM Developer Toolkit und dem im Lieferumfang des Produkts enthaltenen Installations- und Anpassungshandbuch verwendet werden. SCLM Developer Toolkit stellt Java/J2EE-Sprachumsetzer für SCLM bereit, um vollständige Java/J2EE-Erstellungsfunktionen zu ermöglichen.

SCLM Developer Toolkit bietet nicht nur Funktionen zum Speichern von JAVA/J2EE-Quellcode und -Objekten, sondern durch diese Umsetzer auch die Möglichkeit, in SCLM Java/J2EE-Anwendungen zu erstellen und vollständig zu verwalten. Diese Sprachumsetzer unterstützen folgende Java/J2EE-Objekte: Java-Klassen, Java-Archivdateien (JAR), Enterprise Java Beans (EJB-JAR-Dateien), Webarchivdateien (WAR) und Unternehmensarchivdateien (EAR).

SCLM Developer Toolkit wird im Erstellungsdienstprogramm integriert, um die Replikation von SCLM-J2EE-Projekten im Arbeitsbereich des Erstellungsdienstprogramms unter UNIX System Services unter z/OS zu steuern. Die replizierten Projekte werden anschließend durch ARCHDEF-Erstellungsprozesse erstellt, die Erstellungsdienstprogrammsscripts referenzieren. Beispielerstellungsscripts werden in SCLM Developer Toolkit bereitgestellt. Objekte, die bei der Builderstellung/Kompilierung entstehen, werden in SCLM gespeichert.

Das Erstellungsdienstprogramm unterstützt Builds aller J2EE-Projekte, die unter Rational Application Developer V7 unterstützt werden. Projekte, die in früheren Versionen von RAD erstellt wurden, müssen in einen RAD V7-Arbeitsbereich auf dem IDE-Client migriert werden, bevor sie zu SCLM hinzugefügt werden können. Normale Eclipse-Java-Projekte werden ebenfalls unterstützt.

Speichern von Java/J2EE-Objekten in SCLM

Java/J2EE-Objekte werden in SCLM wie folgt gespeichert:

- Java-Quellcode wird in Klassen kompiliert. Klassen werden in SCLM als Typ JAVACLAS gespeichert. Der Kurzname und der ausgeschriebene Name werden in Umsetztabelle gespeichert.
- Unterstützung von EJB und regulären JAR-Dateien (enthält Klassen und möglicherweise weitere Java-Projektkomponenten wie XML/HTML/JSP in paketierter Struktur). JAR-Dateien werden im SCLM-Typ J2EEJAR gespeichert.
- Unterstützung von WAR-Dateien, die basierend auf J2EE-Dateien „Web.xml“ in J2EE-Projekten assembliert wurden. WAR-Dateien werden im SCLM-Typ J2EEWAR gespeichert.
- Unterstützung von EAR-Dateien, die für die Implementierung basierend auf application.xml in J2EE-Projekten erstellt wurden. EAR-Dateien werden im SCLM-Typ J2EEEAR gespeichert.
- Implementierungsunterstützung von EAR-Dateien in Websphere Application Server (WAS) unter z/OS.
- SQLJ-Buildunterstützung
- Alle Ausgabelisten werden im SCLM-Typ J2EELIST gespeichert.

Das Erstellungsdiensprogramm im Vergleich mit dem nativen ANT-Erstellungsprozess

SCLM Developer Toolkit umfasst einen nativen ANT-Erstellungsprozess für Java/J2EE-Unterstützung. Weitere Informationen hierzu finden Sie im SCLM Developer Toolkit-Onlinebenutzerhandbuch-Plug-in und Installations-/Anpassungshandbuch.

Das Erstellungsdiensprogramm für Rational Application Developer for WebSphere Software erweitert die Java/J2EE-Buildunterstützung durch eine vollständige Nachbildung der Eclipse IDE-Umgebung unter z/OS bei der Erstellung.

Der Prozess des Erstellungsdiensprogramms verwendet wie auch der ANT-Erstellungsprozess die ARCHDEF, die Member enthält, die das Java/J2EE-Projekt ausmachen, und stellt mithilfe des Kurznamens die Art und Weise des Vorhandenseins des Projekts in einem Eclipse-Arbeitsbereich dar.

Der Benutzer kann entweder den Erstellungsdiensprogramm-Prozess oder den nativen ANT-Erstellungsprozess auswählen, indem das Erstellungsscript (referenziert durch die ARCHDEF) mit dem entsprechenden Sprachumsetzer zugewiesen wird:

- J2EEAST für das Erstellungsdiensprogramm
- J2EEANT für die native ANT-Erstellung

Die ARCHDEF beim Aufruf des während der Buildvorbereitung definierten Sprachumsetzers (J2EEAST) durch den Build. Dieser Sprachentyp J2EEAST ist dem Erstellungsscript zugeordnet, auf das in der ARCHDEF mithilfe des Schlüsselworts SINC verwiesen wird. Dieser Sprachumsetzer kopiert alle ARCHDEF-Komponenten in das Dateisystem von z/OS UNIX System Services, das während der Verwendung des Erstellungsdiensprogramms erstellt werden soll.

Projektkomponenten und Java-Quellcode können in SCLM als EBCDIC oder ASCII gespeichert werden. Im Erstellungsdiensprogramm werden Textkomponenten und Java-Quellcode standardmäßig in ASCII im USS-Arbeitsbereich umgesetzt, bevor der Erstellungsprozess ausgeführt wird.

Installationshinweis zum Erstellungsdienstprogramm von Rational Application Developer for WebSphere Software

Das Erstellungsdienstprogramm von Rational Application Developer for WebSphere Software wird als Teil von Rational Application Developer (RAD) bereitgestellt. RAD muss separat angefordert, installiert und konfiguriert werden. Die Installation und Anpassung dieses Produkts wird in diesem Handbuch nicht beschrieben. Verwenden Sie die im Lieferumfang von RAD enthaltene Dokumentation, wenn Sie Anweisungen zur Installation und Anpassung der Funktionen des Erstellungsdienstprogramms benötigen. Das Verzeichnispaket des Erstellungsdienstprogramms wird im z/OS UNIX-Dateisystem gespeichert.

Für den J2EE-Erstellungsprozess können umfangreiche Regionsgrößen erforderlich sein, um Fehler im Zusammenhang mit unzureichendem Speicherplatz zu vermeiden. Um große Online-Builds zu versorgen, geben Sie mindestens REGION=512M im RSE-Dämon von Developer for System z an. (Standardmäßig ist dies die von RSED gestartete Task). Für die Stapelverarbeitung sollte dieser Parameter für die Regionsgröße im Stapel JOBCARD angegeben werden.

Kombinierter Einsatz von SCLM und Erstellungsdienstprogramm

Die ursprüngliche J2EEAST-Verarbeitung ist J2EEANT ähnlich, wobei der Sprachumsetzer mithilfe der Angaben aus der ARCHDEF festlegt, welche Teile abhängig vom Erstellungsmodus (bedingt oder erzwungen) für einen erneuten Build erforderlich sind. Die Projektquellendateien und enthaltenen Objekte werden anschließend in den Arbeitsbereich des UNIX Systems Services-Dateisystems für das Erstellungsdienstprogramm kopiert. Das Ausführungsscript und die Build-XML, die in SCLM unter Typ J2EEBLD gespeichert sind, werden ebenfalls in denselben Arbeitsbereich kopiert. Das Ausführungsscript wird aufgerufen, um Eclipse im automatischen Modus unter z/OS zu starten und die referenzierte Anwendungsbuild-XML auszuführen. Das Erstellungsdienstprogramm kompiliert und generiert erforderliche Java/J2EE-Objekte, die durch das Ausführungsscript, die Build-XML und die ARCHDEF angegeben sind.

J2EEAST überprüft die generierten Objekte und wählt für die SCLM-Aktualisierung nur die Teile/Objekte aus, die aufgrund des Erstellungsmodus in SCLM (bedingt/erzwungen) neu erstellt werden mussten.

SCLM verarbeitet jede einzelne ARCHDEF-Komponente, indem jeder Sprachumsetzer ausgeführt wird, der der Komponente zugeordnet ist. Der JAVA-Sprachumsetzer, der jedem ausgewählten Java-Quellenmember zugeordnet ist, kopiert die jeweils zugeordneten Klassendateien zurück in SCLM.

Der abschließende Teil des Erstellungsprozesses ist die Verarbeitung der ARCHDEF selbst, wobei der Sprachumsetzer J2EEOBJ ausgeführt wird. Dieser ARCHDEF-Sprachumsetzer bestimmt, welche J2EE-Objekte generiert wurden (JAR, WAR, EAR) und kopiert diese Teile zurück in SCLM.

Rational Application Developer for WebSphere Software - Implementierung und Verwendung des Erstellungsdienstprogramms

Die Implementierung und Verwendung erfolgt folgendermaßen:

- SCLM Developer Toolkit wird standardmäßig mit ARCHDEF und Assistenten für Erstellungsscripts ausgeliefert, die ARCHDEFs für Projekte und zugehörige Erstellungsscripts generieren. Diese generieren native ANT-Erstellungspro-

zessscripts und weisen die Erstellungsscripts mit einer Sprache des Typs J2EEANT zu. In SCLM Developer Toolkit finden Sie auf der Benutzervorgaben-seite von SCLM Build ein Kontrollkästchen, mit dem festgelegt werden kann, dass diese Assistenten stattdessen Erstellungsscripts erstellen.

- Auf der Seite **Team > SCLM-Vorgaben > Build-Script-Optionen** sollte jedem Build-Script-Beispiel (Java, EJB, WAR, EAR, SQLJ) das Laufzeitscript BWBASTR zugeordnet sein.
- Das Ausführungsscript (Beispiel BWBASTR) muss in jedes SCLM-Projekt kopiert werden, für das das Erstellungsdiensprogramm benötigt wird. Das Script muss im SCLM-Typ J2EEBLD gespeichert werden und den Sprachtyp TEXT oder J2EEPART haben. Dies Script muss angepasst werden. Die Anweisungen hierfür werden im Ausführungsscript selbst angegeben. Weitere Informationen zum Ausführungsscript BWBASTR finden Sie in den Erstellungsdienspro-grammscripts und -formaten.
- Jede Projektkomponente (JAR, WAR, EAR) erfordert eine ARCHDEF (Informatio-nen zum ARCHDEF-Formatlayout entnehmen Sie bitte den Erstellungsdienspro-grammscripts) und ein entsprechendes Buildsript (Informationen zum Erstel-lungsscriptlayout finden Sie in Abschnitt 2.4). Diese Erstellungsscripts müssen im SCLM-Typ J2EEBLD erstellt und mit dem Sprachtyp J2EEAST definiert wer-den. Diese Erstellungsscripts und ARCHDEFs können vom Benutzer in SCLM erstellt werden oder der Benutzer generiert diese Scripts mithilfe der Client-AR-CHDEF und der Assistenten für Erstellungsscripts unter Verwendung von **TEAM > Java/J2EE-Build-Script generieren** oder **TEAM > Zu SCLM hinzufü-gen**.
- Fehlerdetails zur Kompilierung oder Komponentenerstellung werden im Opera-tionsprotokoll für die Erstellung zurückgegeben. Falls ein Build fehlschlägt, kön-nen Sie Kompilierungsfehler lokalisieren, indem Sie im Operationsprotokoll nach "COMPILE LISTING" suchen. Wenn das Protokoll einen Fehler im Zusammen-hang mit der Konfiguration oder Installation des Erstellungsdiensprogramms an-gibt, benachrichtigen Sie den Systemadministrator. Fehlerprotokolle für die Eclipse-Erstellung sind im UNIX System Services-Dateisystem im Verzeichnis AST_INSTALL/Eclipse/configuration zu finden. Diese spezifischen Fehlerproto-kolle sind unter z/OS nicht lesbar, da sie im ASCII-Format gespeichert sind. Sie müssen in EBCDIC übersetzt oder in den Binärmodus übertragen worden sein, damit der Client-Desktop sie anzeigen kann.
- Beim Ausführen von J2EE-Erstellungen im Batch-Modus wird empfohlen, einen Regionsgrößenparameter von REGION=512M auf der Batch-Jobkarte anzugeben. Kleinere Regionsgrößen können zu Problemen aufgrund von nicht ausreichen-dem Speicherplatz führen.

Sprachumsetzer des SCLM-Erstellungsdiensprogramms

Für das Erstellungsdiensprogramm werden bestimmte Sprachumsetzer für die J2EE-Quelle, die ARCHDEF und das Erstellungsscript zugewiesen.

Beispiele für das Generieren dieser Sprachen in einem SCLM-Projekt werden in SCLM Developer Toolkit in der Installationsmusterbibliothek unter SBWBSAMP bereitgestellt.

Java-Quelle

Sprache = JAVA | JAVABIN

J2EE-Text

Sprache = J2EEPART | J2EEBIN (z. B. XML)

J2EE binär

Sprache = J2EEBIN (z. B. jpg)

Erstellungsscript

Sprache = J2EEAST

Ausführungsscript

Sprache = TEXT | J2EEPART

ARCHDEF

Sprache = archdef

Anmerkung: Das Schlüsselwort LKED=J2EEOBJ verarbeitet die ARCHDEF mit dem Sprachumsetzer J2EEOBJ.

J2EEAST: J2EE-Überprüfungs-/Erstellungsumsetzer

Hierbei handelt es sich um einen Sprachumsetzer der Script-XML des Erstellungs-dienstprogramms.

Zusammenfassung: SAMPLE: BWBTRAN4

Der J2EEAST-Überprüfungsumsetzer wird aufgerufen, wenn der Build für die ARCHDEF erstellt wird. J2EEAST ist der Sprachentyp der J2EEBLD-Build-XML-Datei, die mit dem SINC-Schlüsselwort in der ARCHDEF referenziert wird. Dieser Überprüfungsumsetzer kopiert ausgewählte Quellen und alle ARCHDEF-Objekte unabhängig vom Erstellungsmodus in den HFS-Arbeitsbereich von z/OS, damit diese während der Buildzeit als Projektarbeitsbereich referenziert werden. Dieser Projektarbeitsbereich wird temporär im Arbeitsbereich von SCLM Developer Toolkit gespeichert. SCLM DT erstellt einen temporären Projektarbeitsbereich unter der folgenden Verzeichnisstruktur: /var/SCLMDT/WORKAREA/userid/project/group/type/member/project

Das Erstellungsdienstprogramm benötigt die vollständige Projektstruktur, daher werden alle Projektkomponenten in das UNIX System Services-Dateisystem kopiert. Alle textbasierten Komponenten werden als ASCII in den Projektarbeitsbereich kopiert. Die Build-XML wird auch in den HFS-Arbeitsbereich kopiert. Die Build-XML enthält Projektdetails und J2EE-Buildanforderungen sowie andere Eigenschaftendetails, die durch SCLM referenziert werden.

Die Build-XML referenziert ein zugeordnetes Erstellungsdienstprogramm-Ausführungsscript. Dieses Ausführungsscript wird auch in den Arbeitsbereich (EBCDIC) kopiert. Wenn es ausgeführt wird, startet es Eclipse im automatischen Modus unter z/OS, um die Anwendungsbuilt-XML auszuführen. Die aktuelle Implementierung bewirkt eine vollständige Erstellung im Arbeitsbereich. SCLM verarbeitet jedoch nur die ausgewählten Teile, wenn bedingte Erstellung angefordert wurde, und der Umsetzer ermittelt anhand des Erstellungsmodus, welche generierten Objekte in SCLM aktualisiert werden müssen.

Generierte J2EE-Objekte wie JAR-, WAR- oder EAR-Archivdateien werden wieder in SCLM zurückgespeichert, wenn der ARCHDEF-Umsetzer J2EEOBJ ausgeführt wird. Ausgewählte Klassendateien werden protokolliert und in SCLM aktualisiert, wenn die ARCHDEF die einzelnen Java-Komponenten verarbeitet (Java-Umsetzer).

J2EEOBJ: J2EE-ARCHDEF-Umsetzer

Dies ist ein ARCHDEF-Sprachumsetzer, auf den mit dem LKED-Schlüsselwort verwiesen wird.

Zusammenfassung: SAMPLE: BWBTRAN3

Dies ist der endgültige Buildumsetzer, der während des ARCHDEF-Erstellungsprozesses aufgerufen wird. Dieser Umsetzer ermittelt, welche J2EE-Objekte zuvor im Umsetzer J2EEAST erstellt wurden, und kopiert diese Objekte mit dem angegebenen generierten Kurznamen in SCLM.

1. Wenn die Variable SCLM_BLDMAP gesetzt ist, rufen Sie die Build-Map-Informationen ab und aktualisieren Sie das J2EE-Objekt im Verzeichnis META-INF.
2. Kopieren Sie diese J2EE-Objekte (JAR, WAR, EAR) in SCLM mit dem zugewiesenen Kurznamen.
 - JAR/EJB in SCLM-Typ J2EEJAR
 - WAR in SCLM-Typ J2EEWAR
 - EAR in SCLM-Typ J2EEEAR
3. Kopieren Sie die Dateien mit den Details des Erstellungsprotokolls in den SCLM-Typ J2EELIST.

JAVA: Java-Sprachumsetzer (EBCDIC)

Hierbei handelt es sich um einen Sprachumsetzer des Java-Quellcodes, der in EBCDIC gespeichert ist.

Zusammenfassung: SAMPLE: BWBTRAN1

Der Sprachtyp für Java-Quellcode wird durch das Beispiel BWBTRAN1 definiert. Der Java-Umsetzer ermittelt, welcher Buildtyp für den Java-Quellcode ausgegeben wurde. Hinweis: Diese Sprachdefinition muss Java-Programmen zugewiesen werden, wenn Sie den Java-Quellcode in EBCDIC auf dem Host speichern möchten (d. h. der Quellcode kann über ISPF direkt auf dem Host angezeigt und bearbeitet werden). Der Vorteil des Definierens von Programmen mit dieser Sprachdefinition liegt darin, dass der Quellcode direkt auf dem z/OS-Host bearbeitet und angezeigt werden kann. Die Nachteile bestehen darin, dass die Codepagekonvertierungen stattfinden müssen, wenn Projekte vom Client zum Host migriert oder importiert werden müssen.

JAVABIN: Java-Sprachumsetzer (ASCII)

Hierbei handelt es sich um einen Sprachumsetzer des Java-Quellcodes, der in ASCII gespeichert ist.

Zusammenfassung: SAMPLE: BWBTRAN1

Sprachtyp, der Java ähnelt und beim Speichern von Java-Quellen als ASCII in SCLM verwendet wird.

J2EEPART: J2EE-Text-Sprachumsetzer

Hierbei handelt es sich um einen Sprachumsetzer der J2EE-Textkomponente, die in EBCDIC gespeichert ist.

Zusammenfassung: SAMPLE: BWBTRANJ

J2EEPART ist ein Sprachtyp, der eine JAVA/J2EE-Komponente angibt und durch das Beispiel BWBTRANJ definiert wird. Es wird keine spezielle Syntaxanalyse bei der Erstellung dieser Sprachdefinition ausgeführt. Nicht-Java-Quellcode oder J2EE-Komponenten, für die ASCII/EBCDIC-Sprachkonvertierung erforderlich ist, können unter dieser Sprachdefinition generisch eingefügt werden, wenn keine spezielle Build-Syntaxanalyse benötigt wird (z. B. HTML, XML, .classpath, .project, Definitionstabellen). Optional kann die Sprachdefinition TEXT verwendet werden.

J2EEBIN: Sprachumsetzer für J2EE (binär)

Hierbei handelt es sich um einen Sprachumsetzer des Java-Quellcodes, der in EBCDIC gespeichert ist.

Zusammenfassung: SAMPLE: BWBTRANJ

Sprachtyp, der die binär oder in ASCII gespeicherte JAVA/J2EE-Komponente angibt und durch das Beispiel BWBTRANJ definiert wird. Es wird keine spezielle Syntaxanalyse bei der Erstellung dieser Sprachdefinition ausgeführt. JAVA/J2EE-Binärdateien und -Textdateien, die als ASCII gespeichert werden sollen, können generisch unter dieser Sprachdefinition eingefügt werden, wenn keine gesonderte Syntaxanalyse benötigt wird.

Erstellungsscripts und Formate des Erstellungsdienstprogramms

Ebenso wie die konventionelle J2EE-Build-Servicemethode umfasst die Methode des Erstellungsdienstprogramms Erstellungsscripts, die in der ARCHDEF durch das Schlüsselwort SINC referenziert werden.

Das aufgerufene Erstellungsscript ist eine Erstellungsscriptanwendung zum Erstellen von XML, die für jedes J2EE-Projekt angepasst wird und eindeutig ist. Dieses Erstellungsscript referenziert außerdem ein Erstellungsdienstprogramm-RUN-Script, das globale Erstellungsdienstprogramm-Eigenschaften und Eclipse-Laufzeitbefehle enthält, um Eclipse unter z/OS im automatischen Modus zu starten und die ausgewählte Build-XML auszuführen. Im Allgemeinen wird das angepasste BWBASTR-Script von allen Erstellungsdienstprogramm-Build-Scripts verwendet. Der Sprachtyp muss für alle Erstellungsdienstprogramm-Erstellungsscripts J2EEAST sein. Der Sprachtyp für das BWBASTR-Script sollte ein Nur-Text-Sprachumsetzer sein, z. B. TEXT oder J2EEPART.

Das Ausführungsscript des Erstellungsdienstprogramms (BWBASTR), die Beispiel-Erstellungsscripts für Java/Jar-Projekte (BWBASTJ), EJB-Projekte (BWBASTEJ), Anwendungsclientprojekte (BWBASTAP), Webprojekte (BWBASTW) und EAR-Projekte (BWBASTE) können aus der SCLM Developer Toolkit-AST-Beispielbibliothek im Typ J2EEBLD in die SCLM-Projekte des Kunden kopiert werden.

Format des Erstellungsscripts

Das Format des Erstellungsscript enthält folgende Elemente:

SCLM_ARCHDEF

Der Name der referenzierten ARCHDEF, die erstellt wird.

AST_SHSCRIPT

Name des AST-RUN-Scripts zum Ausführen von AST unter z/OS (siehe Beispiel BWBASTR).

PROJECT NAME

Der J2EE-Projektnamen (nicht der SCLM-Projektnamen).

JAR_FILE_NAME

Bei Java-Projekten der Name der generierten JAR-Datei

WAR_NAME

Bei Webprojekten der Name der generierten WAR-Datei.

EJB_NAME

Bei EJB-Projekten der Name der generierten JAR-Datei.

EAR_NAME

Für Unternehmensanwendungen der Name der generierten EAR, die implementiert werden kann.

SCLM_BLDMAP

Falls „YES“, im Verzeichnis MANIFEST in JAR, WAR und EAR einschließen. Stellt die Prüfung und die Buildzuordnung der eingeschlossenen Teile bereit.

Ebenfalls enthalten sind die AST-ANT-Routinen zum Generieren des erforderlichen Projekts. Die in SCLM benötigten Erstellungsscripts sind geänderte Versionen der folgenden durch AST bereitgestellten Beispiel:

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

Nicht modifizierte AST-Routinen

Die folgenden Routinen wurden aus dem Handbuch zu Application Server Toolkit extrahiert.

projectImport: Diese Task importiert ein vorhandenes Dateisystemprojekt in einen Arbeitsbereich.

Tabelle 13. Parameter für projectImport

Attribut	Beschreibung	Erforderlich
ProjectName	Name des zum importierenden Projekts	Ja
ProjectLocation	Die vollständig qualifizierte Speicherposition des Projekts (entweder unter dem Arbeitsbereich oder an anderer Stelle im Dateisystem)	Nein, die Standardeinstellung ist \${workspaceLocation}/\${projectName}

Beispiel: Importieren eines Projekts, das unter dem Arbeitsbereichsverzeichnis gespeichert ist, aber derzeit nicht im Arbeitsbereich vorhanden ist:

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

projectBuild: Diese Task erstellt das angegebene Projekt.

Tabelle 14. Parameter für projectBuild

Attribut	Beschreibung	Erforderlich
ProjectName	Name des zu erstellenden Projekts.	Ja
BuildType	Buildtyp.	Nein, die Standardeinstellung ist „Incremental“. Kann „Incremental“ oder „Full“ sein.
FailOnError	Gibt an, ob Erstellungen bei einem Fehler fehlschlagen sollen.	Nein, die Standardeinstellung ist „true“.
DebugCompilation	Gibt an, ob eine Fehlerbehebung für Kompilierungen ausgeführt werden soll.	Nein, die Standardeinstellung ist „true“.

Tabelle 14. Parameter für projectBuild (Forts.)

Attribut	Beschreibung	Erforderlich
Quiet (veraltet)	Gibt an, ob Nachrichten ausgegeben werden sollen.	Nein, Standardeinstellung ist „false“.
ShowErrors	Gibt an, ob Projektfehler im Ant-Erstellungsprotokoll angezeigt werden sollen.	Nein, die Standardeinstellung ist „true“.
SeverityLevel	Die Problemebene, nach der Erstellungsfehler gezählt und als solche behandelt werden sollen.	Nein, die Standardeinstellung ist „ERROR“. Kann „ERROR“, „WARNING“ oder „INFORMATION“ sein.
CountValidationErrors	Gibt an, ob Überprüfungsprobleme als Projektfehler zählen sollen.	Nein, die Standardeinstellung ist „true“.
PropertyCountName	Eigenschaft zum Empfangen der Fehlerzählung für das Projekt.	Nein, die Standardeinstellung ist „ProjectErrorCount“.
PropertyMessagesName	Eigenschaft zum Empfangen der Fehlermeldungen für das Projekt.	Nein, die Standardeinstellung ist „ProjectErrorMessages“.

Beispiele:

- Build „myProject“. Die Standardeinstellung sieht inkrementelle Erstellung mit Debuginformationen vor:

```
<projectBuild projectName="myProject" />
```
- Führen Sie eine Produktionserstellung von „myProject“ durch, eine vollständige Erstellung ohne Debuginformationen:

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

EJBexport: Diese Task führt denselben Vorgang aus wie der EJB-JAR-Datei-Exportassistent für den Export eines EJB-Projekts in eine EJB-JAR-Datei. Diese Task ist nicht verfügbar in Produkten, die keine EJB-Entwicklungstools enthalten.

Tabelle 15. Parameter von EJBexport

Attribut	Beschreibung	Erforderlich
EJBProjectName	Name des EJB-Projekt (Groß-/Kleinschreibung muss beachtet werden).	Ja
EJBExportFile	Absoluter Pfad der EJB-JAR-Datei.	Ja
ExportSource	Legt fest, ob Quellendateien eingeschlossen werden oder nicht.	Nein, Standardeinstellung ist „false“.
Overwrite	Legt fest, ob die Datei überschrieben wird, falls schon vorhanden.	Nein, Standardeinstellung ist „false“.

Beispiel: Exportieren des Projekts "EJBProject" nach "EJBProject.jar"

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

WARExport: Diese Task führt denselben Vorgang aus wie der WAR-Datei-Exportassistent für den Export eines Webprojekts in eine WAR-Datei.

Tabelle 16. Parameter für WARExport

Attribut	Beschreibung	Erforderlich
WARProjectName	Name des Webprojekts (Groß-/Kleinschreibung muss beachtet werden)	Ja
WARExportFile	Absoluter Pfad der WAR-Datei	Ja
ExportSource	Legt fest, ob Quellendateien eingeschlossen werden oder nicht.	Nein, Standardeinstellung ist „false“.
Overwrite	Legt fest, ob die Datei überschrieben wird, falls schon vorhanden.	Nein, Standardeinstellung ist „false“.

Beispiel: Exportieren des Projekts "ProjectWeb" nach "ProjectWeb.war"

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

EARExport: Diese Task führt denselben Vorgang aus wie der WAR-Datei-Exportassistent für den Export eines Webprojekts in eine WAR-Datei.

Tabelle 17. Parameter für EARExport

Attribut	Beschreibung	Erforderlich
EARProjectName	Name des EAR-Projekts (Groß-/Kleinschreibung muss beachtet werden)	Ja
EARExportFile	Absoluter Pfad der EAR-Datei	Ja
ExportSource	Legt fest, ob Quellendateien eingeschlossen werden oder nicht.	Nein, Standardeinstellung ist „false“.
IncludeProjectMetaFiles	Angaben dazu, ob die Projektmetadatendateien mit dem Java-Erstellungspfad, die Projektnamen usw. integriert werden sollen. Wird beim erneuten Import als binäres Projekt verwendet.	Nein, Standardeinstellung ist „false“.
Overwrite	Legt fest, ob die Datei überschrieben wird, falls schon vorhanden.	Nein, Standardeinstellung ist „false“.

Beispiel: Exportieren des Projekts "EARProject" nach "EARProject.ear"

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

AppClientExport: Diese Task führt denselben Vorgang aus wie der WAR-Datei-Exportassistent für den Export eines Webprojekts in eine WAR-Datei.

Tabelle 18. Parameter für AppClientExport

Attribut	Beschreibung	Erforderlich
AppClientProjectName	Name des Anwendungsclientprojekts (Groß-/Kleinschreibung muss beachtet werden)	Ja
AppClientExportFile	Absoluter Pfad der Anwendungsclient-JAR-Datei	Ja
ExportSource	Legt fest, ob Quellendateien eingeschlossen werden oder nicht.	Nein, Standardeinstellung ist „false“.
Overwrite	Legt fest, ob die Datei überschrieben wird, falls schon vorhanden.	Nein, Standardeinstellung ist „false“.

Beispiel: Exportieren des Projekts "ProjectClient" nach "ProjectClient.jar":


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

AST-Ausführungsscript (Beispiel-BWBASTR): Das AST-Ausführungsscript wird referenziert durch AST-Erstellungsscripts (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 'PARGS
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-Erstellungsscript (Beispiel): Das folgende Erstellungsscript hat type=J2EEBLD lang=J2EEAST. Es handelt sich um einen beliebigen Namen (bis zu 8 Zeichen). Variablen werden im Standard-XML-Format definiert.

```

<!--

```

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-Erstellungsscript (Beispiel): Das folgende Erstellungsscript hat 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-Erstellungsscript (Beispiel): Das folgende Erstellungsscript hat 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-Erstellungsscript (Beispiel): Das folgende Erstellungsscript hat 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: Das Format für die ARCHDEF ist dasselbe wie für den normalen J2EEANT-Erstellungsprozess.

SINC Quelle enthält das J2EEBLD-Erstellungsscript.

INCLD

SCLM enthält J2EE-Komponente (z. B. Java-Quellcode).

INCL SCLM enthält eine andere ARCHDEF

OUT1 Gibt den J2EE-Objektyp an, der von dieser ARCHDEF erstellt wurde:

- J2EEEAR
- J2EEWAR
- J2EEJAR

LIST Zusammenfassungsliste der Komponenten und Prüfung der ARCHDEF-Erstellung. Ist in TYPE=J2EELIST unter dem ARCHDEF-Membernamen enthalten.

LKED Gibt LEC ARCHDEF an sowie die Sprache des aufzurufenden ARCHDEF-Umsetzers (für J2EE ARCHDEFs ist dies immer J2EEOBJ).

```

SINC SCRIPT1      J2EEBLD
INCLD XX000001    SOURCE
INCL  PAULWAR2    ARCHDEF

OUT1  *           J2EEEAR
LIST  * J2EELIST
LKED  J2EEOBJ

```

SQLJ-Buildunterstützung

SCLM bietet SQLJ-Unterstützung durch die Java/J2EE-Buildumsetzer, die in SCLM Developer Toolkit enthalten sind. Diese Umsetzer in Verbindung mit den AST-Erstellungsscripts bieten Unterstützung zum Speichern und Erstellen von SQLJ-Projekten. Sie bieten außerdem Integrationspunkte, mit denen in Kundenroutinen db2-sqlj-Anpassungseigenschaften festgelegt werden können und die den db2-Bindungsprozess in den SCLM-Erstellungs- und Umstufungsschritten durch SCLM-Erstellungs- und Umstufungs-Exits unterstützen.

Weitere Informationen zur SQLJ-Unterstützung finden Sie im Benutzerhandbuch für SCLM Developer Toolkit.

Die SQLJ-Unterstützung in SCLM (unter Verwendung der AST-Methode) kann wie folgt zusammengefasst werden:

- Speichern der SQLJ-Quelle in SCLM und Zuweisen eines Sprachtyps von SQLJ für die Quelle
- Erstellen einer Java/J2EE-ARCHDEF, die SQLJ-Member als Teil der ARCHDEF umfasst. Die ARCHDEF referenziert ein SQLJ-Erstellungsscript mit dem Typ J2EEBLD, das SQLJ-Eigenschaften enthält, die vom Benutzer definiert und angepasst werden müssen (siehe Beispielscripts BWBASTSQ und BWBASTSE).
- Optionale DB2-Bindungsnachbearbeitung (unter Verwendung von generierten und in SCLM gespeicherten DBRM-Dateien). Die Bindungsnachbearbeitung wird durch Erstellungs- und Umstufungs-Benutzer-Exits gesteuert. DBRM-Dateien, auf die innerhalb der von SCLM gesteuerten Datensätze direkt zugegriffen wird.
- Implementierung der erstellten JAR/EAR durch den SCLM DT-Implementierungsprozess

SQLJ-Anpassung (für den SCLM-Administrator)

Systemvoraussetzungen: DB2-SQLJ-Unterstützung muss installiert sein. Das folgende DB2-Verzeichnis (oder ein anderes Verzeichnis, abhängig vom Standort-Installationsverzeichnis) muss im z/OS USS-Dateisystem gespeichert werden: /usr/lpp/db2/db2810/* (dies ist das DB2 v8-Verzeichnis). Dieses Verzeichnis wird für die Anpassung des SQLJ-Erstellungsscripts benötigt. Für SQLJ sind Klassenpfadabhängigkeiten unter /usr/lpp/db2/db2810/jcc/classes/sqlj.zip vorhanden. Für db2sqljcustomize sind die Klassenpfadabhängigkeiten unter /usr/lpp/db2/db2810/jcc/classes/db2jcc.jar gespeichert.

Sprachumsetzer:

SQLJ: Ein neuer Sprachumsetzer für SQLJ wird bereitgestellt und sollte als Sprachtyp für den gesamten SQLJ-Quellcode zugewiesen werden, der in SCLM gespeichert wird. Für den neuen Umsetzer müssen zusätzliche SCLM-Typen definiert werden. Der neue Umsetzer ist ähnlich aufgebaut wie der JAVA-Umsetzer, enthält jedoch weitere IOTYPE-Definitionen für die SCLM-Ausgabetypen SQLJSER und DBRMLIB. Wenn der Kunde keine DBRM-Dateien während des db2sqljcustomize-Schritts generiert, kann dieser DBRMLIB IOTYPE aus der SQLJ-Sprachdefinition entfernt werden. (Verwenden Sie die SQLJ-Beispiel-Umsetzungsdefinition BWB-TRANS.) Generieren Sie innerhalb der Projekt-PROJDEF den neuen SQLJ-Umsetzer und die zusätzlichen Typen folgendermaßen:

- **SQLJSER:** Ein Typ, der die generierten serialisierten Profildateien (.ser-Dateien) enthalten muss, die in den Schritten sqlj und db2sqljcustomize erstellt/angepasst wurden. Es wird empfohlen, diesen Datensatz vom Typ SCLM als recfm=VB, lrecl= 256 zu definieren.

- DBRMLIB: Ein Typ, der die generierten DBRM-Dateien enthalten muss, die im Schritt db2sqljcustomize erstellt wurden. Dieser Typ wird nur für Kunden benötigt, die generierte DBRM-Dateien im Rahmen des DB2-Bindeprozesses verwenden. Es wird empfohlen, diesen Datensatz vom Typ SCLM als recfm=VB, lrecl=256 zu definieren.

SQLJ-Benutzeranpassung

SQLJ-Build-Eigenschaftenscript: Kunden müssen SQLJ-DB2-Eigenschaften im Haupt-SQLJ-Erstellungsscript (Beispiel BWBASTSQ) definieren, das im Typ J2EEBLD gespeichert wird. Diese Eigenschaften werden in den Schritten "sqlj" und "db2sqljcustomize" verwendet.

Anmerkung: Der bereitgestellte SQLJ-Entwurf BWBASTSQ ist lediglich eine Erweiterung des normalen Java-Buildentwurfs BWBASTJ. BWBASTSE ist ebenfalls eine sqlj-Erweiterung des EJB-Beispiels BWBASTEJ. Kunden können SQLJ- & Java-Quellenmember in derselben ARCHDEF mischen und BWBASTSQ verwenden. Der dabei entstehende Build erstellt eine JAR-Datei, die alle generierten Klassen- und .ser-Dateien enthält.

Im Folgenden finden Sie eine Liste der erforderlichen "sqlj"- und "db2sqljcustomize"-Eigenschaftendefinitionen.

Allgemeine Eigenschaftseinstellungen

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

SQLJ (Structured Query Language for Java)

```
<!-- 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 ***

Die Argumenteigenschaften werden verwendet, um die erforderliche Argumentzeichenfolge im Aufruf von SQLJ oder db2sqljcustomize zu erstellen. Beispiel:

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

SCLM ARCHDEF (Beispiel ASTSQLJ):

```
*
*
LKED J2EE0BJ          * 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-Erstellungsscript (Beispiel 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-Quelle in Archivdateien

Standardmäßig wird der Java-Quellcode als ASCII in den USS-Arbeitsbereich kopiert, bevor die Kompilierungserstellungen ausgeführt werden. Dies gilt auch, wenn der Java-Quellcode in SCLM als EBCDIC gespeichert wird.

Wenn der Benutzer anfordert, dass Java-Quellcode in der Archivdatei (JAR) enthalten sein soll, wird der Quellcode standardmäßig in ASCII gespeichert. Es ist jedoch möglich, die Erstellungsscripts so zu konfigurieren, dass die Java-Quelle in den Arbeitsbereich kopiert und im EBCDIC-Format kompiliert wird, so dass dann, wenn der Einschluss der Java-Quelle gewünscht wird, diese im EBCDIC-Format vorliegt.

Um den Java-Quellcode im EBCDIC-Format einzuschließen, müssen folgende Erstellungsscripts entsprechend geändert werden, und zwar folgendermaßen:

- Fügen Sie die Zeile `<property name="ASTJAVA_ENCODING" value="EBCDIC" />` in das XML-Haupterstellungsscript ein und entfernen Sie wie folgt das Schlüsselwort EXCLUDES aus der JAR-Aktualisierungsroutine: `jar update="true" destfile="${JAR_FILE_NAME}"> <fileset dir="." excludes="**/*.java"/>`.
- Im AST-Ausführungsscript, das durch das Schlüsselwort AST_SHSCRIPT referenziert wird, entfernen Sie das Schlüsselwort `-Dfile.encoding=ISO8859-1` `A_parm = '-Dfile.encoding=ISO8859-1 -Xnoargsconversion'`

EINSATZSZENARIO: 'PlantsByWebSphere'

In WebSphere Application Server sind zahlreiche Beispielprojekte vorhanden. Diese können entweder mithilfe von vorerstellten EARs implementiert werden oder mit einem ANT-Erstellungsprozess assembliert werden. An dieser Stelle finden Sie ein Beispiel für die Verwendung von SCLM Developer Toolkit zum Einchecken des Projekts in SCLM und Erstellen/Implementieren unter z/OS. Dabei wird AST als Erstellungsprozess verwendet. PlantsByWebSphere ist ein J2EE-Projekt, mit dem ein Online-Einkaufsshop für Pflanzen eingerichtet wird. Es besteht aus den folgenden vier Komponenten:

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

In diesem Dokument wird ein Szenario beschrieben, bei dem diese Quelle in eine Eclipse-Projektstruktur eingefügt, in Developer Toolkit eingchecked und auf dem Host erstellt und implementiert wird.

1. VERWALTUNGSAUFGABE: Fügen Sie die folgenden Typen zur Projektdefinition hinzu:
 - PLANTEAR - Typ für die EAR-Datei
 - PLANTEJB - Typ für die Enterprise JavaBeans

- PLANTWE1 - Typ für das erste Webarchiv
 - PLANTWE2 - Typ für das zweite Webarchiv
2. VERWALTUNGSAUFGABE: Erstellen Sie das Projekt neu (Job übergeben).
 3. ADMINISTRATIVE TASK: Ordnen Sie die folgenden Datengruppen zu:
 - <HLQ>.<DEVGROU>.PLANTEJB
 - <HLQ>.<DEVGROU>.PLANTEAR (groß)
 - <HLQ>.<DEVGROU>.PLANTWE1 (groß)
 - <HLQ>.<DEVGROU>.PLANTWE2 (groß)
 4. Lokalisieren Sie den Quellcode „PlantsByWebSphere“ auf einer WAS-Installation. (~root/AppServer/samples/src/PlantsByWebSphere)
 5. Starten Sie ein Developer Toolkit-fähiges Eclipse-Produkt Ihrer Wahl.
 6. Importieren Sie die 4 Komponenten in 4 Projekte in Ihrem Eclipse-Arbeitsbereich.
 7. Stellen Sie sicher, dass die Dateien des Typs „.project“ und „.settings“ eingeschlossen und für jedes Projekt ordnungsgemäß eingerichtet werden.
 8. Richten Sie die Compilereinstellungen und weitere Websphere-spezifische Konfigurationsdateien ein. (Projekt... Eigenschaften)
 - Setzen Sie die Java-Facette auf 1.4.
 - Stellen Sie sicher, dass dem Projekt die richtige WAS-Laufzeit-Stub-Datei zugeordnet wird.
 9. Zu SCLM hinzufügen (Team->Zu SCLM auf Projektknoten hinzufügen)
 - a. Die zu verwendenden Sprachen sind folgende:
 - Images – J2EEBIN
 - XML-Dateien – J2EEPART
 - Einstellungen – J2EEPART
 - Java-Quellcode – JAVA
 - b. Die zu verwendenden Typen sind folgende:
 - EAR-Dateien – PLANTEAR
 - EJB-Dateien – PLANTEJB
 - WAR 1-Dateien – PLANTWE1
 - WAR 2-Dateien – PLANTWE2
 - c. ARCHDEF (letzte Seite des Assistenten 'Zu SCLM hinzufügen')
 - Jede Komponente erhält eine ARCHDEF mit J2EE-Schlüsselwörtern und AST-Erstellungsscript.
 - EAR ARCHDEF sollte ARCHDEFs anderer Komponenten enthalten.

PLANTEAR ARCHDEF (Beispiel):

```
*
*
LKED J2EE0BJ          * 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. Erstellungsscripts mit den wie folgt benannten Ausgabemodellen:

- PlantsByWebSphere.ear (PLANTEAR)
- PlantsByWebSphereEJB.jar (PLANTEJB)
- PlantsByWebSphere.war (PLANTWE1)
- PlantsGallery.war (PLANTWE2)

PLANTEAR BUILDSCRIPT (Beispiel):

```

<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>

```

Für PLANTWE1 muss EJB sich zum Zeitpunkt der Erstellung im Klassenpfad befinden. Ändern Sie daher das Erstellungsscript von PLANTWE1, indem Sie eine Eigenschaft für CLASSPATH_JARS_FILES und den Wert "PlantsByWebSphereEJB.jar" hinzufügen.

10. Erstellungsprojekt (durch Archdef PLANTEAR)

11. Führen Sie die Implementierung aus:

- a. Bereiten Sie das Implementierungsaktionsscript vor (deploy_ben.jacl). Für diese Implementierung muss eine Datenquelle bzw. ein Connector/Mail-Provider in WAS erstellt werden. Daher ist ein angepasstes Implementierungsscript erforderlich.

```

#-----
#
# 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 {[length $app] == 0} {
    puts "deploy: Error -- No application specified."
    return
}
```

```
#-----
# -- was the appname supplied
#-----
if {[length $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  "slamples"
createJAASAuthenticationAlias $cellName $secAuthAlias $secDescript
    $secUserID $secPassword
```

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

```
set temp1Name "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
# Additional 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 myappl 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. Generieren Sie ein Eigenschaftsscript im Front-End und starten Sie die Implementierung. (Implementierungsentwurf)
12. Überprüfen Sie die Implementierungsnachrichten. Wenn die Implementierung erfolgreich war, zeigen Sie das Projekt auf Ihrem WAS-Server an.

Anhang E. BUILD FORGE und SCLM

In diesem Abschnitt werden die Implementierungs- und Konfigurationsprobleme für den Aufruf und die Verwendung von SCLM über Build Forge dargestellt. Das Kapitel enthält Anpassungs- und Testmuster für SCLM-Erstellungen und -Umstellungen.

Übersicht

Der Build Forge-Konsolenserver befindet sich im Allgemeinen auf einer Clientplattform und muss mit einem SCLM-Serviceaufruf im Projektbereich der Konsole konfiguriert werden. Dieses SCLM-konfigurierte Projekt stellt beim Starten eine Verbindung mit dem Build Forge-Agentenserver unter z/OS her, der zuvor gestartet worden sein muss. Das Plug-in Build Forge von Rational Developer for System z ermöglicht den Start von Konsolprojekten aus der Rational Developer for System z-Umgebung über eine Socketverbindung zum Konsolenserver. Auf die Ausgabe abgeschlossener Projektläufe kann auch über das Plug-in Build Forge von Rational Developer for System z innerhalb von Rational Developer for System z zugegriffen werden.

Voraussetzungen

- Plug-in für Build Forge V7.1 installiert in Rational Developer for System z 7.6 oder höher
- Build Forge-Konsole/Server V7.1
- Build Forge-Agent für z V7.1 (src-bfagent-7.1.1.0-0022.tar.gz oder höher)
<http://www-01.ibm.com/support/docview.wss?rs=3099&uid=swg24016541>

Anmerkung: Stellen Sie sicher, dass die richtige Version des Build Forge-Plug-ins mit der entsprechenden Version des Servers und z/OS-Agenten verwendet wird. Die kompatiblen Versionen des Plug-ins und z-Agenten sind im Paket enthalten, das durch den Haupt-Build Forge-Server verteilt wird.

Um die korrekte Version des Plug-ins hochzuladen, nehmen Sie die Aktualisierung von Ihrer Serverposition aus vor: http://<console_host_name>/prism/eclipse/updateSite/site.xml

Build Forge-Agent unter z/OS aufrufen

Starten Sie den Build Forge-Agenten unter z/OS. Der Standardport für den Agenten ist 5555. Beispiel:

```
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)
```

Die Beispielantwort lautet wie folgt:

```
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 - Konsolenserverkonfiguration

Die folgende Mindestkonfiguration wird benötigt, um SCLM-Funktionen in Build Forge zu aktivieren.

1. Server (Konfigurieren Sie den Server so, dass er auf den Build Forge-Agenten für System z verweist.)
 - a. Richten Sie die Serverautorisierung mit z/OS-Benutzer-ID/Kennwort ein.
Klicken Sie auf **Server -> Serverautorisierung**

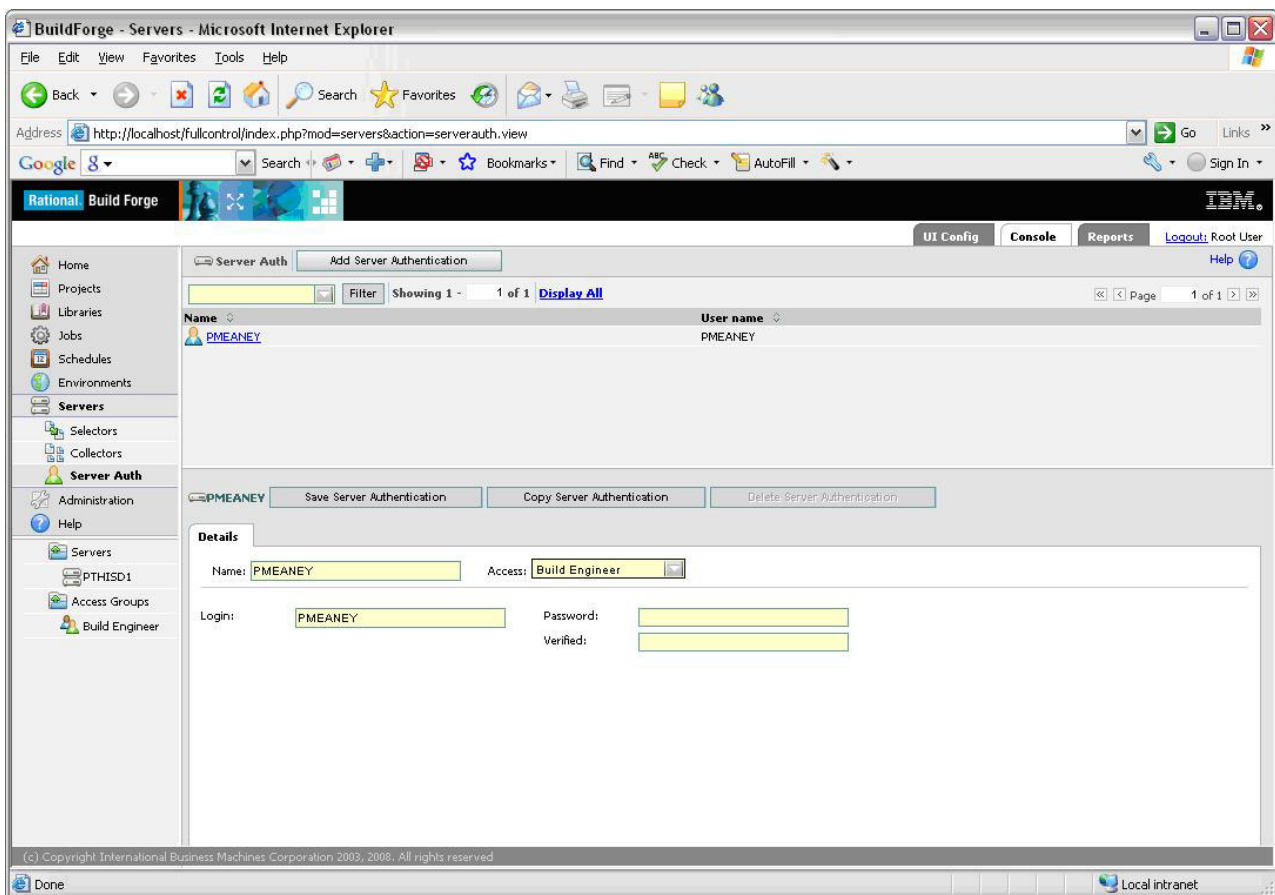


Abbildung 32. Serverautorisierung

Diese Benutzer-ID muss eine gültige Benutzer-ID sein und das Kennwort muss auf dem z/OS-System vorhanden sein, auf dem der Agent ausgeführt wird und auf dem Sie den SCLM-Service ausführen möchten. Die Zugriffsgruppe wird eine vorhandene Zugriffsgruppe oder eine Standardzugriffsgruppe sein, die sich im Konsolenserver befindet und über die entsprechenden Berechtigungen verfügt: **Verwaltung -> Zugriffsgruppen**.

- b. Richten Sie die Haupt-z/OS-Serverkonfiguration ein. (Klicken Sie auf **Server**.)

NAME: Identifizierbarer eindeutiger Name für Ihre Serverdefinition

PFAD: Das Pfadverzeichnis auf dem z/OS-Host, auf dem die Build-

verzeichnisse erstellt werden. Die Buildverzeichnisse haben üblicherweise das Format PATH/Projektname/BUILD_#/*

Die SCLM-Eingabedatei wird hier erstellt und die entsprechende Ausgabeantwort auf den Serviceaufruf wird hier gespeichert entsprechend der Namenskonvention, die in der Projektdefinition für die Konsole angegeben ist.

HOST: Geben Sie den Namen des Zielhosts (DNS oder IP-Adresse) und den Port ein (z/OS-Agent-Standard 5555). Hostname:Port

ZUGRIFF: Geben Sie den Berechtigungszugriffstyp ein (Beispiel: Build Engineer).

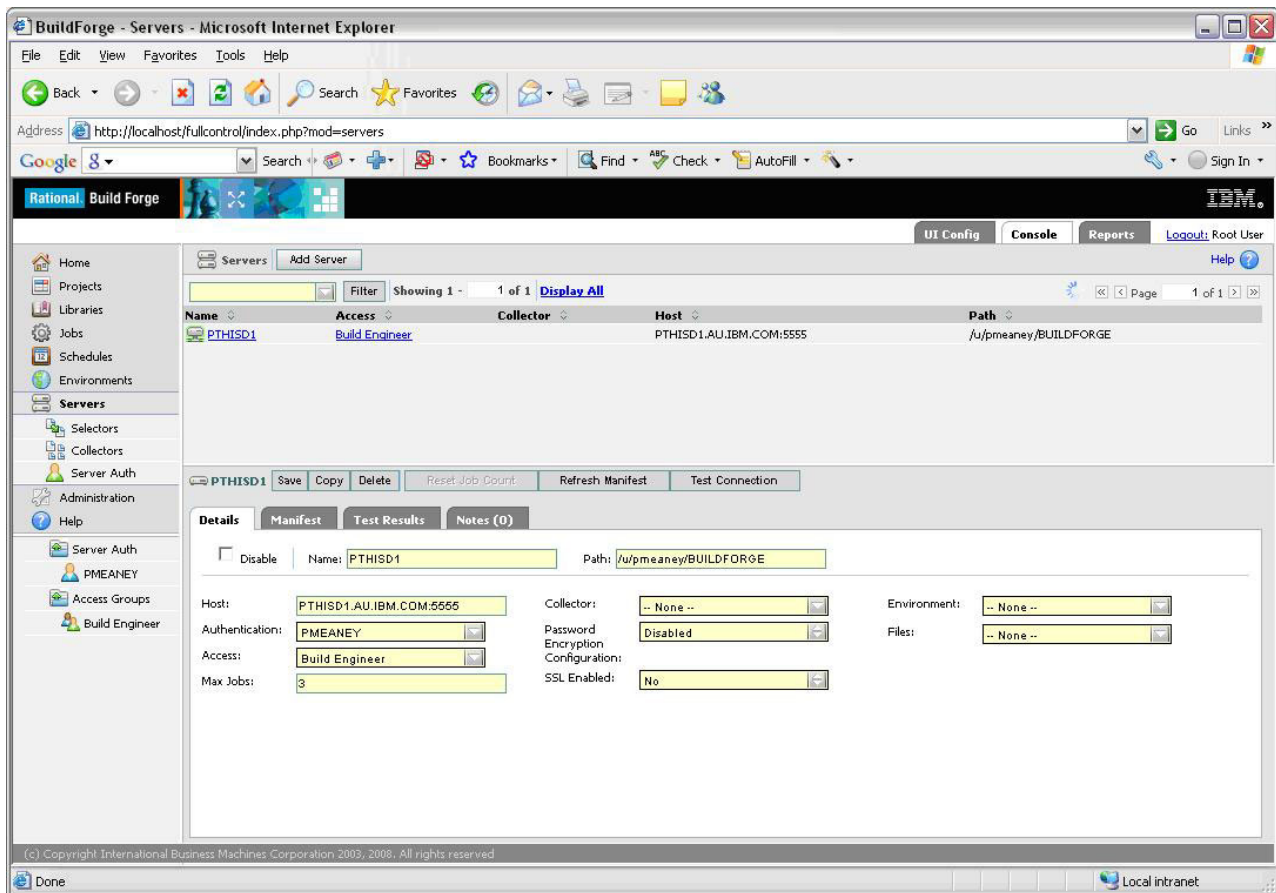


Abbildung 33. Serverdefinition

Testen Sie die Verbindung mit dem Host, indem Sie auf **Verbindung testen** klicken. Dadurch wird ein Verbindungstest für den Serveragenten durchgeführt, der unter z/OS ausgeführt wird. Ein erfolgreicher Test sieht wie folgt aus:

```
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. Richten Sie den Hauptsektor für den Server ein. Klicken Sie auf **Server -> Selektor**.

Weisen Sie Ihrem Servernamen BF_NAME zu. (Beispiel: PTHISD1)

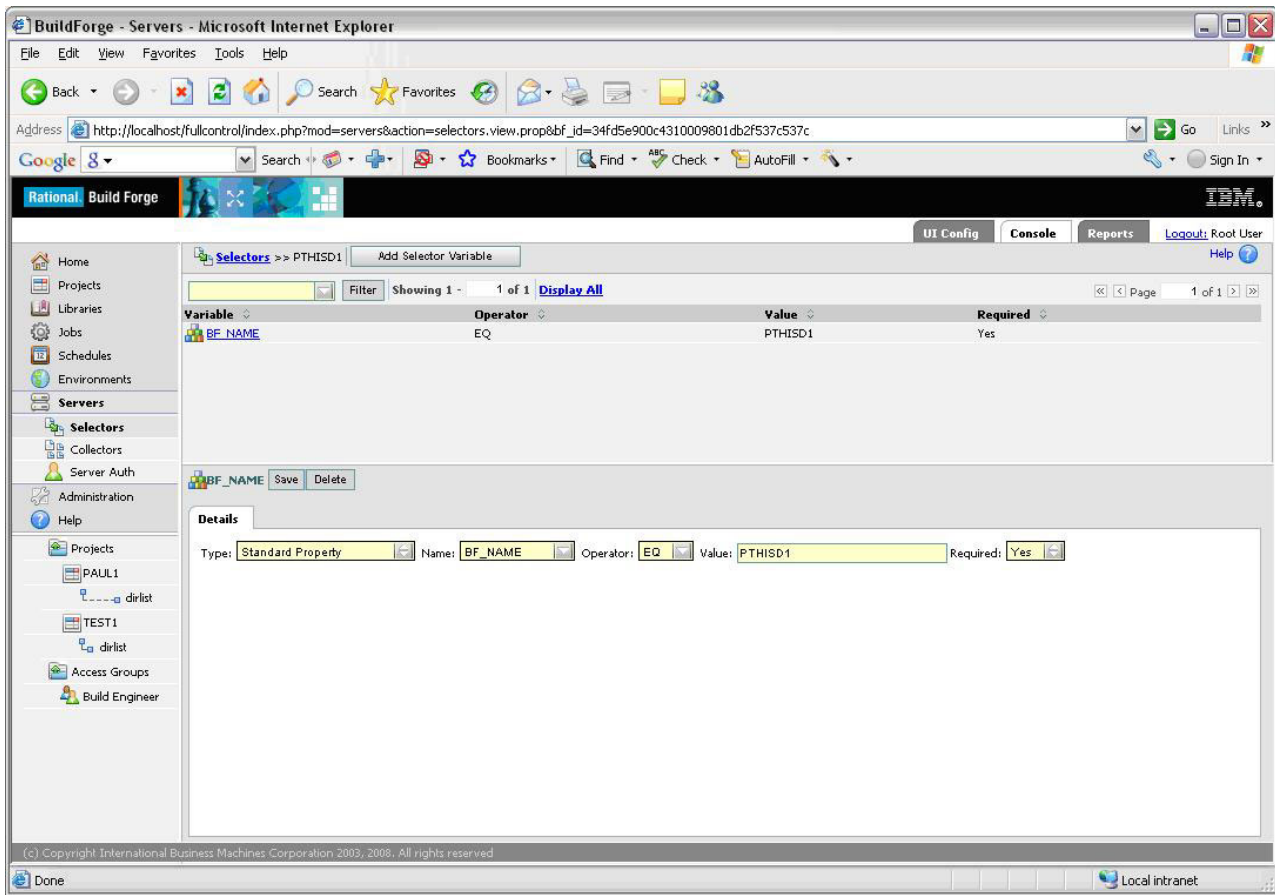


Abbildung 34. Definition des Serverselektors

2. Einrichtung der Umgebungsvariablen (Konfigurieren der SCLM Developer Toolkit-Umgebungsvariablen für das SCLM-Projekt)
 - a. Klicken Sie auf **Umgebungen** – Erstellen Sie einen Namen für die SCLM DT-Umgebungsvariablenliste. Beispiel: SCLMDT
Konfigurieren Sie die folgenden SCLM DT-Umgebungsvariablen:

STEPLIB: Die Dateigruppe STEPLIB, in der die SCLM Developer Toolkit-Module sich befinden. Diese befinden sich im SFEKLOAD-Datensatz von Rational Developer for System z.
(Beispiel: RD4Z.V760.SFEKLOAD) Der Aktionstyp in der Definition sollte APPEND sein.

_CMDSERV_BASE_HOME: Das Basisverzeichnis, in dem das ISPF-Gateway installiert ist (Beispiel: /usr/lpp/ispf) .

_CMDSERV_CONF_HOME: Das Konfigurationsverzeichnis für das ISPF-Gateway (Beispiel: /etc/ispf) .

_CMDSERV_WORK_HOME: Das Arbeitsverzeichnis für das ISPF-Gateway (Beispiel: /var/ispf) .

_SCLMDT_CONF_HOME: Das Konfigurationsverzeichnis für SCLM DT innerhalb von Rational Developer for System z.
(Beispiel: /etc/rd4z760/sclmdt) .

_SCLMDT_WORK_HOME: Das Arbeitsverzeichnis für SCLM DT innerhalb von Rational Developer for System z.

In der Regel verwenden Sie dasselbe Verzeichnis wie für die Variable `_CMDSERV_WORK_HOME`.
(Beispiel: `$_CMDSERV_WORK_HOME`).

PATH: Die folgenden Verzeichnispfade -- Rational Developer for System z-Bin, die `_CMDSERV_BASE_HOME`-Bin und das Rational Developer for System z SCLMDT-Scriptverzeichnis.

Der Aktionstyp sollte APPEND sein.

Beispiel:

`/apc/trdz750/usr/lpp/rdz/bin:/etc/rd4z760/sclmdt/CONFIG/script:
$_CMDSERV_BASE_HOME/bin`

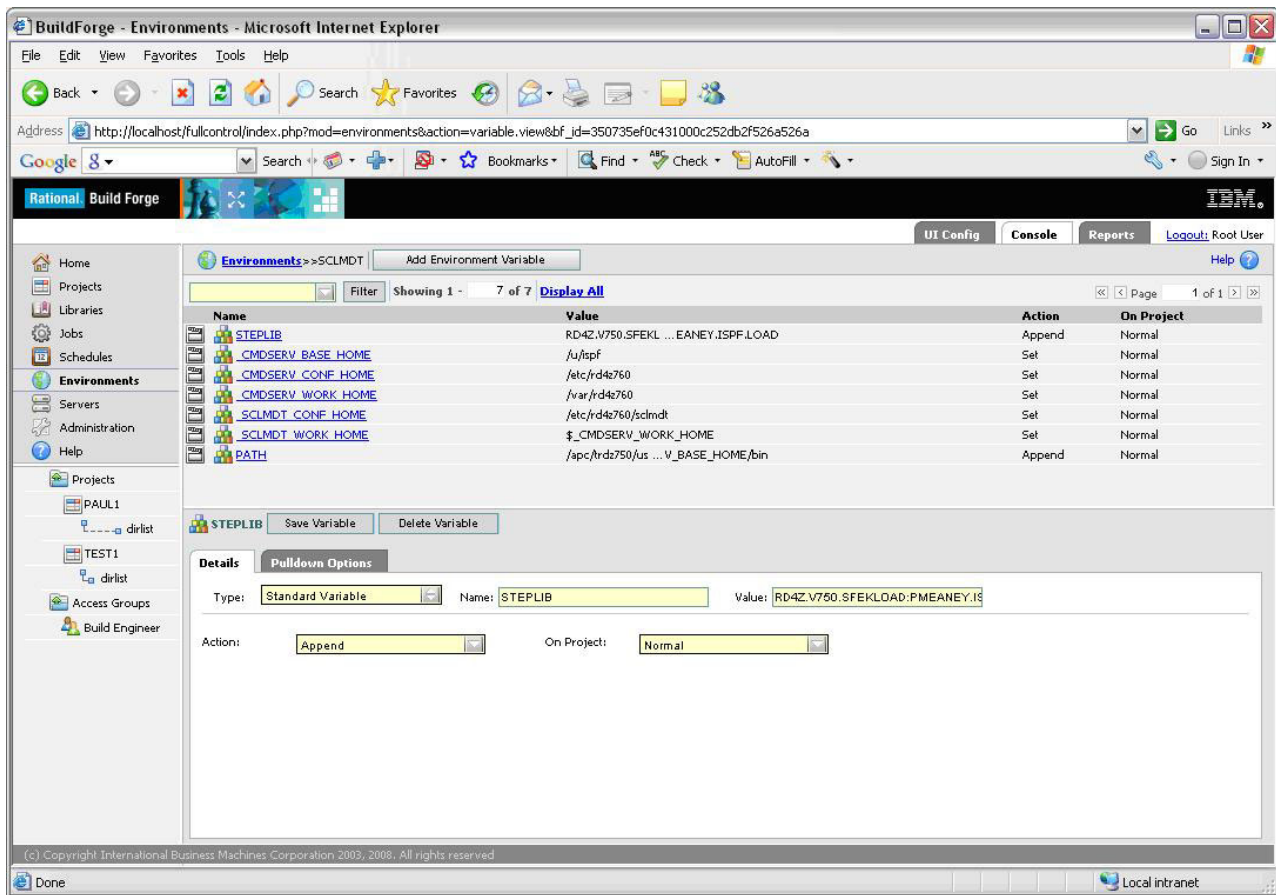


Abbildung 35. Umgebungsvariablendefinitionen

3. SCLM-Projektdefinition in Build Forge

Nun müssen SCLM-Projekte für die verschiedenen SCLM-Serviceanforderungen konfiguriert werden. Die unten stehenden Beispiele gelten für SCLM-Erstellungs- und -Umstufungsservices. Diese sind im Allgemeinen die wichtigsten Services, die für die Build Forge-Jobplanung geeignet sind. Die Projektskriptmuster basieren auf dem Format SCLM XML API, das im *SCLM Developer Toolkit Administratorhandbuch* dokumentiert wird.

- Zuerst definieren Sie ein SCLM-Projekt durch Klicken auf **Projekte -> Projekt hinzufügen**.

NAME: Geben Sie den Namen des Projekts ein.

(Beispiel: SCLM build sample1)

SELEKTOR: Geben Sie den Selektornamen ein, der in der Serverselektortabelle konfiguriert wurde (wie im obigen Abschnitt 1c auf Seite 135).

UMGEBUNG: Geben Sie den Umgebungsnamen ein, der für dieses SCLM-Projekt konfiguriert wurde und der die SCLM-Umgebungsvariablen enthält (wie im vorherigen Abschnitt 2a auf Seite 136).

Beispiel: SCLMDT

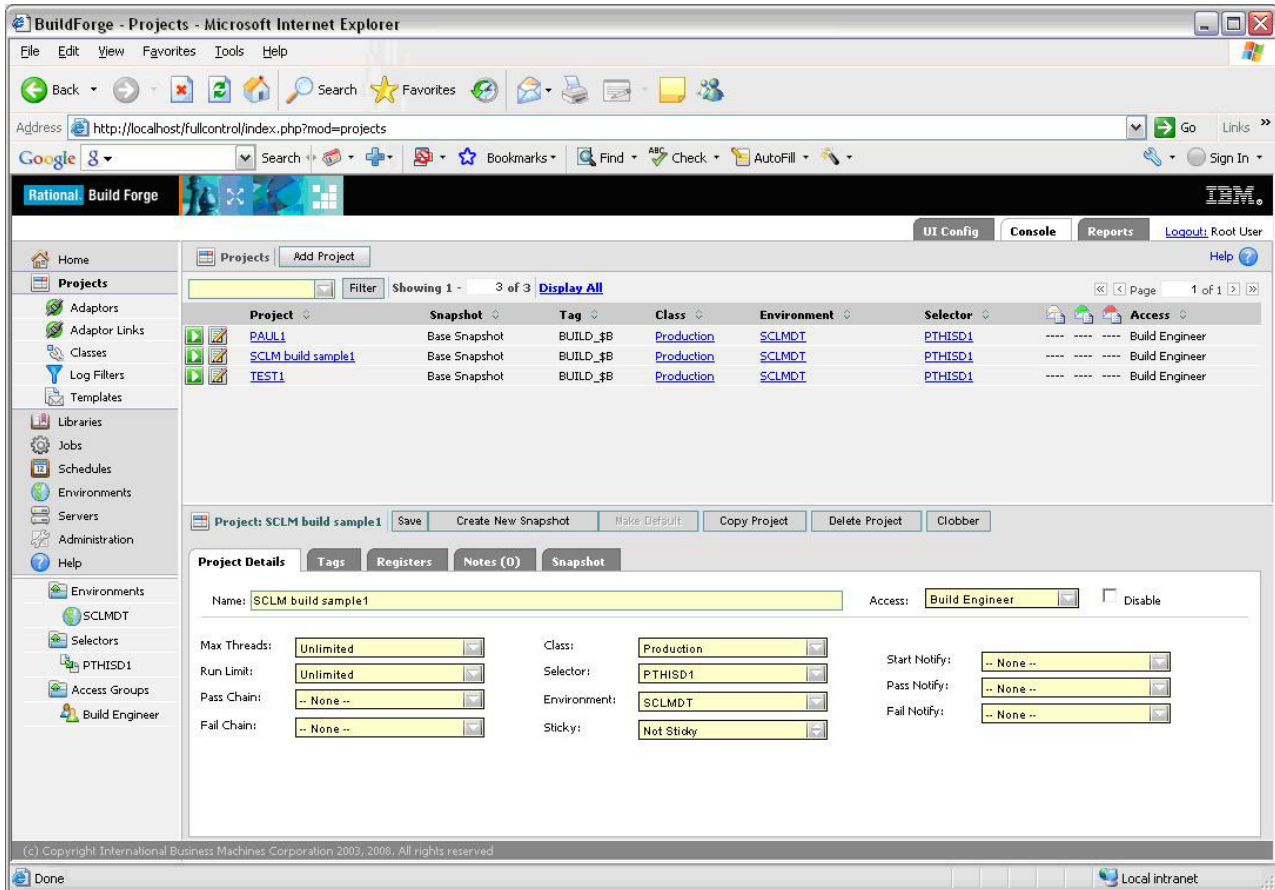


Abbildung 36. Projektmustereinrichtung für SCLM-Build (SCLM build sample1)

- b. Fügen Sie nun einen Erstellungsschritt für dieses SCLM-Projekt hinzu.
Das folgende Beispiel ist ein konfiguriertes Erstellungsbeispiel, das im Hostdatensatz SAMPLIB verteilt wurde (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

```

Abbildung 37. **** SCLM-Build sample1 ****

Konfigurieren Sie das obige Beispiel für Ihr SCLM-Projekt sowie Gruppe, Typ, Member und Buildoptionen, wie in der SCLM DT-Anwendungsprogrammierschnittstelle im Abschnitt ANHANG C für Buildfunktionen angegeben.

Fügen Sie dieses konfigurierte Beispiel zum ersten Schritt im Projekt „SCLM Build sample1“ hinzu.

Projekte -> SCLM-Build sample1 -> Schritt hinzufügen

NAME: Der Name, den Sie für diesen Schritt auswählen

BEFEHL: Fügen Sie in diesem Bereich das obige konfigurierte Beispiel (BWBBFBLD)

Alle anderen Felder können den Standardwert beibehalten. (Dieser Schritt übernimmt die Haupt-Projektumgebungseinstellungen.)

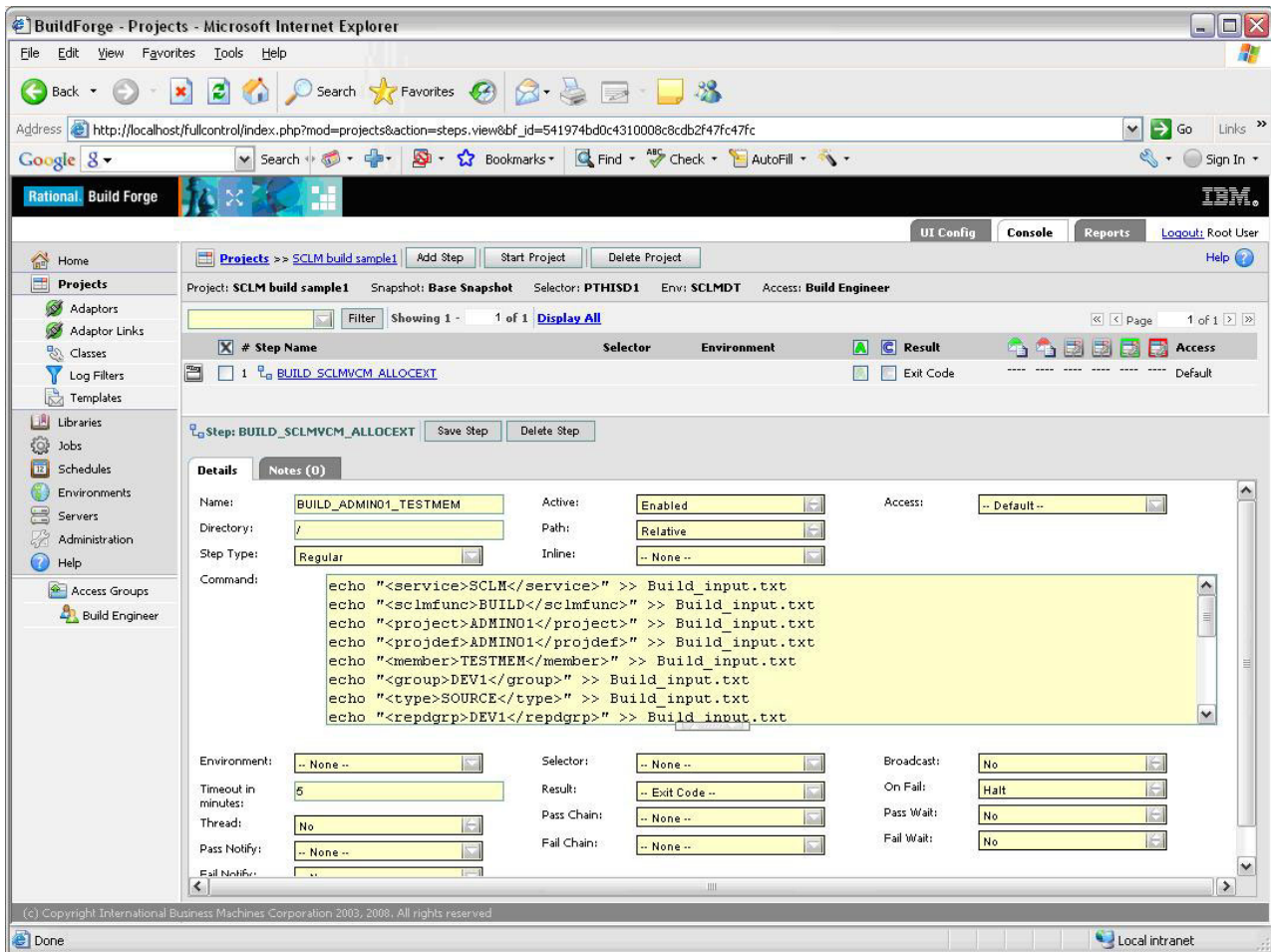


Abbildung 38. Projektmusterschritt in SCLM build sample1

- c. Führen Sie das SCLM-Buildprojekt „SCLM Build sample1“ aus.
- Klicken Sie auf **Projekte -> SCLM-Build sample1 -> Projekt starten**. Die Ausgabe vom Build wird an die Konsole **JOBS -> Build-Tag** zurückgegeben.
- Die Anforderungs- und Antwortausgabe wird ebenfalls im z/OS Unix Systems Services-Verzeichnis gespeichert, das durch den in der Serverkonfiguration angegebenen PATH bestimmt wird. Im vorherigen Beispiel für Server PTHISD1 mit `PATH=/u/IBMUUSER/BUILDFORGE` würden die Anforderungs- und Antwortdateien in folgendem Verzeichnis gespeichert werden:
- ```
/u/IBMUUSER/BUILDFORGE/SCLM_Build1_sample/BUILD_1
: >ls
Build_input.txt Build_output.txt
```
- Die obigen Anforderungs- und Antwortdateien können im angepassten Erstellungsschritt umbenannt werden.

## Beispiel für SCLM-Umstufung

Die Schritte im vorherigen Abschnitt können repliziert werden, um ein SCLM-Umstufungsprojekt in Build Forge zu erstellen. Ersetzen Sie das Erstellungsscript durch ein Beispiel für ein Umstufungsscript im Projektbereich "COMMAND". Das folgende Beispiel ist ein konfiguriertes Umstufungsbeispiel, das im Hostdatensatz SAMPLIB verteilt wurde (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 ' <prmmmsg>Y</prmmmsg>' >> Promote_input.txt
echo ' <prmmmsgds>NONE</prmmmsgds>' >> Promote_input.txt
echo ' <prmxtds>NONE</prmxtds>' >> 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

```

Abbildung 39. \*\* SCLM-Umstufung sample1 \*\*

Konfigurieren Sie das vorherige Beispiel mit Ihren Daten zu SCLM-Projekt, Gruppe, Typ, Member und Optionen zum Umstufen, so wie dies in der Funktionalität Anhang C, „SCLM Developer Toolkit-API“, auf Seite 71 zum Umstufen ausführlich beschrieben ist.



---

## Dokumentationshinweise für IBM Rational Developer for System z

© Copyright IBM Corporation 2009, 2012.

© Copyright IBM Deutschland GmbH 2009, 2012.

Die vorliegenden Informationen wurden für Produkte und Services entwickelt, die auf dem deutschen Markt angeboten werden.

Möglicherweise bietet IBM die in dieser Dokumentation beschriebenen Produkte, Services oder Funktionen in anderen Ländern nicht an. Informationen über die gegenwärtig im jeweiligen Land verfügbaren Produkte und Services sind beim zuständigen IBM Ansprechpartner erhältlich. Hinweise auf IBM Lizenzprogramme oder andere IBM Produkte bedeuten nicht, dass nur Programme, Produkte oder Services von IBM verwendet werden können. An Stelle der IBM Produkte, Programme oder Services können auch andere, ihnen äquivalente Produkte, Programme oder Services verwendet werden, solange diese keine gewerblichen oder anderen Schutzrechte der IBM verletzen. Die Verantwortung für den Betrieb von Produkten, Programmen und Services anderer Anbieter liegt beim Kunden.

Für in diesem Handbuch beschriebene Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieses Handbuchs ist keine Lizenzierung dieser Patente verbunden. Lizenzanforderungen sind schriftlich an folgende Adresse zu richten (Anfragen an diese Adresse müssen auf Englisch formuliert werden):

*IBM Director of Licensing  
IBM Europe, Middle East & Africa  
Tour Descartes  
2, avenue Gambetta  
92066 Paris La Defense  
France*

Trotz sorgfältiger Bearbeitung können technische Ungenauigkeiten oder Druckfehler in dieser Veröffentlichung nicht ausgeschlossen werden. Die hier enthaltenen Informationen werden in regelmäßigen Zeitabständen aktualisiert und als Neuausgabe veröffentlicht. IBM kann ohne weitere Mitteilung jederzeit Verbesserungen und/oder Änderungen an den in dieser Veröffentlichung beschriebenen Produkten und/oder Programmen vornehmen.

Verweise in diesen Informationen auf Websites anderer Anbieter werden lediglich als Service für den Kunden bereitgestellt und stellen keinerlei Billigung des Inhalts dieser Websites dar. Das über diese Websites verfügbare Material ist nicht Bestandteil des Materials für dieses IBM Produkt. Die Verwendung dieser Websites geschieht auf eigene Verantwortung.

Werden an IBM Informationen eingesandt, können diese beliebig verwendet werden, ohne dass eine Verpflichtung gegenüber dem Einsender entsteht.

Lizenznehmer des Programms, die Informationen zu diesem Produkt wünschen mit der Zielsetzung: (i) den Austausch von Informationen zwischen unabhängig

voneinander erstellten Programmen und anderen Programmen (einschließlich des vorliegenden Programms) sowie (ii) die gemeinsame Nutzung der ausgetauschten Informationen zu ermöglichen, wenden sich an folgende Adresse:

*Intellectual Property Dept. for Rational Software  
IBM Europe, Middle East & Africa  
5 Technology Park Drive  
Westford, MA 01886  
USA*

Die Bereitstellung dieser Informationen kann unter Umständen von bestimmten Bedingungen - in einigen Fällen auch von der Zahlung einer Gebühr - abhängig sein.

Die Lieferung des im Dokument aufgeführten Lizenzprogramms sowie des zugehörigen Lizenzmaterials erfolgt auf der Basis der IBM Rahmenvereinbarung bzw. der Allgemeinen Geschäftsbedingungen von IBM, der IBM Internationalen Nutzungsbedingungen für Programmpakete oder einer äquivalenten Vereinbarung.

Alle in diesem Dokument enthaltenen Leistungsdaten stammen aus einer kontrollierten Umgebung. Die Ergebnisse, die in anderen Betriebsumgebungen erzielt werden, können daher erheblich von den hier erzielten Ergebnissen abweichen. Einige Daten stammen möglicherweise von Systemen, deren Entwicklung noch nicht abgeschlossen ist. Eine Gewährleistung, dass diese Daten auch in allgemein verfügbaren Systemen erzielt werden, kann nicht gegeben werden. Darüber hinaus wurden einige Daten unter Umständen durch Extrapolation berechnet. Die tatsächlichen Ergebnisse können davon abweichen. Benutzer dieses Dokuments sollten die entsprechenden Daten in ihrer spezifischen Umgebung prüfen.

Alle Informationen zu Produkten anderer Anbieter stammen von den Anbietern der aufgeführten Produkte, deren veröffentlichten Ankündigungen oder anderen allgemein verfügbaren Quellen. IBM hat diese Produkte nicht getestet und kann daher keine Aussagen zu Leistung, Kompatibilität oder anderen Merkmalen machen. Fragen zu den Leistungsmerkmalen von Produkten anderer Anbieter sind an den jeweiligen Anbieter zu richten.

Aussagen über Pläne und Absichten von IBM unterliegen Änderungen oder können zurückgenommen werden und repräsentieren nur die Ziele von IBM.

Diese Veröffentlichung dient nur zu Planungszwecken. Die in dieser Veröffentlichung enthaltenen Informationen können geändert werden, bevor die beschriebenen Produkte verfügbar sind.

Diese Veröffentlichung enthält Beispiele für Daten und Berichte des alltäglichen Geschäftsablaufes. Sie sollen nur die Funktionen des Lizenzprogramms illustrieren; sie können Namen von Personen, Firmen, Marken oder Produkten enthalten. Alle diese Namen sind frei erfunden; Ähnlichkeiten mit tatsächlichen Namen und Adressen sind rein zufällig.

## **Copyright-Lizenz**

Diese Veröffentlichung enthält Beispielanwendungsprogramme, die in Quellsprache geschrieben sind und Programmiertechniken in verschiedenen Betriebsumgebungen veranschaulichen. Sie dürfen diese Beispielpprogramme kostenlos kopieren, ändern und verteilen, wenn dies zu dem Zweck geschieht, Anwendungsprogramme zu entwickeln, zu verwenden, zu vermarkten oder zu verteilen, die mit der

Anwendungsprogrammierschnittstelle für die Betriebsumgebung konform sind, für die diese Beispielprogramme geschrieben werden. Diese Beispiele wurden nicht unter allen denkbaren Bedingungen getestet. Daher kann IBM die Zuverlässigkeit, Wartungsfreundlichkeit oder Funktion dieser Programme weder zusagen noch gewährleisten. Die Beispielprogramme werden auf der Grundlage des gegenwärtigen Zustands (auf "as-is"-Basis) und ohne Gewährleistung zur Verfügung gestellt. IBM haftet nicht für Schäden, die durch die Verwendung dieser Beispielprogramme entstehen.

Kopien oder Teile der Beispielprogramme bzw. daraus abgeleiteter Code müssen folgenden Copyrightvermerk beinhalten:

© (Name Ihrer Firma) (Jahr). Teile des vorliegenden Codes wurden aus Beispielprogrammen der IBM Corp. abgeleitet. © Copyright IBM Corp. 2009, 2012.

Wird dieses Buch als Softcopy (Book) angezeigt, erscheinen keine Fotografien oder Farabbildungen.

## **Markenhinweise**

IBM, das IBM Logo und [ibm.com](http://ibm.com) sind Marken oder eingetragene Marken von International Business Machines Corp. in den USA und/oder anderen Ländern. Andere Produkt- und Servicenamen können Marken von IBM oder anderen Unternehmen sein. Eine aktuelle Liste der IBM Marken finden Sie auf der Webseite [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

Adobe, das Adobe-Logo, PostScript und das PostScript-Logo sind Marken oder eingetragene Marken der Adobe Systems Incorporated in den USA und/oder anderen Ländern.

Linux ist eine eingetragene Marke von Linus Torvalds in den USA und/oder anderen Ländern.

Windows ist eine Marke der Microsoft Corporation in den USA und/oder anderen Ländern.

UNIX ist eine eingetragene Marke von The Open Group in den USA und anderen Ländern.

Java und alle auf Java basierenden Marken und Logos sind Marken oder eingetragene Marken der Oracle Corporation und/oder ihrer verbundenen Unternehmen.

Andere Produkt- und Servicenamen können Marken von IBM oder anderen Unternehmen sein.

---

## **Copyright-Lizenz**

Diese Veröffentlichung enthält Beispielanwendungsprogramme, die in Quellsprache geschrieben sind und Programmier Techniken in verschiedenen Betriebsumgebungen veranschaulichen. Sie dürfen diese Beispielprogramme kostenlos kopieren, ändern und verteilen, wenn dies zu dem Zweck geschieht, Anwendungsprogramme zu entwickeln, zu verwenden, zu vermarkten oder zu verteilen, die mit der Anwendungsprogrammierschnittstelle für die Betriebsumgebung konform sind, für die diese Beispielprogramme geschrieben werden. Diese Beispiele wurden nicht unter allen denkbaren Bedingungen getestet. Daher kann IBM die Zuverlässigkeit, Wartungsfreundlichkeit oder Funktion dieser Programme weder zusagen noch

gewährleisten. Die Beispielpprogramme werden ohne Wartung (auf "as-is"-Basis) und ohne jegliche Gewährleistung zur Verfügung gestellt. IBM haftet nicht für Schäden, die durch die Verwendung dieser Beispielpprogramme entstehen.

---

## Markenhinweise

IBM, das IBM Logo und [ibm.com](http://ibm.com) sind Marken oder eingetragene Marken von International Business Machines Corp. in den USA und/oder anderen Ländern. Andere Produkt- und Servicenamen können Marken von IBM oder anderen Unternehmen sein. Eine aktuelle Liste der IBM Marken finden auf der Webseite [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).

Rational ist eine Marke der International Business Machines Corporation und der Rational Software Corporation in den USA und/oder anderen Ländern.

Intel und Pentium sind Marken der Intel Corporation in den USA und/oder anderen Ländern.

Microsoft, Windows und das Windows-Logo sind Marken oder eingetragene Marken der Microsoft Corporation in den USA und/oder anderen Ländern.

Java und alle auf Java basierenden Marken und Logos sind Marken oder eingetragene Marken von Sun Microsystems, Inc. in den USA und/oder anderen Ländern.

UNIX ist eine eingetragene Marke von The Open Group in den USA und anderen Ländern.

---

# Antwort

IBM Rational Developer for System z  
8.5.0  
SCLMDT Administrator

IBM Form SC12-4344-03

Anregungen zur Verbesserung und Ergänzung dieser Veröffentlichung nehmen wir gerne entgegen. Bitte informieren Sie uns über Fehler, ungenaue Darstellungen oder andere Mängel.

Zur Klärung technischer Fragen sowie zu Liefermöglichkeiten und Preisen wenden Sie sich bitte entweder an Ihre IBM Geschäftsstelle, Ihren IBM Geschäftspartner oder Ihren Händler.

**Unsere Telefonauskunft "HALLO IBM" (Telefonnr.: 0180 3 313233) steht Ihnen ebenfalls zur Klärung allgemeiner Fragen zur Verfügung.**

Kommentare:

Danke für Ihre Bemühungen.

Als Brief an die Postanschrift auf der Rückseite dieses Formulars

\_\_\_\_\_  
Name

\_\_\_\_\_  
Adresse

\_\_\_\_\_  
Firma oder Organisation

\_\_\_\_\_  
Rufnummer

\_\_\_\_\_  
E-Mail-Adresse

IBM  
Corporation  
Building 501  
P.O Box 12195  
Research Triangle Park, NC  
USA







Gedruckt in Deutschland

SC12-4344-03

