

SCLM Developer Toolkit



インストールおよびカスタマイズ・ガイド

バージョン 3.1

SCLM Developer Toolkit



インストールおよびカスタマイズ・ガイド

バージョン 3.1

— お願い —

本書および本書で紹介する製品をご使用になる前に、111 ページの『特記事項』に記載されている情報をお読みください。

| 本書は、IBM SCLM Developer Toolkit、バージョン 3 リリース 1、プログラム番号 5655-S72 に適用されます。また、新しい版で指示があるまでは後続のリリースにも適用されます。必ず、製品のレベルにあった正しい版をご使用ください。

| IBM 発行のマニュアルに関する情報のページ

| <http://www.ibm.com/jp/manuals/>

| こちらから、日本語版および英語版のオンライン・ライブラリーをご利用いただけます。また、マニュアルに関するご意見やご感想を、上記ページよりお送りください。今後の参考にさせていただきます。

| (URL は、変更になる場合があります)

| お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

| 原 典： SC23-8504-01
| SCLM Developer Toolkit
| Installation and Customization Guide
| Version 3.1

| 発 行： 日本アイ・ビー・エム株式会社

| 担 当： ナショナル・ランゲージ・サポート

| 第2版第1刷 2007.12

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

目次

図	v
---	---

表	vii
---	-----

本書について	ix
--------	----

本書の対象読者	ix
---------	----

旧版からの変更点	ix
----------	----

本書の追加情報の入手先	x
-------------	---

資料	x
----	---

ソフトコピー資料	xi
----------	----

IBM Systems Center 資料	xi
-----------------------	----

インストールの概要	xiii
-----------	------

TCP/IP の考慮事項	xiii
--------------	------

SMP/E のインストール	xiv
---------------	-----

バッチ・ジョブの考慮事項	xiv
--------------	-----

個別の SCLM インストール	xiv
-----------------	-----

第 1 部 SCLM Developer Toolkit のインストール

第 1 章 z/OS 上での SCLM Developer Toolkit のインストールとカスタマイズ

ステップ 1: z/OS ソフトウェア要件の確認	4
--------------------------	---

ステップ 2: 構成に関する考慮事項	5
--------------------	---

ステップ 3: セットアップ JCL の実行	5
------------------------	---

ステップ 4: SCLM Developer Toolkit 構成ファイルのカスタマイズ	7
--	---

ISPF 構成ファイルのカスタマイズ	7
--------------------	---

TRANSLATE 構成ファイルのカスタマイズ	8
-------------------------	---

TRANSLATE.conf ファイル内の設定のオーバーライド	11
---------------------------------	----

ステップ 5a: SCLM Developer Toolkit HTTP Server の構成	11
---	----

HTTP サーバー構成ファイルのカスタマイズ	12
------------------------	----

HTTP サーバー環境ファイルのカスタマイズ	13
------------------------	----

HTTP サーバー JCL/STARTED TASK のカスタマイズ	13
------------------------------------	----

ステップ 5b: リモート・システム・エクスプローラーの構成	16
--------------------------------	----

RSE 環境ファイルのカスタマイズ	16
-------------------	----

RSE 環境セットアップ・スクリプトのカスタマイズ	18
---------------------------	----

RSE 環境ファイルのアクティブ化	19
-------------------	----

ステップ 6: ロング・ショート・ネーム・テーブル VSAM ファイルの構成	19
--	----

ステップ 7: Ant のインストールとカスタマイズ	21
----------------------------	----

ステップ 8a: HTTP のインストールとカスタマイズが正しく行われたことを確認するための IVP の実行	23
--	----

HTTP サーバーへの接続のテスト	27
-------------------	----

ステップ 8b: RSE のインストールとカスタマイズが正しく行われたことを確認するための IVP の実行	28
---	----

第 2 章 Eclipse ベースのクライアントの PC へのインストール

インストールの準備	31
-----------	----

メディア要件	31
--------	----

ハードウェア要件およびソフトウェア要件	31
---------------------	----

SCLM Developer Toolkit のインストール	32
--------------------------------	----

ステップ 1. CD または電子イメージからのインストール	32
-------------------------------	----

ステップ 2. SCLM Developer Toolkit のインストール	33
--	----

第 2 部 SCLM Developer Toolkit のカスタマイズ

第 3 章 SCLM 管理者のための SCLM のカスタマイズ

JAVA/J2EE をサポートするための言語変換プログラム	37
-------------------------------	----

JAVA/J2EE ビルドの要約	38
------------------	----

生成される JAVA/J2EE ビルド・オブジェクト	39
----------------------------	----

SCLM の言語定義	40
------------	----

SCLM タイプ	42
----------	----

SCLM メンバー形式	44
-------------	----

JAVA/J2EE Ant XML のビルド・スケルトン	52
------------------------------	----

J2EE プロジェクトの SCLM へのマッピング	53
---------------------------	----

推奨される J2EE プロジェクトの SCLM へのマッピング	54
---------------------------------	----

SCLM Developer Toolkit のデプロイメント	56
---------------------------------	----

WebSphere Application Server (WAS) のデプロイメント	57
---	----

SCLM から Unix システム・サービスへのデプロイメント	58
---------------------------------	----

セキュア・デプロイメント	58
--------------	----

他のデプロイメント・オプション	59
-----------------	----

ASCII または EBCDIC のストレージ・オプション	59
-------------------------------	----

ASCII/EBCDIC 言語変換プログラム	60
------------------------	----

\$GLOBAL メンバー	61
---------------	----

SITE とプロジェクト固有のオプション	62
----------------------	----

TRANSLATE.conf のオーバーライドの併用例	68
-----------------------------	----

BIDIPROP のオーバーライドの併用例	69
-----------------------	----

第 4 章 SCLM セキュリティ

ビルド/プロモート/デプロイ・セキュリティ・フラグおよびプロセス・フロー	71
--------------------------------------	----

セキュリティ規則および代理ユーザー ID	72
----------------------	----

ビルド規則形式	72
---------	----

プロモート規則形式	72
-----------	----

デプロイ規則形式	73
SAF/RACF ビルド、プロモート、デプロイ、および プロファイルに関する規則	73

第 5 章 CRON 開始のビルドおよびプロ

モート	77
STEPLIB および PATH に関する要件	78
CRON ビルド・ジョブの実行	79
CRON ビルド・ジョブのサンプル	79

付録 A. SCLM の概要 83

SCLM の概念.	83
ファイル命名.	83
タイプ	83
言語.	84
SCLM のプロパティ.	84
SCLM プロジェクト構造.	84
ARCHDEF.	85
JAVA/J2EE の概念.	85

付録 B. SQLJ Support. 89

SQL とは ?	89
DB2 とは ?	89
JDBC とは ?	89
SQLJ とは ?	90
JDBC と SQLJ の比較.	90
直列化プロファイルとは ?	91
DBRM とは ?	92

SQLJ プログラム準備	92
変換.	93
例:	94
カスタマイズ.	94
バインド	95
SCLM DT タイプと変換プログラム	95
ビルド・プロセスの調整	96
ビルド・スクリプトの調整	96

付録 C. ロング/ショート・ネーム変換テ ーブル 103

SCLM 変換プログラムの技術要約	103
単一ロング/ショート・ネーム・レコード処理.	104
FINDLONG 処理	104
FINDSHORT 処理.	105
TRANSLATE 処理.	105
複数ロング/ショート・ネーム・レコード処理.	105
IMPORT 処理	106
MIGRATE 処理.	106

参考文献. 109

特記事項.	111
商標	112

用語集 113

索引 117



1.	サンプル ISPF.conf	8	19.	サンプル EJB アプリケーション (EJB)	
2.	TRANSLATE.conf キーワードのユーザー	10		ARCHDEF	47
3.	Pass および Exec ディレクティブ	15	20.	サンプル EAR アプリケーション (EAR)	
4.	サンプル rsed.envars	18		ARCHDEF	48
5.	サンプル・ロング/ショート変換 VSAM ファイル JCL	20	21.	J2EE Ant ビルド・スクリプト	48
6.	HTTP サーバーのログオン・プロンプト	24	22.	J2EE ビルド・スクリプトの JAR サンプル	50
7.	ホストのインストールとカスタマイズのウェルカム画面	25	23.	J2EE ビルド・スクリプトの WAR サンプル	51
8.	検証応答の例 (パート 1)	26	24.	J2EE ビルド・スクリプトの EJB サンプル	51
9.	検証応答の例 (パート 2)	26	25.	J2EE ビルド・スクリプトの EAR サンプル	52
10.	検証応答の例 (パート 3)	27	26.	複数タイプ	54
11.	サーバー接続が成功した場合のメッセージ	28	27.	SCLM ビルド階層	55
12.	ディレクトリー変更コマンド	29	28.	SCLM 言語変換プログラムと ASCII/EBCDIC	60
13.	IVP 検証応答	30	29.	サンプルの SITE 固有の SCLM プロジェクト設定	64
14.	サンプルの変換プログラム	37	30.	サンプルのプロジェクト固有の SCLM プロジェクト設定	65
15.	サンプル Jar アプリケーション (JAR)		31.	サンプル CRON メンバー	78
	ARCHDEF	39	32.	サンプル CRON ビルド Exec	80
16.	J2EE ビルド・スクリプトの JAR サンプル	39	33.	サンプル・ビルド・パラメーター・ファイル	81
17.	サンプル Jar アプリケーション (JAR)		34.	複数タイプ	84
	ARCHDEF	45	35.	変換モジュール呼び出し用のサンプル REXX	107
18.	サンプル Web アプリケーション (WAR)				
	ARCHDEF	46			

表

1. 資料	x	7. 必須オペレーティング・システム	32
2. IBM Systems Center 資料	xii	8. ユーザー定義変数	49
3. httpd.conf のカスタマイズ	12	9. \$GLOBAL 変数	61
4. httpd.env のカスタマイズ	13	10. SITE/プロジェクト・オプション	65
5. RSE 環境ファイルのカスタマイズ	16	11. ロング/ショート・ネーム変換パラメーター	104
6. 必須ハードウェア	31		

本書について

本書は標準の z/OS® インストール手順と UNIX® システム・サービス、および IBM z/OS HTTP サーバーの構成を結合する IBM® SCLM Developer Toolkit 製品の構成手順について記載しています。また、Windows® での Eclipse クライアントのインストール方法についても解説します。

以下に本書で使用する個々の名称を示します。

- IBM z/OS HTTP サーバーを、「HTTP サーバー」と呼びます。
- Rational® Developer for System z を「Rational Developer」と呼びます。

「IBM SCLM Developer Toolkit」を、「Developer Toolkit」または「SCLMDT」と呼ぶことがあります。

本書の対象読者

本書のパート 1 は IBM SCLM Developer Toolkit のインストール、構成、および管理を担当するシステム・プログラマー向けです。本書では読者が、z/OS UNIX システム・サービス環境 (z/OS UNIX システム・サービスのファイル・システム)、構造、z/OS UNIX システム・サービスをサポートするために必要なセキュリティ・ソフトウェア (例: リソース・アクセス制御機能 [RACF®]) プロファイル、開始タスク (またはインストールされたセキュリティ製品と等価の製品)、および HTTP サーバー (使用している場合) に精通していることを前提としています。

またパート 1 に、クライアント・コンポーネントを Windows にインストールする場合の章を用意しました。

本書のパート 2 は SCLM Developer Toolkit で使用される任意の SCLM プロジェクトの管理者向け情報を記載しています。これには Java® および z/OS UNIX システム・サービスのコンポーネント言語のほか、従来の SCLM プロジェクトを使用したプロジェクトが含まれます。これらの管理者は、z/OS UNIX システム・サービス環境および z/OS UNIX システム・サービスのファイル・システム構造、REXX スクリプト、Java コンパイラー、SCLM プロジェクトおよび言語定義などにも精通している必要があります。

旧版からの変更点

第 1 版 (2007.9)

- SQLJ Support が追加され、SQLJ ソース・コードが SCLM に格納され、ビルドできるようになりました。これにより、プロジェクトの Jar ファイルのビルドおよび作成を行う際に、既存の J2EE ARCHDEF に .sqlj ソース・ファイルをインクルード (必要な場合は既存の Java ソースと混合) する機能が提供されます。
- インストールに必要な各種ステップを明確にするための追加変更。
- 付録のメッセージを削除。
- ドキュメンテーションへのその他の小さい変更。

変更点にはリビジョン・バーが付与されています。

本書の追加情報の入手先

本書では必要に応じて他のブックの情報を参照する場合がありますが、この場合、該当するブックのショート・タイトルが使用されます。z/OS 関連製品のすべてのブックの完全なタイトルとオーダー番号については、「z/OS 情報ロードマップ」を参照してください。IBM 資料の入手については、IBM 担当員またはお近くの IBM のブランチ・オフィスまで直接ご連絡ください。

また、カスタマー・サポートはフリーダイヤル (1-800-879-2755) で、月から金、午前 6:30 から午後 5:00 (山岳部時間帯) にて承ります (米国のみ)。この電話番号では、以下も可能です。

- IBM 資料のオーダーまたはお問い合わせ
- ソフトウェアの製造または配送に関するお問い合わせ
- ソフトウェア更新のオーダーがより迅速かつ便利になるプログラム再オーダー・フォームの利用

資料

表 1. 資料

本書で使用されている ショート・タイトル	資料のタイトル	オーダー番号
HTTP Server ガイド	<i>Domino Go Webserver for OS/390 概説およびインストール</i>	SD88-7879
Rational Developer for System z Host Configuration Guide	<i>Rational Developer for System z Host Configuration Guide</i>	SC31-6930
Communications Server for z/OS V1R2 TCP/IP Implementation Guide	<i>Communications Server for z/OS V1R2 TCP/IP Implementation Guide</i>	SG24-5227
z/OS UNIX システム・サービス 計画	<i>z/OS UNIX システム・サービス 計画</i>	GA88-8639
z/OS UNIX System Services Messages	<i>z/OS UNIX System Services Messages and Codes</i>	GA22-7807
z/OS UNIX システム・サービス コマンド解説書	<i>z/OS UNIX システム・サービス コマンド解説書</i>	SA88-8641
IBM Ported Tools for z/OS User' Guide	<i>IBM Ported Tools for z/OS User' Guide</i>	SA22-7985
SCLM プロジェクト管理者ガイド	対話式システム生産性向上機能 (ISPF) SCLM プロジェクト管理および開発者のガイド z/OS	SC88-8960
SCLM Developer Toolkit: Program Directory (GI10-3352-00)	<i>SCLM Developer Toolkit: Program Directory</i>	GI10-3352

表 1. 資料 (続き)

本書で使用されている ショート・タイトル	資料のタイトル	オーダー番号
SCLM 解説書	対話式システム生産性向上機能 (ISPF) ソフトウェア構成およびライ ブラリー管理機能 (SCLM) リファレ ンス z/OS	SC88-8961

ソフトコピー資料

z/OS のライブラリーは z/OS コレクション・キット、SK2T-6700 からご利用いただけます。このソフトコピー・コレクションには、z/OS とそれに関連するライセンスを受けていない製品ブックが含まれています。CD-ROM コレクションには、ソフトコピー・ブックを読む際にご利用いただける IBM Library Reader (ILR)TM、というプログラムが付属します。

ソフトコピー z/OS 資料は、Web でも参照することができます。Adobe Acrobat Reader を使用して、参照および印刷が可能な、PDF 版の z/OS 関連資料は、以下の URL から入手できます。

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

「Library」を選択します。

Rational Developer for System z、Eclipse、および SCLM Advanced Edition に関する情報は、以下の URL で参照できます。

- <http://www.ibm.com/software/awdtools/devzseries> - Rational Developer for System z Web サイト
- <http://www.eclipse.org> - Eclipse ホーム・ページ
- <http://www.ibm.com/software/awdtools/scslmsuite>

IBM Systems Center 資料

IBM Systems Center では z/OS UNIX システム・サービスのセットアップと使用に役立つ、RedbooksTM もご提供しています。これらの資料のオーダーは、IBM の担当者にお問い合わせいただくか、以下の URL から Web ブラウザーでもご覧いただけます。

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

これらのブックは正式なレビュー、あるいは技術的な検証を終えたものではありませんが、現在の製品に関する情報 (資料が発行された時点)、および z/OS にまつわる広範なトピックに関する価値ある情報が掲載されています。これらは別にオーダーしてください。本ブックの選択リストは以下のとおりです。

本書の追加情報の入手先

表 2. IBM Systems Center 資料

資料のタイトル	オーダー番号	コメント
<i>P/390, R/390, S/390 Integrated Server: OS/390 New User's Cookbook</i>	SG24-4757-01	タイトルからは分かりにくいですが、本書はシステム・プログラマーを対象としており、z/OS UNIX システム・サービス環境の考慮点について説明しています。
<i>Debugging UNIX System Services, Lotus Domino, Novell Network Services</i>	SG24-5613-00	z/OS UNIX システム・サービス環境の概要と、セットアップおよび問題分析のヒント、推奨される方法が掲載されています。
<i>OS/390 e-business Infrastructure: IBM HTTP Server V5.1 for OS/390</i>	SG24-5603-00	一般的な Web サーバーの概要と、OS/390® サーバー固有の詳細、セットアップとカスタマイズについてのヒントなどが掲載されています。
<i>ABCs of z/OS System Programming Vol 9</i>	SG24-6989-00	システム・プログラマー向けに UNIX システム・サービスについて解説しています。
e-business Enablement Cookbook for OS/390 Volumes 1、2、および 3	SG24-5664-00 SG24-5981-00 SG24-5980-00	

インストールの概要

本書は SCLM Developer Toolkit のすべてのコンポーネントのインストール手順について解説します。SCLM Developer Toolkit には z/OS ホスト・コンポーネント、およびクライアント・コンポーネントの 2 種類のインストール・コンポーネントが含まれています。クライアント・コンポーネントは Eclipse プラグインで、既存の Eclipse にインストールするか、独立型 SCLM Developer Toolkit のインストールに同梱される Eclipse とともにインストールするか、あるいは、Rational Developer for System z がインストールされていれば、インストール中にプラグインをインストールすることができます。Eclipse はオープンで拡張可能な統合開発環境 (IDE) で、本書の中では Eclipse IDE と呼びます。

インストール手順は標準の z/OS インストール手順、z/OS UNIX システム・サービスのファイル・セットアップ、HTTP サーバーの構成、および (Rational Developer for System z のプラグインの一部としてインストールされている場合) RSE 構成などをインストールすることで行います。

本書の構成を以下に示します。

- 3 ページの『第 1 章 z/OS 上での SCLM Developer Toolkit のインストールとカスタマイズ』
- 37 ページの『第 3 章 SCLM 管理者のための SCLM のカスタマイズ』
- 71 ページの『第 4 章 SCLM セキュリティー』
- 77 ページの『第 5 章 CRON 開始のビルドおよびプロモート』

付録は 83 ページから開始します。

SCLM と JAVA/J2EE の概念に関する概要は、83 ページの『付録 A. SCLM の概要』を参照してください。

TCP/IP の考慮事項

SCLM Developer Toolkit をセットアップする場合は、サイトの TCP/IP のインストール状況も考慮する必要があります。SCLM Developer Toolkit が HTTP サーバーか、Rational Developer for System z のプラグインとしてインストールされているのであれば、RSE 接続を使用することもできます。

SCLM Developer Toolkit が HTTP サーバーを使用するように構成されている場合は、TCPIP.DATA ファイルを有効にする必要があります。z/OS UNIX システム・サービス計画でこのファイルを見つける方法を説明しています。

ただし、このファイルのロケーションの定義に別の方法を使用している場合 (System Resolver など)、SCLM Developer Toolkit のサーバーのジョブとして、//SYSTCPD DD カードを追加する必要があります。TCP ポートを使用可能にし、ポート番号を予約することを推奨します。System Resolver と TCPIP.DATA について詳しくは、以下の資料を参照してください。

- z/OS UNIX システム・サービス 計画
- Communications Server for z/OS V1R2 TCP/IP Implementation Guide

SMP/E のインストール

本書では SCLM Developer Toolkit の実装について説明しません。むしろ、製品を正しく構成するまでの過程をガイドすることを目的としています。本書では SCLM Developer Toolkit のシステム修正変更プログラム/拡張 (SMP/E) のインストールが完了していることを前提としています。SCLM Developer Toolkit の SMP/E の手順は、IBM SCLM Developer Toolkit のプログラム・ディレクトリーに含まれています。SCLM Developer Toolkit のインストールを開始する前に、SMP/E のインストールに対して、以下のアクションを実行することを推奨します。

- ルートの z/OS UNIX システム・サービスのファイル・システムが読み取り専用であること
- SCLM Developer Toolkit のディレクトリーが独立したファイル・システムとしてセットアップされ、ルートのファイル・システムに /usr/lpp/SCLMDT としてマウントされていること (または SCLM Developer Toolkit のルート・ディレクトリーとして適切な名前)。

これらの推奨内容は「z/OS UNIX システム・サービス 計画」ガイドで指定された内容に準拠します。詳しくは、『ルート z/OS UNIX システム・サービス・ファイル・システムをマウントして実行する方法の決定』という章を参照してください。

SMP/E のインストールが正常に完了した後は、その後の章に記載されている指示に従って、z/OS 側のインストールとカスタマイズを完了し、PC に Eclipse ベースのクライアントをインストールします。

バッチ・ジョブの考慮事項

SCLM Developer Toolkit は SDSF を使用して、ジョブの完了状況とジョブの出力を検索します。JES2 または SDSF を所有していないユーザーのために、SCLM Developer Toolkit で OUTPUT コマンドを使用するためのサポートが追加されました。z/OS に同梱される OUTPUT コマンドは、ログオンしたユーザー ID で開始されるジョブ出力を取り出すことができます。OUTPUT 機能を完全に使用したい場合、提供される TSO/E エグジット IKJEFF53 を変更して、自分の所有するジョブ出力であるが、自分のユーザー ID で開始されていないジョブ出力も取り出せるように変更する必要性が生ずる場合があります。このエグジットについては、「z/OS TSO/E カスタマイズ」ガイドを参照してください。

個別の SCLM インストール

本書では SCLM 製品の実装およびロードについて取り扱いません。

第 1 部 SCLM Developer Toolkit のインストール

第 1 章 z/OS 上での SCLM Developer Toolkit の

インストールとカスタマイズ	3
ステップ 1: z/OS ソフトウェア要件の確認	4
ステップ 2: 構成に関する考慮事項	5
ステップ 3: セットアップ JCL の実行	5
ステップ 4: SCLM Developer Toolkit 構成ファイルの カスタマイズ	7
ISPF 構成ファイルのカスタマイズ	7
TRANSLATE 構成ファイルのカスタマイズ	8
TRANSLATE 構成ファイルの例	10
TRANSLATE.conf ファイル内の設定のオーバーラ イド	11
ステップ 5a: SCLM Developer Toolkit HTTP Server の構成	11
HTTP サーバー構成ファイルのカスタマイズ	12
HTTP サーバー環境ファイルのカスタマイズ	13
HTTP サーバー JCL/STARTED TASK のカスタ マイズ	13
SCLM サポートのための既存の HTTP サーバ ーのカスタマイズ	14
SCLM Developer Toolkit HTTP サーバーの始 動	15
HTTP サーバーのトレースの使用可能化	15
ステップ 5b: リモート・システム・エクスプローラ の構成	16
RSE 環境ファイルのカスタマイズ	16
RSE 環境セットアップ・スクリプトのカスタマイ ズ	18
RSE 環境ファイルのアクティブ化	19
ステップ 6: ロング/ショート・ネーム・テーブル VSAM ファイルの構成	19
ステップ 7: Ant のインストールとカスタマイズ	21
ステップ 8a: HTTP のインストールとカスタマイズ が正しく行われたことを確認するための IVP の実行	23
HTTP サーバーへの接続のテスト	27
ステップ 8b: RSE のインストールとカスタマイズが 正しく行われたことを確認するための IVP の実行	28

第 2 章 Eclipse ベースのクライアントの PC への

インストール	31
インストールの準備	31
メディア要件	31
ハードウェア要件およびソフトウェア要件	31
SCLM Developer Toolkit の前提条件	31
SCLM Developer Toolkit のインストール	32
ステップ 1. CD または電子イメージからのイン ストール	32
ステップ 2. SCLM Developer Toolkit のインスト ール	33

第 1 章 z/OS 上での SCLM Developer Toolkit のインストールとカスタマイズ

この章では、お客様のホスト z/OS システムに SCLM Developer Toolkit をインストールするために必要なタスクのリストを示します。また、SCLM Developer Toolkit の Eclipse クライアント・プラグインのインストールに関連するタスクも示します。

次のステップを使用して、SCLM Developer Toolkit をホスト z/OS システムにインストールします。

- 4 ページの『ステップ 1: z/OS ソフトウェア要件の確認』
- 5 ページの『ステップ 2: 構成に関する考慮事項』
- 5 ページの『ステップ 3: セットアップ JCL の実行』
- 7 ページの『ステップ 4: SCLM Developer Toolkit 構成ファイルのカスタマイズ』
- 11 ページの『ステップ 5a: SCLM Developer Toolkit HTTP Server の構成』、または
- 16 ページの『ステップ 5b: リモート・システム・エクスプローラーの構成』
- 19 ページの『ステップ 6: ロング/ショート・ネーム・テーブル VSAM ファイルの構成』
- 21 ページの『ステップ 7: Ant のインストールとカスタマイズ』
- 23 ページの『ステップ 8a: HTTP のインストールとカスタマイズが正しく行われたことを確認するための IVP の実行』、または
- 28 ページの『ステップ 8b: RSE のインストールとカスタマイズが正しく行われたことを確認するための IVP の実行』

これらのステップは、z/OS システム・プログラマーが実施しなければならないものです。

SCLM Developer Toolkit は、HTTP サーバーを使用するか、RSE (リモート・システム・エクスプローラー) 接続を使用して、z/OS ホストに接続できます。SCLM Developer Toolkit が Rational Developer for System z のもとで実行されている場合、通信メカニズムは RSE です。SCLM Developer Toolkit を RSE 通信用に構成する方法については、16 ページの『ステップ 5b: リモート・システム・エクスプローラーの構成』を参照してください。SCLM Developer Toolkit が Rational® Application Developer (RAD) など他の Eclipse インストール環境で実行されている場合、通信メカニズムは HTTP です。これについては、11 ページの『ステップ 5a: SCLM Developer Toolkit HTTP Server の構成』を参照してください。

SCLM Developer Toolkit を特定の SCLM プロジェクト用に構成する方法については、37 ページの『第 3 章 SCLM 管理者のための SCLM のカスタマイズ』を参照してください。この章には、下記のものについてさらにカスタマイズするための情報が記載されています。

- サイトまたは SCLM プロジェクト固有の設定

- JAVA/J2EE 言語サポート

ステップ 1: z/OS ソフトウェア要件の確認

SCLM Developer Toolkit を正常にインストールするには、ユーザーのインストール環境で次のシステム要件が満たされている必要があります。

- 以下の PTF が適用されている z/OS V1.7 以上:
 - ログ出力のための適切なメッセージ機能を可能にし、追加の BUILD サービス処理情報を提供する APAR OA20345 に対応する ISPF PTF。
 - SCLMINFO 機能強化に関する APAR OA21104 に対応し、構文チェックのためのモード情報を構築する ISPF/SCLM PTF。
 - SCLMINFO サービスを強化する APAR OA16924 に対応する ISPF PTF。
 - SCLM で長いファイル名を持つファイルを保管する場合は、ISPF APAR OA11426 に対応し、ロング/ショート・ファイル名 (ロング/ショート・ネーム) の変換をサポートする PTF。これは z/OS V1.8 では不要です。
 - セキュアなビルド、プロモート、およびデプロイを使用する場合は (71 ページの『第 4 章 SCLM セキュリティー』を参照)、APAR OA16804 に対応する ISPF/SCLM PTF。
- z/OS UNIX システム・サービス (および TCP/IP)
- REXX ランタイム・ライブラリーまたは REXX 代替ライブラリー (REXX.**.SEAGLPA、または REXX.**.SEAGALT)。REXX Library も REXX Alternate Library もインストールされていない場合は、REXX Alternate Library をインストールすることで REXX Library 要件を満たすことができます。REXX Alternate Library は <http://www-1.ibm.com/support/> から無料でダウンロードできます。

REXX ランタイム・ライブラリー REXX.**.SEAGLPA (これがインストールされていない場合は代替ライブラリー REXX.**.SEAGALT) は、まだ LINKLIST データ・セットとして定義されていない場合は、HTTP サーバーの JCL で STEPLIB に追加する必要があります。お客様のサイトでは REXX とは異なる高位修飾子を使用できます。

これらのデータ・セットは APF 許可されている必要があります。

- UNIX システム・サービスにインストールされている Ant ランタイム (JAVA/J2EE ビルドを実行する場合)。 <http://Ant.apache.org/> からダウンロード可能です。

21 ページの『ステップ 7: Ant のインストールとカスタマイズ』を参照してください。

- 次のいずれかのバージョンの z/OS Java。

- Java V1.4.0 (5655-I56) 以降
- Java V1.4.2 (5655-M30) 以降
- Java V5 - 31bit (5655-N98) 以降
- Java V5 - 64bit (5655-N99) 以降

MEMLIMIT の設定

z/OS は領域サイズを使用して、プログラムの実行に使用可能なストレージ容量を決定します。64 ビット製品の場合、実行中の SDK には

256 MB 以上が割り当て可能です。この設定には MEMLIMIT パラメーターを使用します。MEMLIMIT パラメーターについては、「Limiting Storage use above the bar in z/Architecture」を参照してください。

ステップ 2: 構成に関する考慮事項

システムを構成する前に以下の事項について考慮してください。

1. SCLM Developer Toolkit を使用する各ユーザーは、有効なゼロ以外の UID、ホーム・ディレクトリー、およびシェル・コマンドを指定する RACF OMVS セグメント (または同等なもの) を定義する必要があります。

これを指定しないと、Eclipse IDE を使用して SCLM Developer Toolkit にログインできません。

2. BPXPRMxx parmlib メンバーの MAXPROCUSER を少なくとも 50 に設定してください。これは、以下のコマンドを使用して、動的に (次の IPL まで) 検査および設定できます (「z/OS MVS システム・コマンド (SA88-8593)」を参照)。

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

低すぎる値を設定すると、SCLM 変換に加え、場合によってはその他のアクティビティーも失敗するおそれがあります。

3. LINKLIST の一部であるデータ・セットに BWB* モジュールを保存しておくことをお勧めします。あるいは、このデータ・セットを次のいずれかに追加しても構いません。
 - HTTP サーバーの STEPLIB (11 ページの『ステップ 5a: SCLM Developer Toolkit HTTP Server の構成』を参照)、または
 - RSE サーバーの rsed.envvars にある STEPLIB ステートメント (16 ページの『ステップ 5b: リモート・システム・エクスプローラーの構成』を参照)。
4. 64 ビット・バージョンの Java では、2 GB 境界を超えて使用可能なストレージを使用できます。ただし、デフォルトではこのストレージは使用できません。SMFPRMxx parmlib メンバーの MEMLIMIT を NOLIMIT に設定するか、デフォルトの 0 M より大きい値に設定してください。詳しくは、「z/OS MVS 初期設定およびチューニング解説書」(SA88-8564) を参照してください。

ステップ 3: セットアップ JCL の実行

1. SBWBSAMP データ・セットに保管されている BWBINST1 をカスタマイズして、実行してください。

このメンバー内のカスタマイズ指示に従ってください。

このジョブは以下のタスクを実行します。

- z/OS UNIX システム・サービス・ファイル・システム内の指定したディレクトリーに、CONFIG、LOGS、および WORKAREA ディレクトリーを作成する。

セットアップ JCL の実行

- z/OS UNIX システム・サービス・ファイル・システム内の CONFIG ディレクトリーに PROJECT ディレクトリーを作成する。PROJECT ディレクトリーの使用について詳しくは、62 ページの『SITE とプロジェクト固有のオプション』を参照してください。
- サンプル HTTP サーバー構成ファイルを、SBWBSAMP ライブラリー内のメンバー BWBHTTPC および BWBHTTPE から CONFIG ディレクトリー内のファイル httpd.conf および httpd.env にコピーする。これら 2 つのファイルはカスタマイズが必要です。11 ページの『ステップ 5a: SCLM Developer Toolkit HTTP Server の構成』を参照してください。
- サンプル ISPF 構成テーブルを、SBWBSAMP ライブラリー内のメンバー BWBISPFC から CONFIG ディレクトリー内のファイル ISPF.conf にコピーする。このファイルはカスタマイズが必要です。7 ページの『ステップ 4: SCLM Developer Toolkit 構成ファイルのカスタマイズ』を参照してください。
- サンプル変換構成テーブルを、SBWBSAMP ライブラリー内のメンバー BWBTRANC から CONFIG ディレクトリー内のファイル TRANSLATE.conf にコピーする。このファイルはカスタマイズが必要です。7 ページの『ステップ 4: SCLM Developer Toolkit 構成ファイルのカスタマイズ』を参照してください。

これらの構成ファイルの推奨基底ディレクトリーは /etc/SCLMDT です。このジョブを実行する前に、このディレクトリーの SCLMDT までの部分が存在していなければなりません。

WORKAREA および LOGS ディレクトリーに対する読み取りおよび書き込みアクセス権限が必要です。デフォルトでは、/var/SCLMDT/WORKAREA および /var/SCLMDT/LOGS に対する読み取りおよび書き込みアクセス権限が必要です。WORKAREA は、ファイルの転送、ASCII/EBCDIC 変換、および JAVA/J2EE ビルドのために使用されます。

/var/SCLMDT/WORKAREA/userid/* という形式の一時ディレクトリーが Developer Toolkit の使用時に作成されます。実行する機能のタイプに応じて、WORKAREA ディレクトリー内のユーザー ID のディレクトリーの下に、次のディレクトリーが作成されます。

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

注:

一部の一時セッション・ファイルが、/tmp ディレクトリーに作成されることがあります。すべてのユーザーが /tmp ディレクトリーへの書き込み権限を持つことを確認します。

SCLM Developer Toolkit は、WORKAREA ディレクトリーに作成した一時ファイルをすべて除去します。ところが、処理中に通信エラーが発生した場合など、一時出力が残される場合があります。このため、WORKAREA および LOGS ディレクトリーを適宜整理することをお勧めします。

これを行うには、OMVS で以下のコマンドを使用します。

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

/var/SCLMDT/WORKAREA の保存先は、どこに WORKAREA ディレクトリーを作成するか依存します。

これはすべてのエントリーを除去するもので、同じ手順を LOGS ディレクトリーに対しても使用することができます。

ステップ 4: SCLM Developer Toolkit 構成ファイルのカスタマイズ

ファイル ISPF.conf および TRANSLATE.conf は、デフォルトのディレクトリー・ロケーション /etc/SCLMDT/CONFIG に保存され、さらにカスタマイズを必要とする場合があります。

ISPF 構成ファイルのカスタマイズ

SCLM Developer Toolkit で ISPF および SCLM サービスを実行するには、有効な ISPF 環境を確立する必要があります。ISPF 構成には、ユーザーが TSO/ISPF 環境セッションを確立するために、SCLM Developer Toolkit の必要な割り振りが含まれています。

CONFIG ディレクトリーにある ISPF 構成ファイル ISPF.conf をカスタマイズして、ISPF データ・セット割り振りに関するサイト要件を満たす必要があります。提供サンプル ISPF.conf にはカスタマイズを完了するための説明が含まれており、ユーザー・サイトは次のことを実行できます。

- SCLM Developer Toolkit の動作に必要な最小 ISPF データ・セット割り振りを組み込む。これは、3270 エミュレーターで ISPF を起動する最小 ISPF データ・セットを割り振ることを意味します。提供サンプルでは、これらは isp.sisp* データ・セットとして指定されています。場合によっては、これらをサイト指定のデータ・セット名に変更する必要があります。
- DDNAME ファイル割り振りをさらに追加するか、追加の ISPF データ・セットを連結する。
- カスタマー定義の割り振り実行可能ファイル (exec) を起動して、さらにプロジェクトまたはユーザー ID によるデータ・セット割り振りを提供する。サンプル exec は SBWBSAMP ライブラリーのメンバー BWBISPF2 で提供されます。
- SCLM Developer Toolkit は標準の割り振り済み ISPF/SCLM スケルトン (例えば、FLMLIBS) を使用するため、必要なスケルトン・ライブラリーが確実に ISPF.conf の ISPSLIB DD に割り振られるようにする。

各 ISPF DD ごとの割り振りは、各データ・セットをコンマで区切った単一行に指定する必要があります。コメント行は、その行をアスタリスク (*) で開始することで追加できます。例については、次のサンプル ISPF.conf を参照してください。

```
* REQUIRED:
* Below is the minimum requirements for ISPF allocation.
* Change the default ISPF data set names below to match your host site.
* Add additional dsn concatenations on same line and separate by comma.
* Order of data sets listed is search order in concatenation.
* The sclmdt loadlib data set is required to be added to the ISPLLIB
* concatenation to access the JAVA/J2EE SCLM Language translators.
* Change BWB.SBWBLOAD to the appropriate data set where the
* BWBxxx load modules are stored.
*
* The libraries beginning BZZ.* are for the Breeze product and are
* included to show how multiple data sets are added to the concatenations.
* These should be removed if the Breeze product is not installed.

sysproc=ISP.SISPCLIB,BZZ.SBZZCLIB
ispmlib=ISP.SISPMENU
isptlib=ISP.SISPTEU
ispplib=ISP.SISPPENU
ispslib=BZZ.SBZZSENU,ISP.SISPSLIB
ispllib=BWB.SBWBLOAD,BZZ.SBZZLOAD
```

図 1. サンプル *ISPF.conf*

TRANSLATE 構成ファイルのカスタマイズ

CONFIG ディレクトリーに保管されている TRANSLATE 構成ファイル TRANSLATE.conf を確認してください。デフォルトの ASCII=ISO8859-1 および EBCDIC=IBM-1047 以外の場合で、別の ASCII/EBCDIC 変換コード・ページが必要な場合は、サンプルに含まれる指示に従ってください。

TRANSLATE.conf ファイルは、コードが SCLM 内でどのように保管されるかを指定するキーワードを提供します。構成ファイルには、ファイルがその言語定義に応じてどのようにホストに転送されるかを決定するキーワードが含まれています。特定のキーワードによって、その言語タイプのファイルがバイナリーであり、転送されてかつ保管されるか、またはテキスト・ベースのソースが ASCII から EBCDIC へのデフォルト変換なしに ASCII フォーマットのままであるかが決まります。

さらに、SCLM 言語定義は、ロング・ネーム・ファイルを SCLM で保管するのにふさわしい有効なショート・ホスト名に変換するかどうかを制御します。ロング・ネームからショート・ネームへのマッピングは、SCLM ロング/ショート・ネーム変換ファイルによって制御されます。

注: ロング・ネームからショート・ネームへの変換またはバイナリー転送言語定義(あるいはその両方)を決定するための指針として、デフォルトの言語定義が提供されています。

TRANSLATE.conf ファイル内で有効なキーワードは次のとおりです。

キーワード	説明
-------	----

CODEPAGE	変換時に使用する ASCII および EBCDIC コード・ページを決定します。
-----------------	--

形式:

- CODEPAGE ASCII = ISO8859-1
- CODEPAGE EBCDIC = IBM-1047

転送するファイルの変換方法を決定するには、SCLM Developer Toolkit の ASCII と EBCDIC の両方に対して CODEPAGE キーワードが必要です。

TRANVRLS

インストール済み DFSMS のレベルが 1.3 以降の場合、変換テーブルの VSAM データ・セットがシステム全体で共用されることを SCLM で許可するかどうかを示します。デフォルトは NO です。

SCLM は VSAM RLS (Record Level Sharing) を使用して VSAM データ・セットの共用を可能にします。共用環境での VSAM データ・セットの保全性を維持するために、VSAM データ・セットは RLS に対して割り振る必要があります、RLS をサポートするあらゆるハードウェアとソフトウェアがシステムに装備されている必要があります。変換テーブルは、RLS 使用のための適切なストレージ・クラスとともに定義する必要があります。(ハードウェアおよびソフトウェア要件については、DFSMS 資料を参照してください。)

TRANVRLS = YES

ショート・ネームまたはロング・ネームの変換テーブルが、VSAM RLS を使用するシステム間共用のために定義されることを指定します。

TRANVRLS =NO

ショート・ネームまたはロング・ネームの変換テーブルが、VSAM RLS を使用するシステム間共用に定義されないことを指定します。これはデフォルトです。

TRANLANG

ホストへの ASCII/EBCDIC 変換を必要としない SCLM 言語タイプを決定します (ファイルはバイナリー転送されます)。

ファイルが Eclipse クライアント内の ASCII テキストの場合、そのファイルが SCLM に追加されても ASCII コード・ページ内にあるままとなります。

形式:

- TRANLANG JAVABIN
- TRANLANG DOC
- TRANLANG JPEG

上記の例では、これらの言語について SCLM でダミーの言語変換プログラムがセットアップされます。SCLM 言語変換プログラムについて詳しくは、37 ページの『第 3 章 SCLM 管理者のための SCLM のカスタマイズ』を参照してください。

LONGLANG

ロング・ネームからショート・ネームへの変換を必要とする SCLM 言語タイプを決定します。ロング・ネームからショート・ネームへの変換とは、クライアント上のロング・ネーム・ファイル (ディレクトリー・パッケージ構造を含む) を 8 文字の有効なホスト・メンバー名にマップし、この変換したホストのショート・ネームを使用してファイルを SCLM で保管することを意味します。

形式:

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

SCLM 言語が LONGLANG キーワードで指定されていない場合、クライアント・ファイルは既にホストのショート・ネーム形式 (8 文字以下) であるとみなされ、そのまま保管されます。

注: コメント行は、その行をアスタリスク (*) で開始することで追加できます。

```
*
* ----- CODEPAGE SECTION -----
*
CODEPAGE ASCII = ISO8859-1
CODEPAGE EBCDIC = IBM-1047
*
* ----- ASCII to EBCDIC TRANSLATION SECTION -----
*
TRANLANG JAVABIN
TRANLANG J2EEBIN
TRANLANG J2EEOBJ
TRANLANG TEXTBIN
TRANLANG BINARY
TRANLANG DOC
TRANLANG XLS
*
* ----- LONG/SHORT NAME TRANSLATION SECTION -----
*
LONGLANG JAVA
LONGLANG SQLJ
LONGLANG J2EEPART
LONGLANG JAVABIN
LONGLANG J2EEBIN
LONGLANG J2EEOBJ
LONGLANG DOC
LONGLANG XLS
*
```

図 2. *TRANSLATE.conf* キーワードのユーザー

図 2 を含む例については、『TRANSLATE 構成ファイルの例』を参照してください。

TRANSLATE 構成ファイルの例

上記のように、TRANSLATE 構成ファイルは多くのことを制御します。そのうちの 2 つは次のとおりです。

- 一部が ASCII から EBCDIC へ変換されるかどうか。
- パーツの名前をロング・ネームから 8 文字のショート・ネームに変換するかどうか。

この構成ファイルの初期セットアップ方法の決定に、この 2 つの設定をどのように役立てるかについて、以下に例を表示します。

例: SCLM で保管したい Word 文書が多数あります。この場合は、このタイプのファイルをメインフレームで編集することはできません。したがって、これらのファイルを EBCDIC に変換しても意味がなく、ASCII で保管する必要があります。

- SBWBSAMP ライブラリー内のサンプル BWBTRANJ に基づいて SCLM 言語変換プログラムを作成します。これは BINARY と呼ぶこともできますが (保管する方法がそうであるため)、より具体的にいったければ DOC と呼びます。
- このファイルはワークステーション上では InstallGuide.doc などのロング・ネームを持つため、これを保管する SCLM PDS では、生成されたショート・ネームにマップされるようにする必要があります。したがって、ファイルの呼び方に応じて、BINARY または DOC 言語の LONGLANG エントリーを作成します。例えば、LONGLANG BINARY などです。
- ファイルの呼び方に応じて、BINARY または DOC 言語の TRANLANG エントリーを追加します。例えば、TRANLANG BINARY などです。
- Word 文書は、SCLM でチェックアウトされると、EBCDIC から ASCII に変換されないまま Eclipse ワークスペースまで転送されます。チェックアウト後、Word が起動され、そのファイルで作業ができます。変更が完了すると、Word 文書はチェックインして SCLM に戻され、ASCII から EBCDIC への変換は行われません。

TRANSLATE.conf ファイル内の設定のオーバーライド

TRANSLATE.conf ファイル内で設定されている値を、SITE および SCLM プロジェクト・レベルでオーバーライドすることができます。このフィーチャーについては、62 ページの『SITE とプロジェクト固有のオプション』を参照してください。

ステップ 5a: SCLM Developer Toolkit HTTP Server の構成

このセクションでは、HTTP サーバー経由で z/OS ホストと通信するように SCLM Developer Toolkit を構成する場合の、HTTP サーバーのセットアップとカスタマイズについて説明します。HTTP サーバーが使用されるのは、SCLM Developer Toolkit が Rational Developer for System z とともにインストールされているときではなく、Rational Application Developer (RAD) など他の Eclipse インストール環境のもとにあるときです。

SCLM Developer Toolkit を Rational Developer for System z とともに使用する場合は、RSE (リモート・システム・エクスプローラー) が通信メカニズムとして使用されます。これについては、16 ページの『ステップ 5b: リモート・システム・エクスプローラーの構成』を参照してください。

HTTP サーバーを、このインターフェースをサポートする専用の Web サーバーとして使用することをお勧めします。ただし、必要な SCLM/HTTP 構成ディレクティブを既存の HTTP サーバーに組み込むこともできます。14 ページの『SCLM サポートのための既存の HTTP サーバーのカスタマイズ』を参照してください。

デフォルトでは、SCLM/HTTP サーバーはポート 80 を使用するように構成されます。ただし、カスタマイズ時に別の適切な専用ポートを選ぶこともできます (1024 以上。これより低いポート番号はシステム内部での使用のために予約されています)。

デフォルトのポート番号を変更する場合は、HTTP サーバーの JCL で変更する必要があります。

サンプル・セットアップでは、エンド・ユーザーがこのインターフェースを使用してホスト・システムにアクセスするとき、有効な z/OS ユーザー ID およびパスワードを指定する必要があります。

注: IBM HTTP Web サーバーの構成について詳しくは、以下の IBM マニュアルを参照してください。

- IBM HTTP Server for OS/390 HTTP Server 計画、インストールおよび使用の手引き (SD88-7879)
- OS/390 e-business Infrastructure: IBM HTTP Server V5.1 for OS/390 (SG24-5603-00)

以下のセクションでは、提供サンプルのカスタマイズと HTTP サーバーの始動の手順を概説します。

デフォルトでは、HTTP サーバー構成ファイルおよび環境ファイルは、SCLM Developer Toolkit CONFIG ディレクトリーに保管されます。これらのファイルは、必要に応じて別のユーザー・ディレクトリーまたは既存のサーバー構成および環境ファイルにコピーすることができます。すべての場合において、HTTP サーバー開始タスクは、適切なディレクトリーを反映するようにカスタマイズする必要があります。

HTTP サーバー構成ファイルのカスタマイズ

必要な変更を加えるために、構成ファイルにある手順に従って、サンプル HTTP 構成ファイル `httpd.conf` (セットアップ・ジョブ `BWBINST1` で指定した CONFIG ディレクトリーに保管済み) をカスタマイズします。以下のディレクティブを検討する必要があります。

表 3. `httpd.conf` のカスタマイズ

ディレクティブ	変更の説明
Port	ポート 80 のままにしておくか、または HTTP サーバーの JCL で指定した有効なポート番号に変更します。ポート番号を予約することをお勧めします。
Protection	SCLMDTWB を HTTP サーバー・ジョブの名前に変更します。Protection ディレクティブの使用法については、「IBM HTTP Server for OS/390 HTTP Server 計画、インストールおよび使用の手引き」を参照してください。
PidFile AccessLog ErrorLog	/var/SCLMDT を適切なパスに変更します (デフォルトとは異なるパスが選択された場合)。
Pass および Exec ディレクティブ	/var/SCLMDT を適切なパスに変更します (デフォルトとは異なるパスが選択された場合)。 /usr/lpp/SCLMDT を bin インストール・ディレクトリーの適切なパスに変更します (デフォルトとは異なるパスが選択された場合)。

表 3. httpd.conf のカスタマイズ (続き)

ディレクティブ	変更の説明
SCLM Developer Toolkit 内の標準外コード・ページ変換	<p>標準デフォルト (IBM-1047/ISO8859-1) とは異なる ASCII/EBCDIC コード・ページ変換を必要とする場合は、HTTP サーバーの httpd.conf ファイルに下記のパラメーターをコーディングしてください。</p> <pre>DefaultFsCp ebcdic-codepage DefaultNetCp ascii-codepage</pre> <p>例えば、日本語変換の場合、必要なコード・ページは次のとおりです。</p> <pre>DefaultFsCp IBM-939 DefaultNetCp IBM-932C</pre>

HTTP サーバー環境ファイルのカスタマイズ

必要な変更を加えるために、環境ファイルにある手順に従って、サンプル HTTP 環境変数ファイル httpd.env (インストール・ジョブ BWBINST1 で指定した インストール・ディレクトリーに保管済み) をカスタマイズします。

表 4. httpd.env のカスタマイズ

ディレクティブ	変更の説明
PATH	PATH ディレクティブに正しい Java パス・ディレクトリーがあることを確認します。また、「.」で表示される現行ディレクトリーが、パスに含まれていることを確認します。(例えば、/bin:./usr/sbin:/usr/lpp/internet/bin:/usr/lpp/internet/sbin:/usr/lpp/java/J1.4/bin)。
CGI_DTWORk	<p>このディレクティブは、一時ファイルのために使用される WORKAREA ディレクトリー・パスを決定します。デフォルトは次のとおりです。</p> <pre>CGI_DTWORk=/var/SCLMDT</pre>
CGI_DTCONF	<p>このディレクティブは、構成ファイルがある CONFIG ディレクトリー・パスを決定します。デフォルトは次のとおりです。</p> <pre>CGI_DTCONF=/etc/SCLMDT</pre>
CGI_TRANTABLE	<p>このディレクティブは、ショート・ネームからロング・ネームへの変換に使用される変換テーブルの名前を決定します。この VSAM ファイルについては、19 ページの『ステップ 6: ロング/ショート・ネーム・テーブル VSAM ファイルの構成』を参照してください。デフォルトは次のとおりです。</p> <pre>CGI_TRANTABLE=BWB.LSTRANS.FILE</pre>

HTTP サーバー JCL/STARTED TASK のカスタマイズ

HTTP サーバーを実行するには、以下のタスクを実行する必要があります。

1. サンプル・バッチ・ジョブ BWBSVR をインストール済みのサンプル・ライブラリー SBWBSAMP から JCL ライブラリーまたは PROCLIB データ・セットにコピーし、サンプル内の指示に従ってサイト固有の標準に合わせてカスタマイズします。
2. CAPS OFF コマンドを発行して、大/小文字を区別する値が大文字に変更されないようにします。
3. この HTTP サーバー・ジョブは開始タスクにすることをお勧めしますが、HTTP サーバー JCL をテストするために独立型ジョブとして実行することもできます。
4. フォアグラウンド Java ビルドを実行する場合は、HTTP サーバー・ジョブで 512M の領域サイズを使用することをお勧めします。
5. RACF に関する考慮事項:
 - HTTP サーバーに割り当てられたユーザー ID の RACF OMVS セグメントを作成します。
 - ユーザー ID を所有する HTTP サーバーは、/usr/lpp/internet/sbin ファイルに対する実行アクセス権限と、httpd.conf ファイルで参照されている LOGS ディレクトリーに対する読み取り/書き込みアクセス権限が必要です。これは /var/SCLMDT/LOGS のデフォルトです。
6. 使用されるデフォルト・ポートは 80 です。これを特定の専用ポートに変更する場合は、httpd.conf 構成ファイル内のポート番号を開始タスク JCL 内のポート番号と一致するように変更することもあります。
7. BWB* モジュールが LINKLIST に保管されていない場合、これらのモジュールを含むロード・ライブラリーを指定するように STEPLIB を編集します。デフォルトでは、これらのモジュールは SBWBLOAD ライブラリーに存在します。
8. モジュール BWBTSOW は、APF の許可済みロード・ライブラリーに保管する必要があります。
9. ホスト上に REXX/370 ランタイム環境を確立するか、REXX/370 代替ライブラリーを使用します。

注: HTTP サーバーに割り当てられたユーザー ID は、FACILITY クラス内の BPX.SERVER リソースに対して読み取り権限を持っている必要があります。このリソースが定義されていない場合は、UID 0 が必要です。

SCLM サポートのための既存の HTTP サーバーのカスタマイズ

オプションとして SCLM Developer Toolkit サポートを既存の HTTP サーバーに組み込むことを選択する場合は、下記の指示に従います。

次の pass/exec ディレクティブを httpd.conf 構成ファイルに追加します。

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

図 3. Pass および Exec ディレクティブ

注: 次のようにサンプルをカスタマイズする必要があります。

- /usr/lpp/SCLMDT を bin インストール・ディレクトリーで置き換えます。
- /var を WORKAREA ディレクトリーで置き換えます。

SCLM Developer Toolkit HTTP サーバーの始動

ジョブを実行依頼して Web サーバーを始動するか、ジョブが開始タスク・プロシージャの場合は z/OS コンソールから次のコマンドを入力します。

```
Start server_proc_name
```

ここで server_proc_name は STC 名です。

(プロシージャが PROCLIB データ・セット内のメンバーであることを確認してください。)

HTTP サーバーが正常に初期化されたことの確認: サーバーのジョブ・ログには次のメッセージが含まれているはずです。

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

HTTP サーバーのトレースの使用可能化

HTTP サーバーでトレースを使用可能にするには、サーバーの JCL PARM ステートメントを変更してトレース・レベルの 1 つを含むようにします。例えば、次のようにします。

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

提供されるトレース・レベルは次のとおりです。

- v トレース (第 1 レベルの場合)
- vv トレース (第 2 レベルの場合)
- mtv (第 3 レベルの場合)
- debug (最大トレースの場合)

トレースについて詳しくは、「IBM HTTP Server for OS/390 HTTP Server 計画、インストールおよび使用の手引き」を参照してください。

注: トレースはパフォーマンスに影響を及ぼすので、IBM 担当員から助言があった場合にのみ実行するようにしてください。

ステップ 5b: リモート・システム・エクスプローラーの構成

このセクションでは、クライアントが z/OS ホスト上の SCLM にアクセスするために使用できるリモート・システム・エクスプローラー (RSE) のセットアップとカスタマイズについて説明します。RSE は、SCLM Developer Toolkit が Rational Developer for System z とともにインストールされているときに使用されます。

SCLM Developer Toolkit を Rational Developer for System z とともに使用していない場合は、z/OS ホストとの通信に使用されるメカニズムは HTTP となります。これについては、11 ページの『ステップ 5a: SCLM Developer Toolkit HTTP Server の構成』を参照してください。

Rational Developer for System z の RSE コンポーネントのインストールと構成については、「*Program Directory for IBM Rational Developer for System z*」(G111-8298-00) および「*IBM Rational Developer for System z Host Configuration Guide*」(SC31-6930-02) で説明されています。このセクションでは、RSE 接続を通じて SCLM Developer Toolkit が機能できるようにする SCLM Developer Toolkit 固有の設定を追加する手順について説明します。

その前にまず、Rational Developer for System z の RSE コンポーネントのカスタマイズ・ディレクトリーに含まれているファイルを変更しなければならないので、このディレクトリーを知っておく必要があります。

RSE 環境ファイルのカスタマイズ

RSE 接続で使用する rsed.envvars ファイルを探します。このファイルは、デフォルトでは /usr/lpp/wd4z/rse/lib に存在します。このファイルの末尾に、RSE 環境変数を含む SCLM Developer Toolkit メンバーをコピーします。このメンバーは BWBRSED で、インストール済みのサンプル・ライブラリー SBWBSAMP に入っています。

SCLM Developer Toolkit で使用される次の環境変数は、ベースの RSE カスタマイズ時に rsed.envvars に定義されている必要があります。

```
_CMDSERV_BASE_HOME
_CMDSERV_CONF_HOME
_CMDSERV_WORK_HOME
```

下の表にある指示に従って各環境変数をカスタマイズします。

表 5. RSE 環境ファイルのカスタマイズ

ディレクティブ	変更の説明
CGI_DTCONF	/CONFIG ディレクトリーの基本パスを決定します (構成ファイルはこのディレクトリーに保管されます)。デフォルトは次のとおりです。 CGI_DTCONF=\$_CMDSERV_CONF_HOME
CGI_DTWORK	/WORKAREA ディレクトリーの基本パスを決定します (作業域ファイルはこのディレクトリーに保管されます)。デフォルトは次のとおりです。 CGI_DTWORK=\$_CMDSERV_WORK_HOME

表 5. RSE 環境ファイルのカスタマイズ (続き)

ディレクティブ	変更の説明
CGI_TRANTABLE	<p>ショート・ネームからロング・ネームへの変換に使用される変換テーブルの名前を指定します。この VSAM ファイルについては、19 ページの『ステップ 6: ロング/ショート・ネーム・テーブル VSAM ファイルの構成』を参照してください。デフォルトは次のとおりです。</p> <p>CGI_TRANTABLE=BWB.LSTRANS.FILE</p>
STEPLIB	<p>MVS ロード・モジュールの実行元の linklist または steplib を決定します。SCLMDT の BWB ロード・モジュールが steplib から検出される必要がある場合、デフォルトの rsed.envvars ファイルに含まれている次のディレクティブが適切にセットアップされていることを確認してください。</p> <ul style="list-style-type: none"> • <code>\$_CMDSESV_BASE_LOAD=BWB.SBWBLOAD</code> • <code>STEPLIB=\$_CMDSESV_BASE_LOAD</code> <p>Bizz のロード・モジュールなど、他のロード・モジュールが steplib からアクセスされる必要がある場合、rsed.envvars の最後に次の行を追加します。</p> <ul style="list-style-type: none"> • 以前に rsed.envvars で定義した、最後の steplib ディレクティブが <code>STEPLIB=NONE</code> の場合 <p>STEPLIB=BZZ.SBZZLOAD</p> • 以前に rsed.envvars で定義した、最後の steplib ディレクティブが <code>STEPLIB=NONE</code> でない場合 <p>STEPLIB=\$STEPLIB:BZZ.SBZZLOAD</p> <p>追加のデータ・セットはコロン (:) で区切られます。</p>
_SCLM_DT	<p>/bin インストール・ディレクトリーのパスを指定します。デフォルトは次のとおりです。</p> <p>\$_CMDSESV_BASE_HOME</p>
_SCLM_J2EPUT	<p>プット要求に対する /WORKAREA ディレクトリーのパスを指定します。デフォルトは次のとおりです。</p> <p>\$CGI_DTWORK/WORKAREA</p> <p>これは、CGI_DTWORK のデフォルト値を使用して /var/SCLMDT/WORKAREA に解決されます。</p>
_SCLM_DWGET	<p>GET 要求に対する /WORKAREA ディレクトリーのパスを指定します。デフォルトは次のとおりです。</p> <p>\$CGI_DTWORK/WORKAREA</p> <p>これは、CGI_DTWORK のデフォルト値を使用して /var/SCLMDT/WORKAREA に解決されます。</p>
_SCLM_DWTRANSFER	<p>転送要求に対する /WORKAREA ディレクトリーのパスを指定します。デフォルトは次のとおりです。</p> <p>\$CGI_DTWORK/WORKAREA</p> <p>これは、CGI_DTWORK のデフォルト値を使用して /var/SCLMDT/WORKAREA に解決されます。</p>

表 5. RSE 環境ファイルのカスタマイズ (続き)

ディレクティブ	変更の説明
_SCLM_BASE	<p>その他すべての要求に対する /WORKAREA ディレクトリーのパスを指定します。デフォルトは次のとおりです。</p> <p>\$CGI_DTWORk/WORKAREA</p> <p>これは、CGI_DTWORk のデフォルト値を使用して /var/SCLMDT/WORKAREA に解決されます。</p>
_SCLM_BWBCALL	<p>BWBCALL および BWBCALLR スクリプトのロケーションを指定します。デフォルトは次のとおりです。</p> <p>\$_SCLM_DT/bin/BWBCALL</p> <p>これは、_SCLM_DT のデフォルト値を使用して /usr/lpp/SCLMDT/bin/BWBCALL に解決されます。</p>
_BPX_SHAREAS=YES	<p>spawn によって作成された子プロセスを、特定の条件下で、親と同じアドレス・スペースで実行することを指定します。この値は常に次のようになります。</p> <p>_BPX_SHAREAS=YES</p>

コメント行は、その行をハッシュ (#) で開始することで追加できます。下記のサンプル `rsed.envvars` の追加部分を参照してください。

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

図 4. サンプル `rsed.envvars`

RSE 環境セットアップ・スクリプトのカスタマイズ

RSE 自身には複数の接続方式 (デーモンで直接接続する方法と REXEC または SSH で開始されるスクリプトを使用して接続する方法) があります。RExec/SSH 接続方式を使用する場合は、`setup.env.zseries` (RSE 環境変数をセットアップするスクリプト) を変更する必要があります。このスクリプトは `rsed.envvars` と同じロケーション (デフォルト `/usr/lpp/wd4z/rse/lib`) に保管されています。ファイルの末尾に、RSE 環境エクスポート・ステートメントを含む `SCLMDT` メンバーをコピーします。このメンバーは `BWBRExec` で、インストール済みのサンプル・ライブラリー `SBWBSAMP` に入っています。

RSE 環境ファイルのアクティブ化

SCLM Developer Toolkit の変数を取得するには、次のようにすることをお勧めします。

- クライアント・マシンの RSE 接続をすべて切断する。
- すべてのクライアント・マシンで Rational Developer for System z を閉じる。
- INETD を停止して始動する。これを行うには UID 0 および BPX.DAEMON アクセス権が必要です。

注: すべてのクライアントを切断後、新規の変数が使用されていない場合、許可されたユーザー ID を使用して INETD を停止し、開始してください。この場合、「*IBM Rational Developer for System z Host Configuration Guide*」(SC31-6930-02) に説明されているとおり、BPX.DAEMON とおそらく UID 0 アクセス権が必要です。

ステップ 6: ロング/ショート・ネーム・テーブル VSAM ファイルの構成

SCLM Developer Toolkit では、ロング・ネーム・ファイル (8 文字を超える名前または大/小文字混合の名前を持つファイル) を SCLM に保管することができます。これは、長いファイル名から SCLM で使用される 8 文字のメンバー名へのマッピングを含む VSAM ファイルを使用して達成されます。

z/OS V1.8 より前のバージョンの場合、この機能は APAR OA11426 に対応する基本 ISPF/SCLM PTF を介して提供されます。この PTF を適用してからでなければ、ロング・ネームからショート・ネームへの変換は使用できません。z/OS V1.8 以上の場合は、この PTF を適用する必要はありません。

この SCLM Developer Toolkit のインストールでこの機能を使用する場合は、この機能を提供する PTF をインストールする必要があります。そうすれば、ロング・ネームからショート・ネームへのマッピングを含む VSAM ファイルを割り振ることができます。これを行うために、次のサンプル JCL (ISPF サンプル・ライブラリー ISP.SISPSAMP のメンバー FLM02LST に存在する) を変更して実行依頼できます。

このファイルに対して更新権限が必要です。

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

図 5. サンプル・ロング/ショート変換 VSAM ファイル JCL (1/2)

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

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

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

図 5. サンプル・ロング/ショート変換 VSAM ファイル JCL (2/2)

ロング/ショート・ネーム変換プロセスについて詳しくは、103 ページの『付録 C. ロング/ショート・ネーム変換テーブル』を参照してください。

ステップ 7: Ant のインストールとカスタマイズ

このステップは、SCLM で JAVA/J2EE ビルド・サポートを使用する場合に必要となります。

Ant は、<http://ant.apache.org/> から無料でダウンロードできます。 Ant のテキスト・ファイルとスクリプトは ASCII フォーマットで配布されますが、UNIX システム・

Ant のインストールとカスタマイズ

サービスの z/OS 上で実行するためには ASCII/EBCDIC 変換が必要となります。SCLM Developer Toolkit SBWBSAMP ライブラリーのサンプル・メンバー BWBTRANT にはサンプル変換スクリプトが提供されています。また、サンプル・メンバー BWBCPANT には、この変換スクリプトを適切な Ant ディレクトリーにコピーするサンプル・コピー・ジョブが提供されています。以下の手順に従って Ant を z/OS に実装します。

1. 最新の Ant 圧縮ファイルをバイナリー・フォーマットで z/OS UNIX システム・サービス・ファイル・システムにダウンロードし、適切なディレクトリーに unzip します。.zip バージョンの Ant をダウンロードすることを推奨します。接尾部の形式が tar.gz または tar.bz2 のバージョンを unzip すると、z/OS で問題を起こす可能性があるからです。z/OS で unzip する場合は、JAR 抽出コマンドの `jar -xf ANTfile.zip` を使用します。jar コマンドを使用するには、Java bin ディレクトリーがローカルの z/OS USS パスにある必要があります。そうでない場合は、コマンドを `java bin` ロケーション (例えば、`/usr/lpp/java/J1.4/bin/jar -xf ANTfile.zip`) で完全修飾してください。
2. Ant インストール・ディレクトリーを含むようにインストール・コピー・メンバー BWBCPANT をカスタマイズし、変換スクリプトをそのディレクトリーにコピーするジョブを実行します (サンプル・メンバー BWBCPANT に含まれている指示を検討します)。
3. z/OS Unix システム・サービスでファイルをリストするためにどのようなツール (例えば OMVS または ISHELL) を使用した場合でも、変換が正常に実行されたことを確認するために、ANT ディレクトリーの中の README ファイルなどのテキスト・ファイルを確認してください。次にこのファイルを OBROWSE または OMVS などの任意のブラウザーで確認してください。ファイルが読み取り可能であれば、変換は正常に行われています。
4. Ant インストール・ディレクトリーの下にあるすべてのファイルについて、すべてのユーザーが読み取って実行できるようにファイルのアクセス権を変更します。

次に例を示します。

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

5. Ant を使用する前に、z/OS UNIX システム・サービスの環境変数 `JAVA_HOME` と `ANT_HOME` を設定します。

注: ANT で指定した `JAVA_HOME` は、Java/J2EE プロジェクトでコンパイル時に使用される Java バージョンになり、`$GLOBAL` ファイルの `JAVA_BIN` 変数セットをオーバーライドします。

- a. `JAVA_HOME` は Java ホーム・ディレクトリーを指定するために必要です。次に例を示します。

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

- b. `ANT_HOME` は Ant インストール・ディレクトリーを指定するために必要です。次に例を示します。

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

6. Ant はディレクトリー `/etc` で Ant 構成ファイルを探すので、`ant.conf` という名前の構成ファイルを作成し、変数 `JAVA_HOME` と `ANT_HOME` を追加することをお勧めします。つまり、ファイル `/etc/ant.conf` で次のように設定します。

```

JAVA_HOME=/usr/lpp/java/IBM/J1.4
ANT_HOME=/u/antdirectory/Ant/apache-Ant.1.6.2

```

サンプル `ant.conf` ファイルは、インストール済みのサンプル・ライブラリー `SBWBSAMP` のメンバー `BWBANTC` で提供されます。このファイルはファイル `ant.conf` として `/etc/` ディレクトリーにコピーできるので、`JAVA_HOME` および `ANT_HOME` 変数はインストールに適切な値に変更されます。

注: 上記のディレクトリー・パスは単なるサンプル・ディレクトリー・パスに過ぎません。必ず正しいディレクトリー・パスを使用するようにしてください。

これらの変数は、次のように他の方法でも設定することができます。

- これらの変数を `rsed.envvars` に追加して Rational Developer for System z に公開する。これは `RSE` を構成した場合に適用されます。
- システム共通プロファイル (`/etc/profile`) で定義する。
- `export` ステートメントがファイルの先頭にくるように `ANT_HOME` ディレクトリーの `/bin/Ant` ファイルを変更する。例:

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

Ant の初期化が正常に行われたことをテストするには、次のようにします。

1. Ant および Java bin ディレクトリーを環境変数 `PATH` に追加します。この `PATH` 変数を `.profile` に追加するか、または UNIX システム・サービス・コマンド行から次の `PATH` 定義を入力することもできます。

例:

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

2. Ant を実行してバージョンを表示します。

例:

```
Ant -version
```

Ant が正常にインストールされていれば、Ant のバージョンが表示されます。

注: この方法による `PATH` ステートメントの設定はテスト時に必要なものであって、実際の運用に際しては不要です。通常の SCLM Developer Toolkit ビルド処理では、`ANT_HOME` および `JAVA_HOME` 環境変数は `$GLOBAL` メンバーで設定された値から動的に設定されます。`$GLOBAL` メンバー・パラメーターについて詳しくは、61 ページの『`$GLOBAL` メンバー』を参照してください。

ステップ 8a: HTTP のインストールとカスタマイズが正しく行われたことを確認するための IVP の実行

インストール検査プロセス (IVP) は、HTTP サーバーを構成した場合に適用されます。検査処理に成功するためには、HTTP サーバーが実行中で、IVP `pass/exec` デイレクティブが `httpd.conf` ファイルに構成されている必要があります。

ブラウザから、ロケーションの URL アドレスを次のように入力します。

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

HTTP のインストールとカスタマイズが正しく行われたことを確認するための IVP の実行

この場合:

hostname HTTP サーバーを実行している TCP/IP ホストの名前。
port number ジョブおよび httpd.conf ファイルで使用されるポート (デフォルト・ポート 80)。

HTTP サーバーが実行中である場合は、Web サーバーを始動するシステムの有効な TSO ユーザー ID およびパスワードを要求するプロンプトが表示されます (図 6)。

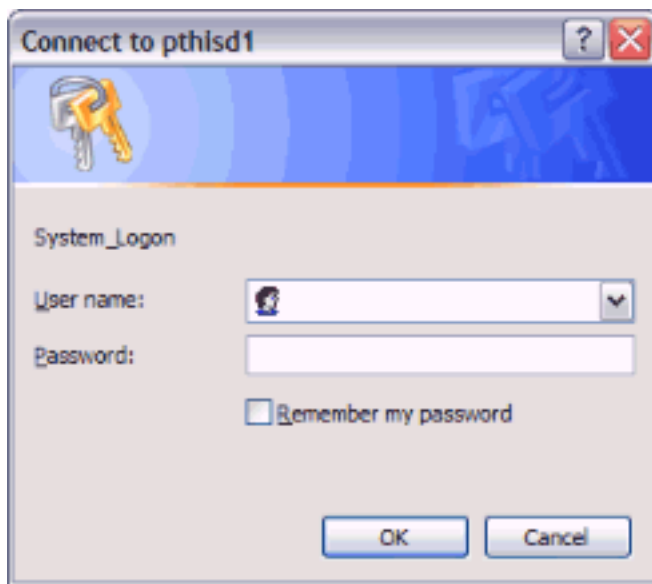


図 6. HTTP サーバーのログオン・プロンプト

TSO ユーザー ID とパスワードを入力すると、ブラウザーは最初に html ウェルカム画面 (25 ページの図 7) を表示します。

HTTP のインストールとカスタマイズが正しく行われたことを確認するための IVP の実行

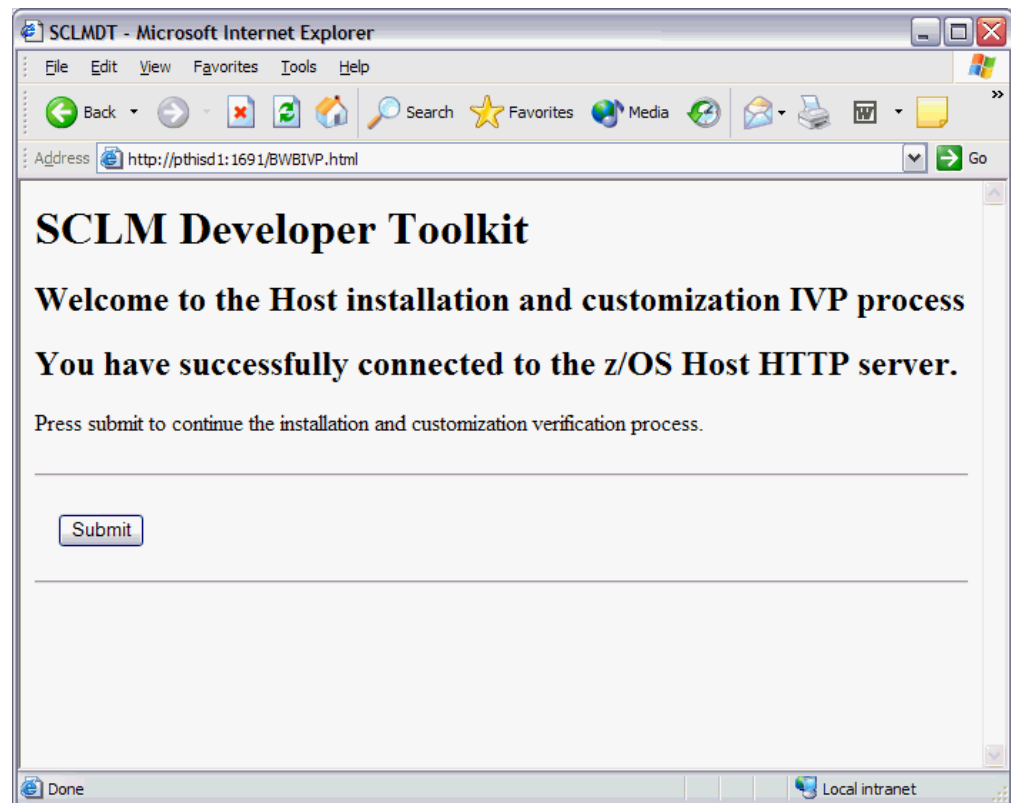


図 7. ホストのインストールとカスタマイズのウェルカム画面

接続に失敗した場合は、以下のことを確認してください。

- HTTP サーバーが正常に初期化された。
- SCLM Developer Toolkit のインストールを含む z/OS UNIX システム・サービス・ファイル・システムのマウント・ポイントがマウントされている。
- hostname:port が正しい (ホスト名の ping を試みる)。
- ファイアウォールの制約がない。
- httpd.conf ファイル内の PASS ディレクティブが次のように正しく設定されている。

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

ウェルカム画面が表示されたら、IVP を続行します。インストールおよびカスタマイズ・プロセスの確認と検証が実行されます。

サンプル画面 (26 ページの図 8 から 27 ページの図 10 まで) は、考えられる検証応答の例を示しています。

HTTP のインストールとカスタマイズが正しく行われたことを確認するための IVP の実行

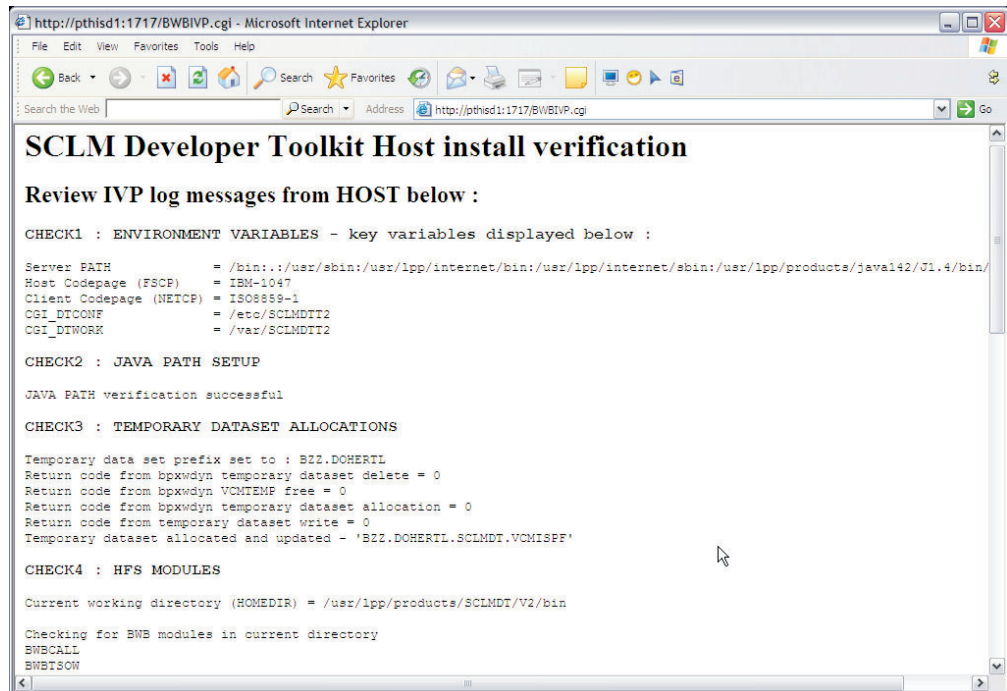


図8. 検証応答の例 (パート 1)

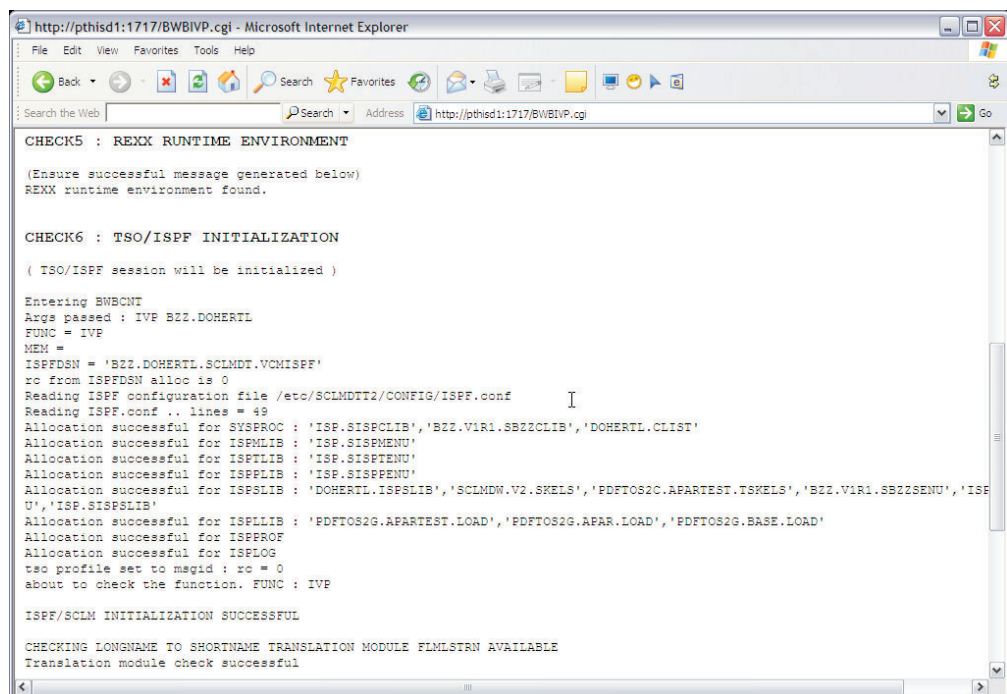


図9. 検証応答の例 (パート 2)

HTTP のインストールとカスタマイズが正しく行われたことを確認するための IVP の実行

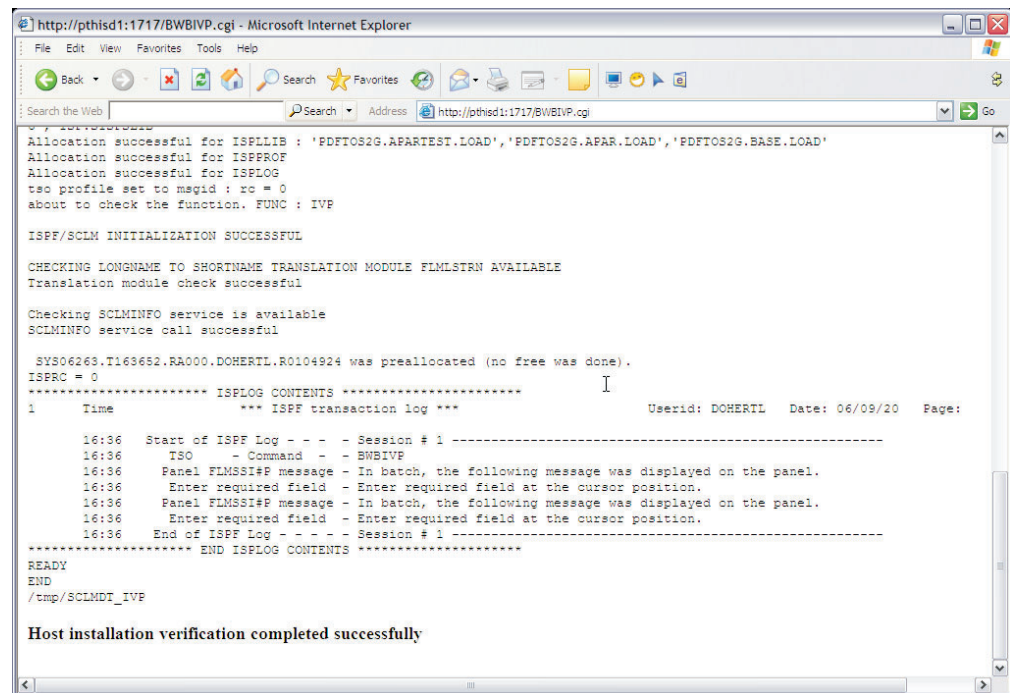


図 10. 検証応答の例 (パート 3)

HTTP サーバーへの接続のテスト

サーバー接続のテストは、IVP 完全検査を実行しなくても、いつでも行うことができます。

ブラウザから、ロケーションの URL アドレスを次のように入力します。

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

この場合:

hostname HTTP サーバーを実行している TCP/IP ホストの名前。

port number ジョブおよび `httpd.conf` ファイルで使用されるポート (デフォルト・ポート 80)。

Web サーバーを始動するシステムの有効なユーザー ID およびパスワードを要求するプロンプトが表示されます。

すると、ブラウザは 28 ページの図 11 に示すメッセージを表示します。

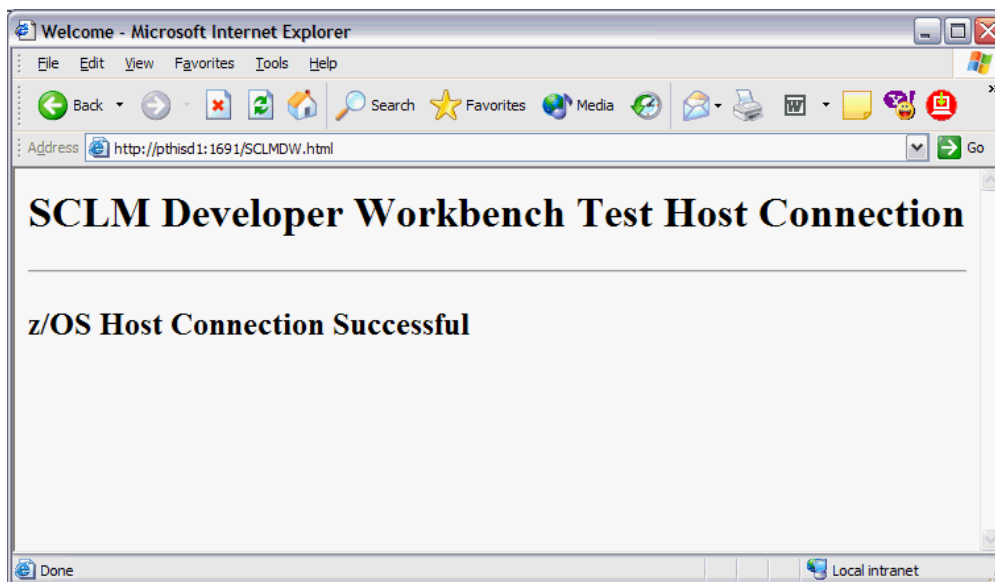


図 11. サーバー接続が成功した場合のメッセージ

ステップ 8b: RSE のインストールとカスタマイズが正しく行われたことを確認するための IVP の実行

インストール検査プロセス (IVP) は、RSE 接続を構成した場合に適用されます。この接続は、Developer Toolkit が Rational Developer for System z プラグインとしてインストールされたときに使用されます。z/OS RSE 接続が構成され、実行中でない限りなりません。検査処理に成功するためには、rsed.envvars ファイルに SCLM Developer Toolkit のディレクティブが構成されている必要があります。この詳細については、16 ページの『ステップ 5b: リモート・システム・エクスプローラーの構成』を参照してください。

注: Rational Developer for System z は、RSE コンポーネント用のいくつかの IVP テストも提供します。詳しくは、「*IBM Rational Developer for System z Host Configuration guide*」(SC31-6930-02) を参照してください。

以下の手順に従って SCLM Developer Toolkit IVP を呼び出します。

1. Rational Developer for System z で、Remote Systems Explorer パースペクティブが開いていることを確認します。SCLM に接続する z/OS 接続の場合は、「USS シェル (USS Shells)」ノードを右クリックし、「シェルの起動 (Launch Shell)」を選択します。
2. シェルのコマンド行で、ディレクトリを SCLM Developer Toolkit z/OS UNIX システム・サービス・ファイル・システム・モジュールのインストール・ディレクトリに変更します。デフォルトでは、これは /usr/lpp/SCLMDT/bin になります。これを行うには、29 ページの図 12 に示すように cd コマンドを使用します。

RSE のインストールとカスタマイズが正しく行われたことを確認するための IVP の実行

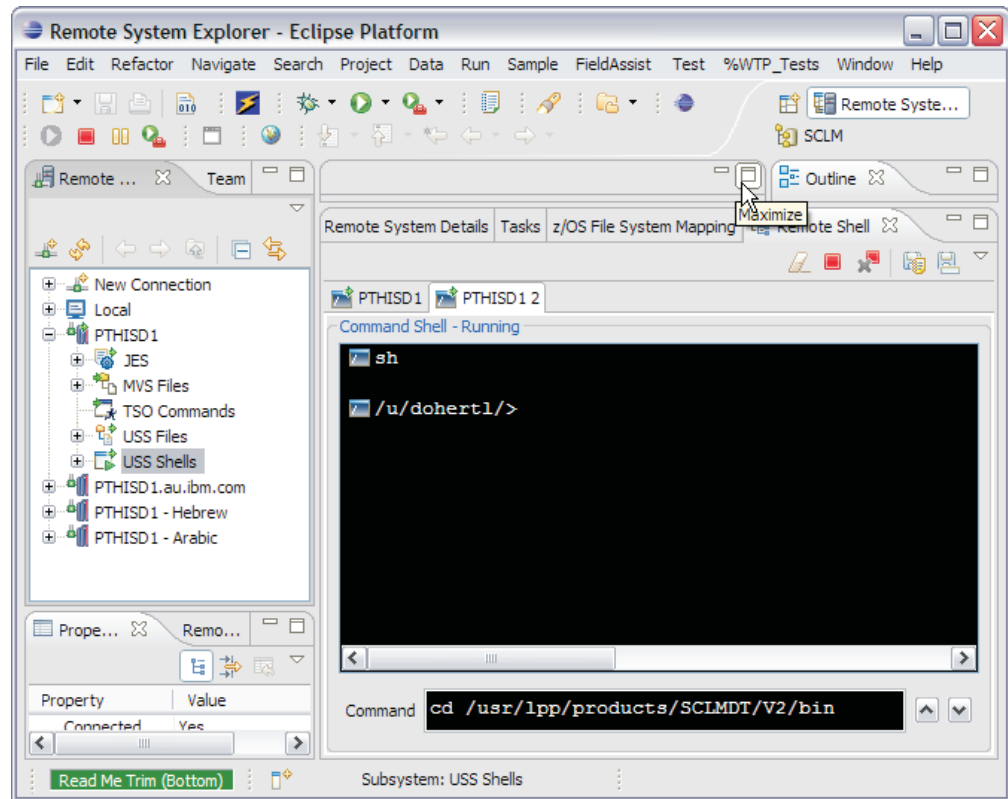


図 12. ディレクトリー変更コマンド

3. コマンド行に `BWBIVPR.cgi` と入力して IVP スクリプトを実行します。このスクリプトはシェルで実行され、IVP のさまざまなテストを経て多数の検証応答を戻します。スクリプトの実行が完了したら、スクロールして前の実行内容を表示してすべての応答を調べ、IVP が正常に機能したかどうかを確認することができます。
4. IVP 検証応答の例を 30 ページの図 13 に示します。

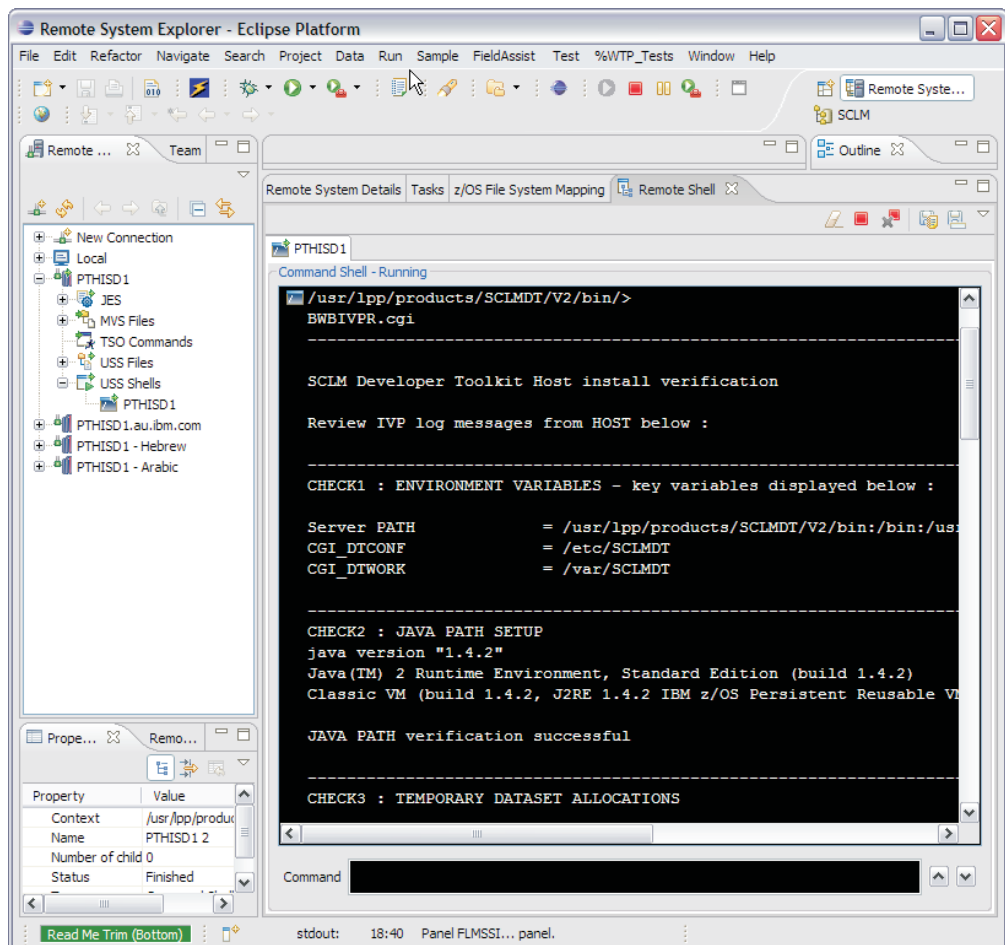


図 13. IVP 検証応答

第 2 章 Eclipse ベースのクライアントの PC へのインストール

既に Rational Developer for System z を PC にインストールしている場合は、SCLM Developer Toolkit プラグインがインストールされているため、クライアントをインストールする必要はありません。ただし、Rational Developer for System z のインストール時に、SCLM Developer Toolkit をインストールしなかった場合は、本章に記載されている手順に従って、既存の Eclipse に SCLM Developer Toolkit のプラグインのみをインストールする必要があります。

注: SCLM Developer Toolkit は、IBM Java Runtime Environment (JRE) バージョン 1.5 のみをサポートしています。

また、Rational Developer for System z 以外の異なる Eclipse に SCLM Developer Toolkit をインストールする場合は、以下のインストール手順を実行する必要があります。

インストールの準備

インストールの準備として、以下に示すメディア、ハードウェア、ソフトウェアの各種要件が満たされていることを確認してください。

メディア要件

ワークステーションに SCLM Developer Toolkit をインストールするには、SCLM Developer Toolkit のインストール CD またはその電子イメージへのアクセスが必要になります。

ハードウェア要件およびソフトウェア要件

以下に SCLM Developer Toolkit のハードウェア要件およびソフトウェア要件を示します。

SCLM Developer Toolkit の前提条件

IBM SCLM Developer Toolkit、ユーザーが z/OS 上の SCLM に配布されたコードを格納、またはビルドできるようにサポートするためのライセンス・プログラムで、IDE を使用して従来の z/OS の成果物とともに機能します。

ハードウェア要件: SCLM Developer Toolkit は、以下の表 6 に示したハードウェアが必要です。

表 6. 必須ハードウェア

プロセッサ	Intel Pentium III 800 MHz 以上
メモリー	768MB RAM が必要です。1GB RAM 以上が推奨されます。

表 6. 必須ハードウェア (続き)

ディスク・スペース	SCLM Developer Toolkit をインストールするには、最小 250 MB のディスク・スペースが必要です。本書では SCLM Developer Toolkit のプラグインに沿って、Eclipse を完全にインストールすると仮定しています。IBM Installation Manager を使用して SCLMDT をインストールするため、これをインストールするために最低で 120 MB のディスク・スペースが必要になります (まだこの製品がインストールされていない場合)。また、インストール時には製品の更新インストール・ファイルや他のファイルのために、さらに 700MB の一時スペースが必要になります。
モニターおよびビデオ・カード	1024x768 以上の VGA モニターが必要です。

サポートされているオペレーティング・システム: SCLM Developer Toolkit には、以下のオペレーティング・システムのいずれかが必要です。

表 7. 必須オペレーティング・システム

Windows XP	Professional (Service Pack 2)
Windows Server 2003	Standard または Enterprise Edition (Service Pack 1)
Windows Vista	Business Edition

SCLM Developer Toolkit のインストール

IBM Installation Manager を使用した SCLM Developer Toolkit のインストール IBM Installation Manager をお持ちでない場合は、SCLM Developer Toolkit CD に含まれています。

以下の手順では IBM Installation Manager のインストールに続いて、SCLM Developer Toolkit がインストールされます。

ステップ 1. CD または電子イメージからのインストール

CD または電子イメージから直接 SCLM Developer Toolkit をインストールすることができます。電子イメージを解凍すると、イメージは CD と同じ内容で構成されています。インストールは IBM Installation Manager により実行され、これは必要に応じてインストールされます。

- インストール CD を挿入し、install.exe (自動的に開始されない場合) を実行します。あるいは、解凍された電子イメージからインストールしている場合は、イメージが解凍されたディレクトリに移動し、install.exe を実行します。
 - ご使用のシステムに IBM Installation Manager がインストールされていない場合は、最初にこれをインストールするか確認するプロンプトが表示されます。これは Eclipse ワークベンチとすべての今後の更新をインストールするために使用されます。表示中の指示に従って、IBM Installation Manager をインストールしてください。
- 1. ご使用条件を受諾し、「次へ (Next)」をクリックします。

2. 必要に応じてデフォルトのインストール・ロケーションを変更し、「次へ (Next)」をクリックします。
 3. 「インストール (Install)」をクリックしてインストールを開始します。
 4. インストールが完了したら、「OK」をクリックして、SCLM Developer Toolkit のインストールに移ります。
 5. Installation Manager が自動的に開始します。このとき、Installation Manager は既に SCLM Developer Toolkit をインストールするように設定されています。ステップ 2 に移ります。
- IBM Installation Manager がインストール済みの場合は、自動的に開始します。このとき、Installation Manager は既に、SCLM Developer Toolkit をインストールするように設定されています。

IBM Installation Manager がインストールされ、ステップ 2 に移ります。

- また、IBM Installation Manager がインストールされている場合は、Installation Manager を開始し、CD または解凍した電子イメージをポイントすることで、SCLM Developer Toolkit のインストールを開始することもできます。
 1. IBM Installation Manager を開始します (「スタート」 > 「すべてのプログラム」 > 「IBM Installation Manager」 > 「IBM Installation Manager」)。
 2. Installation Manager ウィンドウで、「ファイル (File)」 > 「設定 (Preferences)」 > 「リポジトリ (Repositories)」の順に選択し、「リポジトリの追加 (Add Repository)」ボタンをクリックします。
 3. 「参照 (Browse)」ボタンをクリックし、CD または解凍済みイメージで「disk1」ディレクトリーを選択します。
 4. 「OK」をクリックしてリポジトリを保存します。
 5. 「OK」をクリックして、変更を保存し、「設定 (Preferences)」ウィンドウを閉じます。
 6. 「クイック・スタート」パネルで「パッケージのインストール (Install Packages)」ボタンをクリックします。ステップ 2 に移ります。

ステップ 2. SCLM Developer Toolkit のインストール

IBM Installation Manager の「パッケージのインストール (Install Packages)」スクリーンから実行するものを次に示します。

1. 「インストール (Install)」パネルで、使用可能なオファリングのリストで SCLM Developer Toolkit V3.1 がまだ選択されていない場合は選択し、「次へ (Next)」をクリックします。
2. 「ライセンス (License)」パネルではご使用条件を受諾し、「次へ (Next)」をクリックします。
3. 「ロケーション (Location)」パネルで、以下を行います。
 - a. 共有リソース・ディレクトリーの指定: IBM Installation Manager を使用して他のオファリングをインストールしていないと仮定し、共通コンポーネント・ディレクトリーとして使用するディレクトリーを指定します。

注: このディレクトリーは、IBM Installation Manager を使用してインストールされたすべてのオファリングによって使用され、またここに多くのファイル (例えば Eclipse フィーチャーおよびプラグインなど) がインストールされ

ます。そのため、ここに十分なディスク・スペースがあることを確認してください。

このロケーションが指定されると、後でインストールする他のオフリング用の「ロケーション (Location)」 パネルでこれを変更することはできません。

「次へ」をクリックします。

- b. **インストール・ディレクトリーの指定:** 次のスクリーンでオフリングをインストールするインストール・ディレクトリーを指定します。

注: このオフリングに固有の特定のファイルだけが、このロケーションにインストールされます。 オフリングのほとんどは (例えば、Eclipse プラグインとフィーチャーのすべて) この前のステップで指定した共通コンポーネント・ディレクトリーにインストールされます。

「次へ」をクリックします。

- c. **既存の Eclipse のインストールを拡張する:** 次のスクリーンでは、既存の Eclipse にインストールできる SCLM Developer Toolkit のプラグインを選択できます。 デフォルトでは、SCLM Developer Toolkit は独自の Eclipse インストールにインストールされます。 既存の Eclipse にプラグインを追加する場合は、Eclipse IDE のインストールのロケーションを指定します。

注: SCLM Developer Toolkit は、IBM Java Runtime Environment (JRE) バージョン 1.5 のみをサポートしています。 サポートされていない JRE を使用する既存の Eclipse 環境では、SCLM Developer Toolkit の実行時に問題が起る可能性があります。

「次へ」をクリックします。

4. 「フィーチャー (Features)」 パネルで、以下を行います。
 - a. **必須言語の選択:** インストールする言語パックを選択します。「次へ」をクリックします。
 - b. **追加機能の選択:** 次のスクリーンでは必要になる可能性があるフィーチャーを選択します。 SCLM Developer Toolkit はデフォルト・フィーチャーで、デフォルトの状態で選択されています。
5. 「要約 (Summary)」 パネルをレビューし、「インストール (Install)」をクリックして、インストールを開始します。
6. When the 「インストールの完了 (Installation completed)」 パネルが表示されたら、「終了 (Finish)」をクリックします。これで Installation Manager が終了します。

IBM SCLM Developer Toolkit がインストールされました。

Windows の「スタート」メニューからワークベンチを起動するには、「スタート」>「すべてのプログラム」>「IBM SCLM Developer Toolkit」>「IBM SCLM Developer Toolkit」>「IBM SCLM Developer Toolkit」の順に選択します。

第 2 部 SCLM Developer Toolkit のカスタマイズ

第 3 章 SCLM 管理者のための SCLM のカスタマイズ	37
JAVA/J2EE をサポートするための言語変換プログラム	37
JAVA/J2EE ビルドの要約	38
生成される JAVA/J2EE ビルド・オブジェクト	39
SCLM の言語定義	40
SCLM タイプ	42
SCLM メンバー形式	44
\$GLOBAL	44
J2EE ARCHDEF	44
J2EE Ant ビルド・スクリプト	48
JAVA/J2EE Ant XML のビルド・スケルトン	52
J2EE プロジェクトの SCLM へのマッピング	53
推奨される J2EE プロジェクトの SCLM へのマッピング	54
SCLM Developer Toolkit のデプロイメント	56
WebSphere Application Server (WAS) のデプロイメント	57
SCLM から Unix システム・サービスへのデプロイメント	58
セキュア・デプロイメント	58
公開鍵認証	59
他のデプロイメント・オプション	59
ASCII または EBCDIC のストレージ・オプション	59
ASCII/EBCDIC 言語変換プログラム	60
\$GLOBAL メンバー	61
SITE とプロジェクト固有のオプション	62
オプション定義	65
TRANSLATE.conf のオーバーライドの併用例	68
BIDIPROP のオーバーライドの併用例	69
第 4 章 SCLM セキュリティー	71
ビルド/プロモート/デプロイ・セキュリティー・フラグおよびプロセス・フロー	71
セキュリティー規則および代理ユーザー ID	72
ビルド規則形式	72
プロモート規則形式	72
デプロイ規則形式	73
SAF/RACF ビルド、プロモート、デプロイ、およびプロファイルに関する規則	73
第 5 章 CRON 開始のビルドおよびプロモート	77
STEPLIB および PATH に関する要件	78
CRON ビルド・ジョブの実行	79
CRON ビルド・ジョブのサンプル	79

第 3 章 SCLM 管理者のための SCLM のカスタマイズ

この章では SCLM 管理者がどのように SCLM をカスタマイズできるかについて説明します。以下のセクションで構成されています。

- 『JAVA/J2EE をサポートするための言語変換プログラム』
- 38 ページの『JAVA/J2EE ビルドの要約』
- 52 ページの『JAVA/J2EE Ant XML のビルド・スケルトン』
- 53 ページの『J2EE プロジェクトの SCLM へのマッピング』
- 56 ページの『SCLM Developer Toolkit のデプロイメント』
- 59 ページの『ASCII または EBCDIC のストレージ・オプション』

JAVA/J2EE をサポートするための言語変換プログラム

SCLM Developer Toolkit では JAVA/J2EE をサポートするために、SCLM で定義された 5 種類の新しい言語変換プログラムが必要になります。これらの言語変換プログラムは出荷時、以下のように SBWBSAMP のメンバーとして同梱されます。

サンプル

変換プログラム 説明

BWBTRANJ	サンプルのデフォルト・メンバー変換プログラム。構文解析なし。SCLM FLM@TEXT に類似。この変換プログラムはカスタマイズすることで、J2EEPART、J2EEBIN、BINARY、および TEXT の言語定義を作成できます。
BWBTRANS	サンプルの SQLJ 言語変換プログラム。LANG=SQLJ。
BWBTRAN1	サンプルの Java 言語変換プログラム。LANG=JAVA。
BWBTRAN2	Ant (複数の Java コンパイル、JAR、WAR、および EAR ビルド) を取り込んだサンプルの JAVA/J2EE 言語変換プログラム。
BWBTRAN3	サンプルの SCLM ARCHDEF J2EE サポート用 J2EE 言語変換プログラム。LANG=J3EEOBJ。

図 14. サンプルの変換プログラム

SCLM 管理者はこれらのサンプルをコピーし、必要に応じて名前変更を行います。続いて、Java サポートが必要な個所で、各 SCLM プロジェクトに対しこれらのサンプルを PROJDEFS.LOAD ライブラリーに生成します。これらの変換プログラムはプロジェクト定義で追加またはコンパイルを行うときに必要です。

JAVA/J2EE プロジェクトとホスト・コンポーネント用のサンプル・プロジェクト定義は、サンプル BWBSCLM に含まれています。

JAVA/J2EE 言語変換プログラム・モジュールにアクセスするには、BWB* モジュールを含む LOADLIB のデータ・セットを ISPF ISPLLIB の連結に含める必要があります。ISPLLIB 連結は構成ファイル ISPF.conf でカスタマイズされます。

JAVA/J2EE 用 SCLM データ・セット:

SCLM ターゲットのソース・データ・セットは、Toolkit クライアントが長形式のレコード・タイプに対応できるよう、SCLM に格納するすべての JAVA/J2EE ソース用に RECFM=VB、LRECL=10 で作成することを推奨します。Eclipse ベースのクライアントのエディターで作成されるファイルは可変長レコードです。整合性を保つために、ホストに存在する SCLM のターゲット・データ・セットも、RECFM=VB にするのが妥当です。固定長レコードのデータ・セット (RECFM=FB) を使用する場合、結果的に、インポートしたメンバーのレコードの終わりには空白文字が追加されます。

JAVA/J2EE ビルドの要約

提供された変換プログラムを使用した Java と J2EE の各ビルドに対して実行されるプロセスの要約を示します。

注: JAVA/J2EE メンバーか ARCHDEFS をホストの TSO/ISPF で直接ビルドできるだけでなく、Developer Toolkit クライアントを介してビルドすることもできます。

ARCHDEF には JAVA/J2EE プロジェクトをメークするメンバーが含まれ、これらのメンバーはショート・ネームで、プロジェクトが Eclipse のワークスペースに存在する状態を表します。

ARCHDEF 自体がビルドされ、ビルド前検査言語変換プログラム (J2EEANT) が呼び出されます。変換プログラムは ARCHDEF で SINC キーワードにより参照される J2EE のビルド・スクリプトを読み込み、プロパティ SCLM-ANTXML **A** により参照されるスケルトンの Ant XML に指定されたプロパティをオーバーレイします。ビルド・スクリプトが SCLM Developer Toolkit により生成されると、J2EEANT **1** の言語とともに SCLM に格納されます。

ARCHDEF では ARCHDEF **2** で INCLD キーワードにより識別される Java ソースの Java クラスが生成されるほか、各 ARCHDEF では JAR、WAR、EAR ファイルのような J2EE アーカイブ・ファイルが生成されることもあります。作成された J2EE オブジェクトは参照される適切なビルド・スクリプトと、ARCHDEF のキーワード OUT1 **3**、**B** に依存します。

ARCHDEF のビルド時、ビルド・スクリプト(SCLM タイプ J2EEBLD) に関連付けられた、ビルド前検査言語変換プログラムが実行され、再ビルドに必要な ARCHDEF のパーツを判別します (ARCHDEF の INCL キーワードの使用時に識別されるネストされた各 ARCHDEF の **4** などを含む)。続いて、これらのパーツは z/OS UNIX システム・サービスのファイル・システム作業域にコピーされ、Ant はビルド・スクリプトと ARCHDEF で指定された必須の JAVA/J2EE オブジェクトをコンパイルし、生成します。IDE プロジェクトで解決する必要のあるすべての外部 Jar およびクラスの参照は、CLASSPATH_JARS プロパティ **C** で定義されたパスから参照されます。

さらに、SCLM はコンポーネントに関連付けられた各言語変換プログラムを実行する個別の ARCHDEF コンポーネントを処理します。Java ソースに関連付けられた言語変換プログラムの JAVA は、作成されたクラス・ファイルを元の SCLM にコピーします。

最後に、ARCHDEF 変換プログラムは生成された J2EE オブジェクト (JAR、WAR、EAR) を判別し、これらのパーツを元の SCLM にコピーします。

エンタープライズ・アプリケーション (EAR) の構成に使用する可能性のある、各アプリケーション・コンポーネントに個別の ARCHDEF を作成することは不可欠です。つまり、EJB JAR を含む WAR を含んだ EAR には、JAR 用の ARCHDEF と、EJB JAR の ARCHDEF の INCL を備えた WAR 用の ARCHDEF が必要になります。そして、EAR ARCHDEF には WAR ARCHDEF の INCL が含まれている必要があります。

図 15 は対応する JAR サンプルを表示します。

```

*
* Initially generated on 10/05/2006 by SCLM DT V2
*
  LKED   J2EEOBJ           * J2EE Build translator
*
* Source to include in build
*
  INCLD  AN0000002 V2TEST   * com/Angelina.java           *
  INCLD  V20000002 V2TEST   * com/V2Java1.java      2      *
  INCLD  V20000003 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

```

図 15. サンプル Jar アプリケーション (JAR) ARCHDEF

図 16 は対応する JAR スクリプトを表示します。

```

<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>

```

図 16. J2EE ビルド・スクリプトの JAR サンプル

生成される JAVA/J2EE ビルド・オブジェクト

次のオブジェクトが生成されます。

- すべての Java ソースの出力クラスへのコンパイル結果。SCLM タイプ JAVACLAS に格納されます。

- SCLM に格納されたクラスと変換テーブルに格納されたロング/ショート・ネーム。
- オプションの作成済み Jar (クラスを含み、かつ XML や HTML など他の Java プロジェクトのコンポーネントをパッケージ構造で含む場合があります)。
- SCLM に格納された Jar オブジェクトと変換テーブルに格納されたロング/ショート・ネーム。
- 使用された ARCHDEF によって決まる Jar 構造。ARCHDEF のメンバーに関連付けられたロング・ネームにより、Jar のパッケージ化形式が決まります。
- オプションの EJB Jar (クラス、および、場合によって XML、HTML、JSP など他の Java プロジェクトのコンポーネントがパッケージされた構造で含まれます)。
- J2EE プロジェクトの J2EE web.xml ファイルを基にしたオプションの Web WAR ファイル。前述のとおり SCLM に格納されます。
- J2EE プロジェクトの J2EE application.xml を基にしたデプロイメント用のオプションの EAR ファイル。前述のとおり SCLM に格納されます。
- ここにリストされたすべての出力は SCLM タイプ J2EELIST に格納されます。

SCLM の言語定義

サンプルの変換プログラムは次の言語を定義します。

変換プログラム

説明

J2EEPART	JAVA/J2EE コンポーネントを指定し、サンプル BWBTRANJ によって定義された言語タイプ。この言語定義のビルド時、構文解析は特に行われません。ビルドの構文解析が特に必要ない場合 (html、XML、.classpath、.project、定義テーブルなどの場合)、通常、Java 以外のソース、または ASCII/EBCDIC の言語変換を必要とする J2EE コンポーネントは、この言語定義でスロットされます。オプションで、テキストの言語定義を使用することもできます。
J2EEBIN	バイナリー、または ASCII で格納される JAVA/J2EE コンポーネントを指定し、サンプル BWBTRANJ によって定義された言語タイプ。この言語定義のビルド時、構文解析は特に行われません。ビルドの構文解析が特に必要ない場合、通常、ASCII として格納しようとしている JAVA/J2EE のバイナリー・ファイルとテキスト・ファイルは、この言語定義でスロットされます。
JAVA	Java ソースの言語タイプで、サンプルの BWBTRAN1 で定義されています。Java 変換プログラムは Java ソースに対して実行されたビルドのタイプを判別します。

注: Java ソースを EBCDIC でホストに格納する場合 (つまり、ホストで ISPF を使って直接、ソースを表示または編集する可能性がある場合)、この言語定義を Java プログラムに割り当てる必要があります。この言語定義を使用してプログラムを定義する利点は、z/OS のホストから直接ソースを編集したり、表示したりできることです。一方、クライアントからホストにプロジェクトをマイグレー

ションする場合、あるいはインポートする場合、コード・ページ変換が必要になることが欠点と言えます。

- シナリオ 1: 個別の Java プログラムに対して実行されたビルド。

Java 変換プログラムはソースを出力クラスにコンパイルします。クラスは SCLM に JAVACLAS タイプで格納されます。Javac コンパイルによる出力は JAVALIST タイプで格納されます。

\$GLOBAL メンバー・パラメーターの CLASSPATH_JARS で指定されたクラスパス・ディレクトリーに、依存する JAR を格納することで、すべてのクラスパスの依存関係を解決できる場合があります。この詳細については、61 ページの『\$GLOBAL メンバー』を参照してください。

- シナリオ 2: ARCHDEF に対するビルド (ARCHDEF は SINC キーワードで参照される J2EEANT ビルド・スクリプトを呼び出します) は、ビルドを実行するために指定された Ant スクリプトを残します。Java 変換プログラム自体は、ARCHDEF によって呼び出されると、出力クラスを SCLM にコピーします。ANT ビルドの要約ファイルは、JAVALIST に格納されます。個別の Java コンポーネントには、JAVALIST に格納された出力テーブルがあります。

SQLJ

SQLJ ソース・コードの言語タイプで、サンプル BWBTRANS で定義されています。この言語変換プログラムで定義された SQLJ メンバーは、ビルド時に SQLJ 言語変換プログラムを呼び出します。SQLJ ソースは Java ソースに変換され、SQLJSER タイプのクラスと直列化オブジェクト (.ser ファイル) にコンパイルされます。オプションで、DBRM メンバーを DBRMLIB タイプに生成することができます。

JAVABIN

Java に似た言語タイプで、Java ソースを ASCII で SCLM に格納する場合に使用されます。

J2EEANT

これは JAVA/J2EE ビルドの主要なビルド変換プログラムで、この検査変換プログラムは J2EE ARCHDEF がビルドされる場合に呼び出されます。変換プログラムは SCLM に J2EEBLD タイプで格納されている JAVA/J2EE ビルド・スクリプトが、SCLM に J2EEANT の言語で保管された場合に呼び出されます。さらに、この変換プログラムは ARCHDEF で SINC キーワードを介して参照されます。

この検査変換プログラムは (ネストされた ARCHDEF を含め) どのパーツをビルドする必要があるかを判別し、ビルド・モードに応じて、これらのパーツを z/OS UNIX システム・サービスの WORKAREA ディレクトリーにコピーします。スケルトンの Ant XML はビルド・スクリプトと、Ant を使用する作業域でビルドされたパーツに従って動的にカスタマイズされます。クラス・ファイルは JAVA/JAVABIN 言語変換プログラムに渡され、クラス・ファイルは元の SCLM に格納されます。JAR、WAR、または EAR な

どの生成された J2EE オブジェクトは ARCHDEF 言語変換プログラム (J2EEOBJ) に渡され、元の SCLM に格納されます。

J2EEOBJ これは ARCHDEF のビルド・プロセスの一部として、最後に呼び出されるビルド変換プログラムです。この変換プログラムは前回 J2EEANT 変換プログラムでビルドされた J2EE オブジェクト (JAR、WAR、EAR) を判別し、これらのオブジェクトを指定した生成済みのショート・ネームで SCLM にコピーします。この変換プログラムは ARCHDEF 自体では LKED キーワードで参照されます。

注: JAR、WAR、および EAR などのすべてのオブジェクトには、任意のプラットフォームで配布できる ASCII 形式のソース・パーツを内部的に圧縮したものが含まれます。

SCLM タイプ

JAVA/J2EE をサポートするために作成しなければならない SCLM タイプが多数あります。これらのタイプの一部は必須で、JAVA/J2EE サポートが機能するには、これらのタイプを作成する必要があります。

一部の典型的タイプに関する推奨データ・セット属性

次の SCLM TYPES については、デフォルトのデータ・セット属性 DSORG=PO TRACKS(1,5) DIR=50 BLKSIZE=0 (システムにより決定されます) を推奨します。

さらに、レコード・フォーマットとレコード長については、次の属性を推奨します。

J2EEBLD recfm=FB lrecl=256

JAVALIST recfm=VB lrecl=255

J2EELIST recfm=VB lrecl=255

JAVACLAS recfm=VB lrecl=256

J2EEJAR recfm=VB lrecl=256

J2EEWAR recfm=VB lrecl=256

J2EEEAR recfm=VB lrecl=256

DBRMLIB recfm=VB lrecl=256

SQLJSER recfm=VB lrecl=256

Java/J2EE recfm=VB lrecl=1024 の追加ソース・データ・セットのタイプ。

タイプ 説明

J2EEBLD この SCLM タイプは Java および J2EE をビルドおよびデプロイするプロセスに必要です。

J2EEBLD タイプには以下が含まれます。

- Ant をビルドおよびデプロイするプロセスを実行するために使用される J2EEBLD ビルド・スクリプト。
- ビルドおよびデプロイの際に呼び出される Java スクリプトと J2EE ANTXML スクリプト。

注: サンプルの Java スクリプトと J2EE ANTXML スクリプトが提供されています。通常、これらのスクリプトではユーザーによるカスタマイズはほぼ不要です。サイト依存、およびユーザー依存の変数は J2EEBLD スクリプト内でカスタマイズされ、デフォルトの ANTXML 変数をオーバーライドします。(詳しくは、52 ページの『JAVA/J2EE Ant XML のビルド・スケルトン』を参照してください。)

これには Java ビルドと J2EE ビルドの両方で必要な \$GLOBAL メンバーが含まれます (61 ページの『\$GLOBAL メンバー』を参照してください)。

	DBRMLIB	技術的には DB2 タイプです。
		DBRMLIB は SQLJ Support には必須であり、FB=80 である必要があります。データベース要求モジュールを格納します。
	SQLJSER	このタイプは J2EE/SQLJ のビルド・プロセスに必要です。
		SQLJSER タイプは、SQLJ 直列化プロファイルを格納しています。
	ARCHDEF	<p>これには JAVA/J2EE ARCHDEF メンバーが含まれます。</p> <p>各 ARCHDEF メンバーのロング・ネームのパーツにより JAVA/J2EE のプロジェクト構造の全体像を知ることができます。指定されたプロジェクトの ARCHDEF は、新しいプロジェクトへのマイグレーション時にクライアントから動的に作成されるか、既存のプロジェクトに新しいパーツを追加するときに更新されます。</p> <p>SCLM ARCHDEF は JAVA/J2EE プロジェクトのエレメントを定義するための主要 SCLM ファイルです。JAVA/J2EE アプリケーションに関して、ARCHDEF は、クライアントの IDE プロジェクト・ワークスペースで、J2EE アプリケーションがどのような構造を持つかを示します。</p> <p>アプリケーションの Project ファイルの構造は、ARCHDEF に複製されます (ロング・ネーム構造をマップするために SCLM ホストのショート・ネームが使用されます)。LINK、SINC、および OUT1 などの ARCHDEF に含まれる追加キーワードは SCLM に対し、このプロジェクトの J2EE ネーチャーやソースを示し、このプロジェクトのビルド・プロセスを容易にする JAVA/J2EE ビルド・スクリプトを含みます。</p>
	JAVALIST	<p>この SCLM タイプは Java のビルド・プロセスに必要です。</p> <p>JAVALIST タイプには Java ビルドの出力リストが含まれます。</p>
	J2EELIST	<p>この SCLM タイプは J2EE のビルド・プロセスに必要です。</p> <p>J2EELIST タイプには J2EE ビルドの出力リストが含まれます。</p>
	JAVACLAS	<p>この SCLM タイプは Java と J2EE の両方のビルド・プロセスに必要です。</p> <p>JAVACLAS タイプには JAVA J2EEANT 言語定義に関連付けられたビルドの出力クラス・ファイルが含まれます。</p>

- J2EEJAR** この SCLM タイプは JAVA/J2EE ビルド (言語定義 J2EEANT) に必要です。
J2EEJAR タイプには J2EEANT 言語定義に関連付けられたビルドの JAR 出力が含まれます。
- J2EEWAR** この SCLM タイプは J2EE のビルド・プロセスに必要です。
J2EEWAR タイプには J2EEANT 言語定義に関連付けられたビルドの WAR 出力が含まれます。
- J2EEEAR** この SCLM タイプは J2EE のビルド・プロセスに必要です。
J2EEEAR タイプには J2EEANT 言語定義に関連付けられたビルドの EAR 出力が含まれます。

<Java/J2EE> タイプ

SCLM に格納するため個々の JAVA/J2EE プロジェクトには個別の SCLM タイプが必要です。これは JAVA/J2EE プロジェクトで同一ファイル名による競合が発生するのを回避するためです。この詳細については、53 ページの『J2EE プロジェクトの SCLM へのマッピング』を参照してください。

SCLM メンバー形式

このセクションでは SCLM メンバー形式について説明します。

- 『\$GLOBAL』
- 『J2EE ARCHDEF』
- 48 ページの『J2EE Ant ビルド・スクリプト』

\$GLOBAL

\$GLOBAL の形式はタイプ J2EEBLD で言語 J2EEPART です。必ず名前 \$GLOBAL を使用します。変数はタグ付きの言語形式で定義されます。

\$GLOBAL は JAVA/J2EE ビルド・プロセスの SCLM プロジェクトに対し、デフォルト・プロパティを指定します。これは SCLM タイプの J2EEBLD に格納される必要があります。

\$GLOBAL メンバー・パラメーターについて詳しくは、61 ページの『\$GLOBAL メンバー』を参照してください。

\$GLOBAL サンプル:

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

J2EE ARCHDEF

J2EE ARCHDEF の形式はタイプ ARCHDEF で言語 ARCHDEF です。

LKED	J2EEOBJ	
INCLD	SourceFile	SourceType
INCL	ArchdefName	ArchdefType

```

SINC  BuildScriptname  J2EEBLD
OUT1  *                  J2EEOutputObjectType
LIST  *                  J2EELIST

```

ARCHDEF は標準の SCLM アーキテクチャー・キーワードを使用して、ARCHDEF のビルドの処理方法を SCLM に指示します。

LKED これは LEC ARCHDEF であることを示し、呼び出す ARCHDEF の言語変換プログラムの言語を指定します (J2EE ARCHDEF では常に J2EEOBJ になります)。

INCLD

J2EE コンポーネントの SCLM インクルード。 *SourceFile* はソース・メンバーの名前 (例えば Java ソース) で、この ARCHDEF に含まれます。*SourceType* はそのメンバーを含む SCLM タイプです。SCLM Developer Toolkit で生成された ARCHDEF には、ワークベンチのプロジェクトに存在するのと同様に、ファイルのフルネームを表すコメントが含まれます。

INCL 別のネストされた ARCHDEF (EJB アプリケーションのマニフェストを含む ARCHDEF など) の SCLM インクルード。

SINC J2EEBLD ビルド・スクリプトのソース・インクルード。 *BuildScriptName* はビルド・スクリプトの名前です。ソース・タイプは常に J2EEBLD です。

OUT1 この ARCHDEF で作成された J2EE オブジェクトのタイプを示します。メンバー名は常に * です。 *J2EEOutputObjectType* は J2EEEAR、J2EEWAR、または J2EEJAR のいずれかに設定されます。作成されたメンバーの名前は、JAR、WAR、または EAR ファイル用に生成されたショート・ネームの名前になります。

LIST コンポーネントの要約のリストとビルドされた ARCHDEF の監査。メンバー名は常に * です。ソース・タイプは常に J2EELIST です。メンバーの名前は ARCHDEF のメンバー名と同じ値になります。

J2EE ARCHDEF サンプル: このセクションでは JAR、WAR、EJB、および EAR のサンプルを示します。

```

*
* 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

```

図 17. サンプル Jar アプリケーション (JAR) ARCHDEF

```

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

```

図 18. サンプル Web アプリケーション (WAR) ARCHDEF

```

LKED J2EEOBJ
*
INCLD XX000001 SAMPLE3 * .classpath *
INCLD XX000002 SAMPLE3 * .project *
INCLD MA000004 SAMPLE3 * ejbModule/META-INF/MANIFEST.MF *
INCLD EJ000004 SAMPLE3 * ejbModule/META-INF/ejb-jar.xml *
INCLD IB000003 SAMPLE3 * ejbModule/META-INF/ibm-ejb-jar-bnd.xmi *
INCLD XX000008 SAMPLE3 * ejbModule/com/ibm/ejs/container/_EJSWrapper *
* _Stub.java *
INCLD XX000009 SAMPLE3 * ejbModule/com/ibm/ejs/container/_EJSWrapper *
* _Tie.java *
INCLD XX000010 SAMPLE3 * ejbModule/com/ibm/websphere/csi/_CSIServant *
* _Stub.java *
INCLD XX000011 SAMPLE3 * ejbModule/com/ibm/websphere/csi/_Transactio *
* nalObject_Stub.java *
INCLD DA000005 SAMPLE3 * ejbModule/myEJB/DateBean.java *
INCLD DA000006 SAMPLE3 * ejbModule/myEJB/DateBeanBean.java *
INCLD DA000007 SAMPLE3 * ejbModule/myEJB/DateBeanHome.java *
INCLD EJ000001 SAMPLE3 * ejbModule/myEJB/EJSRemoteStatelessDateBeanH *
* ome_1a4c4c85.java *
INCLD EJ000002 SAMPLE3 * ejbModule/myEJB/EJSRemoteStatelessDateBean_ *
* _1a4c4c85.java *
INCLD EJ000003 SAMPLE3 * ejbModule/myEJB/EJSStatelessDateBeanHomeBea *
* nHomeBean_1a4c4c85.java *
INCLD XX000012 SAMPLE3 * ejbModule/myEJB/_DateBeanHome_Stub.java *
INCLD XX000013 SAMPLE3 * ejbModule/myEJB/_DateBean_Stub.java *
INCLD XX000014 SAMPLE3 * ejbModule/myEJB/_EJSRemoteStatelessDateBean *
* Home_1a4c4c85_Tie.java *
INCLD XX000015 SAMPLE3 * ejbModule/myEJB/_EJSRemoteStatelessDateBean *
* _1a4c4c85_Tie.java *
INCLD XX000016 SAMPLE3 * ejbModule/org/omg/stub/javax/ejb/_EJBHome_S *
* ub.java *
INCLD XX000017 SAMPLE3 * ejbModule/org/omg/stub/javax/ejb/_EJBObject *
* _Stub.java *
INCLD XX000018 SAMPLE3 * ejbModule/org/omg/stub/javax/ejb/_Handle_St *
* ub.java *
INCLD XX000019 SAMPLE3 * ejbModule/org/omg/stub/javax/ejb/_HomeHandl *
* e_Stub.java *
INCLD DA000008 SAMPLE3 * ejbModule/services/DateBeanServices.java *
INCLD XX000020 SAMPLE3 * ejbModule/services/_DateBeanServices_Stub.j *
* ava *
*
SINC SAMPLE3 J2EEBLD * J2EE EJB JAR Build script *
OUT1 * J2EEJAR * DateService.jar *
*
LIST * J2EELIST

```

図 19. サンプル EJB アプリケーション (EJB) ARCHDEF

```

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

```

図 20. サンプル EAR アプリケーション (EAR) ARCHDEF

J2EE Ant ビルド・スクリプト

J2EE Ant のビルド・スクリプト形式はタイプ J2EEBLD で言語 J2EEANT です。最大で 8 文字までの任意の名前を使用でき、変数はタグ付きの言語形式で定義されます。ビルド・スクリプトは JAR、WAR、および EAR によく似ています。以下の構文は WAR のビルド・スクリプトを示しています。JAR および EAR の場合、ビルド・スクリプトの変数は WAR_NAME の代わりに EAR_NAME と JAR_NAME がそれぞれ使用される以外、同じです。

```

<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>

```

図 21. J2EE Ant ビルド・スクリプト

SCLM ビルド・スクリプトは Ant ビルド・スクリプトを実行するとき、ビルドの要求に応じて動的にユーザー定義の変数をオーバーレイします。これらの変数は、49 ページの表 8 に示した値に設定されています。

表 8. ユーザー定義変数

変数	説明
J2EE Project name	<p>ビルドされている Java/J2EE プロジェクトのタイプ。これはビルド中に使用するため Ant のビルド・スクリプトに設定された一時的なプロジェクト名です。このタイプが設定される値は以下のとおりです。</p> <ul style="list-style-type: none"> • J2EE EAR Project • J2EE WAR Project • J2EE EJB Project • JAVA Project <p>変数をカスタマイズする必要はありません。</p>
SCLM_ARCHDEF	ARCHDEF 名、またはビルドされる ARCHDEF
SCLM_ANTXML	ビルドに使用されるスケルトン Ant XML の名前
SCLM_BLDMAP	値 Yes または No。 Yes の場合は、SCLM ビルド・マップを JAR、WAR、または EAR 内の MANIFEST ディレクトリーに組み込みます。監査と組み込まれたパーツのビルド・マップを提供します。
JAVA_SOURCE	値 Yes または No。 Yes の場合は、Java ソースを JAR、WAR、または EAR に組み込みます。
CLASSPATH_JARS	z/OS UNIX システム・サービスのクラスパス・ディレクトリー。ビルド中にクラスパスの依存関係を解決するために使用されます。このディレクトリーに配置されたすべての Jar はクラスパスで使用されます。
CLASSPATH_JARS_FILES	<p>ビルドに組み込まれる個々の JAR ファイルとクラス・ファイルの名前。これは以下のようなリスト形式を使用できます。</p> <pre><property name="CLASSPATH_JARS_FILES" value="V2J4.jar,V2J3.jar" /></pre>
エンコード	<p>JAVA コード・ページを ASCII または EBCDIC で指定します。JAVA ソースが z/OS ホストに格納される場合のコード・ページです。例:</p> <ul style="list-style-type: none"> • ASCII の場合、JAVA 標準コード・ページは ISO8859-1 に設定する必要があります。 • EBCDIC の場合、JAVA 標準コード・ページは IBM-1047 に設定する必要があります。
JAR_FILE_NAME EJB_NAME WAR_NAME EAR_NAME	JAR、EJB JAR、WAR、または EAR の名前
DEBUG_MODE	Developer Toolkit で WORKAREA ディレクトリーにあるビルド・ファイルを削除しないよう強制的に指定する場合は、「on」に設定します。これはビルドした Java/J2EE アプリケーションの構造を確認する必要がある場合に役立ちます。

クラスパスの依存関係: ARCHDEF 内に含まれる Java ソースには他の Java ライブラリーやクラスとの間で、クラスパスの依存関係が生じる場合があります。これらの依存関係が同一の ARCHDEF 構造内に含まれる Java コンポーネントに存在す

る場合は、ARCHDEF ビルドの一部として、これらのクラスパスの依存関係は解決されます (ビルド・モードが条件付きまたは強制的のいずれの場合も)。

ただし、J2EE ARCHDEF コンポーネントの場合は、外部の JAR または他の ARCHDEF に含まれるメンバーとの間で、クラスパス依存関係を持つことが可能です。この場合、ARCHDEF に関連付けられた J2EE ビルド・スクリプトは、次のキーワードを使用して依存関係を制御することができます。

CLASSPATH_JARS

これは z/OS UNIX システム・サービスのファイル・システムにあるディレクトリー名で、すべての外部依存 JAR ファイルと特定の ARCHDEF ビルドのクラスを含めることができます。

このディレクトリーは Client Team 機能「JAR ファイルのアップロード (Upload jar files)」を介して CLASSPATH ファイルで更新することができます。この機能は JAR ファイルをクライアントからクラスパス・ディレクトリーにコピーします。また、機能「ファイルを SCLM からクラスパスにファイルをコピー (Copy file from SCLM to classpath)」を使用して、JAR ファイルに格納された SCLM をクラスパス・ディレクトリーにコピーすることができます。

CLASSPATH_JARS_FILES

このキーワードはクラスパスに使用される個々の JAR やクラス・ファイルを選択するために使用できます。このキーワードを使用すると、リストされた JAR/クラス・ファイルが SCLM から検索され、SCLM 内にはない場合は、CLASSPATH_JARS で参照されているディレクトリーが検索されます。このキーワードを使用すると、リストされているファイルのみがクラスパスに使用されます。

J2EE サンプル・スクリプト: 次のサンプルでは JAR、WAR、EJB、および EAR の各スクリプトを示します。

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

図 22. J2EE ビルド・スクリプトの JAR サンプル

```
<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/SCLMDT/CLASSPATH"/>
  <property name="ENCODING" value="IBM-1047"/>
  <!-- WAR file name to be created by this build process -->
  <property name="WAR_NAME" value="Sample5.war" />
  <path id="build.class.path">
    <pathelement location="."/>
    <pathelement location="${J2EE_HOME}/lib/j2ee.jar"/>
    <pathelement location="${CLASSPATH_JARS}/jdom.jar"/>
  <fileset dir="${CLASSPATH_JARS}" includes="**/*.jar, **/*.zip"/>
</path>
</ANTXML>
```

図 23. J2EE ビルド・スクリプトの WAR サンプル

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

図 24. J2EE ビルド・スクリプトの EJB サンプル

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

図 25. J2EE ビルド・スクリプトの EAR サンプル

JAVA/J2EE Ant XML のビルド・スケルトン

このセクションでは SBWBSAMP ライブラリーに含まれているサンプルの Ant ビルド・スケルトンを示します。これらのサンプル・メンバーは SCLM 階層の SCLM タイプ J2EEBLD にコピーすることで、JAVA/J2EE のビルド・スクリプトから参照および使用できるようになります。JAVA/J2EE のビルド・スクリプトは Ant XML スケルトン・ファイルをオーバーレイするプロパティー変数ファイルです。

単純な JAR、SQLJ プロジェクト、EJB JAR、WAR、EAR のビルドまたはデプロイメントのために提供されるサンプル J2EE ビルド・スケルトンは、一般にカスタマイズせずにそのまま使用できます。ただし、J2EE プロジェクトの中には標準モデルに合わないものがあり、提供された Ant XML スケルトンへのカスタマイズが必要になる可能性がある点に注意してください。

注: JAVA/J2EE のビルド・スクリプトは、Developer Toolkit のクライアント・アプリケーションを介して生成することができます。これらのビルド・スクリプトは参照された Ant XML スケルトン (下記) JAVA/J2EE ビルド・プロセスの ARCHDEF を使用します。

ビルド・スクリプトの詳細説明、JAVA/J2EE ビルド処理の Ant スケルトンと例は、クライアントのプラグインとともに提供された「SCLM Developer Toolkit User Guide」に記載されています。

BWBJAVAA サンプル Ant XML JAVA ビルド・スケルトン

この Ant スケルトンは、Java ビルド・スクリプトで使用され、複数の Java プログラムをコンパイルし、オプションで、指定された ARCHDEF で判別される構造を持つ Java アーカイブ (JAR) を作成します。

BWBEJBA	<p>サンプル Ant XML J2EE EJB ビルド・スケルトン</p> <p>この Ant スケルトンは、J2EE ビルド・スクリプトで使用され、EJB プロジェクトをコンパイルまたはビルドします。EJB プロジェクトは通常、EJB JAR を作成します。EJB JAR は ARCHDEF で定義されている構造を持ちます。</p>
BWBWEBA	<p>サンプル Ant XML J2EE WEB ビルド・スケルトン</p> <p>この Ant スケルトンは、J2EE ビルド・スクリプトで使用され、WEB プロジェクトをコンパイルまたはビルドします。WEB プロジェクトは通常、WEB アーカイブ (WAR) ファイルを作成します。</p>
BWBEARA	<p>サンプル Ant XML J2EE EAR アセンブル・スケルトン</p> <p>この Ant スケルトンは J2EE アプリケーション・デプロイメントの準備で、アセンブル・プロセスの 1 つとして J2EE ビルド・スクリプトで使用されます。このプロセスでは、WebSphere Application Server などの Web アプリケーション・サーバーにデプロイできる、エンタープライズ・アーカイブ (EAR) ファイルが作成されます。</p>
BWBSQLB	<p>サンプルの Java/SQLJ ビルド・スクリプト。</p> <p>この Ant スケルトンは J2EE ビルド・スクリプトで使用され、SQLJ を使用する JAR プロジェクトをコンパイルし、ビルドします。</p>
BWBSQLBE	<p>サンプルの EJB/SQLJ ビルド・スクリプト。</p> <p>この Ant スケルトンは、J2EE ビルド・スクリプトで使用され、SQLJ を使用する EJB プロジェクトをコンパイルし、ビルドします。</p>

J2EE プロジェクトの SCLM へのマッピング

IBM SCLM Developer Toolkit には SCLM のプロジェクトを管理、ビルド、およびデプロイする能力が備わっています。このセクションでは JAVA/J2EE などの分散アプリケーション開発をサポートするために、SCLM プロジェクト構造を構成する方法について説明します。

通常の JAVA/J2EE プロジェクトでは、結果的に実行可能な EAR ファイルが作成されます。このアプリケーションは IDE 環境内のプロジェクト、通常は EJB や Web アプリケーションのアセンブリーであり、これらは一般的に EAR プロジェクトにリンクされた個々のプロジェクト構造として開発されます。

複数プロジェクト構造を持つこの形式は、SCLM に直接マップされることはありません。つまり、集約されたプロジェクト構造のフォームを提供するために、SCLM プロジェクトを他の SCLM プロジェクトにリンクすることはできません。ただし、SCLM には、タイプを使用して、単一 SCLM プロジェクト内でこうした複数プロジェクト構造をサポートする手段が提供されています。

SCLM プロジェクトは複数のソース・タイプで定義できます。各タイプでは 1 つの IDE プロジェクトを保持できます。フォームを分離せずに SCLM に含まれる複数の Eclipse IDE プロジェクトを格納しようとすると、プロジェクトの各 .classpath

J2EE プロジェクトの SCLM へのマッピング

および .project ファイルはプロジェクトが SCLM に追加される順に上書きされます。異なるソース・タイプを使用することで、これらのファイルやプロジェクトに関連付けられた、すべての他のオブジェクトを安全に SCLM に格納できるようになります。

図 26 では、1 つの SCLM プロジェクトで複数プロジェクト構造をサポートするための、複数タイプの使い方について説明します。

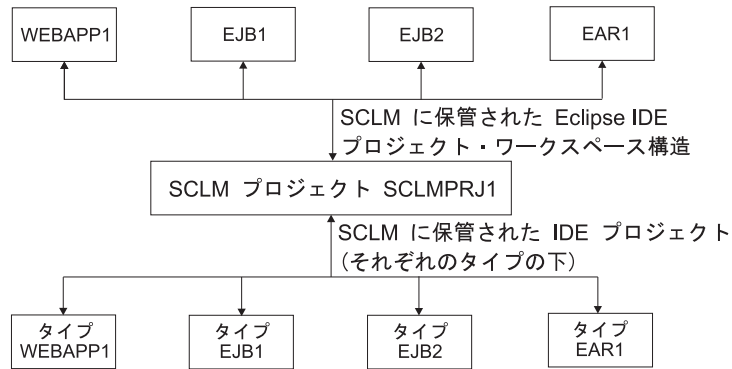


図 26. 複数タイプ

このようにマッピングすることで、タイプを基本的な微分器として使用し、SCLM 内に IDE プロジェクトを独立して格納できるようになります。例えば、EJB1 はタイプ EJB1 として SCLM プロジェクトの SCLMPRJ1 に格納されます。この構造を使用することで、SCLM プロジェクト内で IDE のプロジェクト構造を独立したタイプにマップできるようになります。

注:

1. IDE のプロジェクト名は必ずしも SCLM のタイプ名にマップする必要はありません。これらの名前は互いに独立して存在します。
2. タイプ名は 8 文字以下に制限されています。そのため、「ProjectOne」という名前の IDE プロジェクトで、対応するタイプ名として「ProjectOne」を使用することはできません。代わりに「Proj1」を使用できます。

このような理由から、さまざまな IDE ベース・プロジェクトを単一の SCLM プロジェクト構造にマッピングできるように SCLM プロジェクトの構造を計画することは重要です。これは、大規模な SCLM プロジェクトでは、プロジェクト・タイプの追加が、SCLM のプロジェクト定義の変更や、SCLM プロジェクト定義の再ビルド、新規タイプへのデータ・セットの割り当てなど、想定しなかった大規模な変更につながる可能性があるためです。

この構造は、J2EE スタイルのプロジェクトに限定されるものではありませんが、開発している複数のプロジェクトで互いに依存関係が発生する場合には、このような状態が生じる可能性があります。

推奨される J2EE プロジェクトの SCLM へのマッピング

次のリストは J2EE プロジェクト (およびその他のプロジェクト) を SCLM にマッピングする際に推奨される方法を示します。

推奨される J2EE プロジェクト (およびその他のプロジェクト) の SCLM へのマッピング

- EJB、Web アプリケーションなどに関して、J2EE プロジェクトの構成を特定します。これを、SCLM のプロジェクト構造の計画時に使用できます。
- 各 J2EE IDE プロジェクト・コンポーネントについて、対応するタイプを SCLM プロジェクトに作成します。これをサポートするため、なんらかの意味のある命名規則を指定すると便利です。IDE プロジェクトに SCLM タイプとは関係のない名前を付けることも可能ですが、一定の相関関係を保つ方が、管理が容易になります。
- プロジェクトの要件が変わることもあるため、追加 EJB など、他のコンポーネントの追加を円滑に実行できるよう、追加のタイプ定義を作成します。追加のサービスはタイプ構造から予測できます。
- 複数の IDE プロジェクトを単一の SCLM プロジェクトにマッピングするのは、タイプの構成でサポートされます。そのようなプロジェクト向けに、タイプ定義を想定したパッケージ化構造を適用するのも有用です。
- Java スタイルのパッケージ化規則は、プロジェクト・レベルで定義することもできます。こうすることで、ソースの命名で衝突が起こる可能性を回避できます。
- 複数のプロジェクトにも対応するよう、IDE を構造化した場合は、SCLM でもこの構造をタイプを使用して複製することをお勧めします。

複数の SCLM タイプを使用して個別の IDE プロジェクトを格納することは、これらの IDE プロジェクトのビルドに対する ARCHDEF 構造の操作にも関連します。

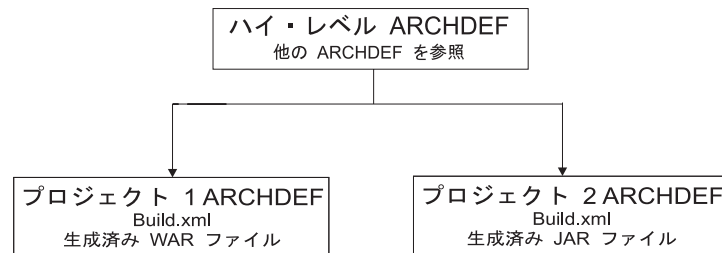


図 27. SCLM ビルド階層

ARCHDEF ファイルにはビルドを構成するファイルのリストが含まれます。J2EE コンテキストでは、ビルドの結果、1 つの EAR ファイルに多数の WAR ファイルと JAR ファイルの組み合わせが含まれる場合があります。プロジェクトのこうした分離は、SCLM でプロジェクトを定義するタイプ構造に似ています。ビルドを構成するこれらの「パーツ」を参照するハイ・レベルの ARCHDEF を使用することにより、ビルド環境を構造化することができます。これは、SCLM でタイプを定義する際に有効なプロジェクト構造を定義することに関連します。

構造化された状態でプロジェクトを定義することで、以下のことが可能になります。

- SCLM プロジェクト・タイプまたは ARCHDEF から IDE プロジェクトにファイルをマイグレーションするとき、個別のパーツを識別する必要がなくなる。
- タイプ定義を基にした ARCHDEF 構造により、プロジェクトの依存関係をより効果的にマップできるようになる。これは、ワークスペースに含まれる他の IDE プロジェクトを参照する IDE プロジェクトで共通です。この概念を他の IDE プロジェクトと同様にサポートし、他の ARCHDEFs から参照される SCLM INCL

推奨される J2EE プロジェクト (およびその他のプロジェクト) の SCLM へのマッピング

キーワードを ARCHDEF で使用することにより、より高いレベルの ARCHDEFS で ARCHDEF をネストできるようになります。

JAR のような他のビルド・オブジェクト、他のプロジェクト、または他のクラスの参照または依存関係を含むアプリケーションをビルドする場合、以下の複数の方法が可能です。

1. ARCHDEF の INCLD ステートメントを使用して JAR への参照を組み込みます。これにより、アプリケーションのビルドで、ライブラリー参照を含む最終的なビルド・パッケージを作成できます。
2. IDE プロジェクトをネストされた INCL SCLM プロジェクト ARCHDEF として組み込みます。
3. 従属 JAR を CLASSPATH ディレクトリーに組み込みます。
 - a. IDE プロジェクトで JAR を参照する必要があるが、最終的なビルド・パッケージには含まれない場合、Developer Toolkit の「JAR のアップロード」サービスを使用して、システムの CLASSPATH にライブラリー・ファイルをコピーすることができます。これにより、異なる SCLM プロジェクトから使用可能なサービスを効果的にメークできるようになります。IDE プロジェクトの JAR への参照はビルド時に解決されます。ただし、実行時、JAR を PATH ステートメントに指定する必要があります。
 - b. ビルド・スクリプト内で CLASSPATH_JARS_FILES プロパティーを使用して、同一の SCLM プロジェクト内にある JAR を参照します。

SCLM Developer Toolkit のデプロイメント

SCLM Developer Toolkit には複数のデプロイメント・フィーチャーがあります。エンタープライズ・アーカイブ・ファイル (EAR) を WebSphere Application Server (WAS) にデプロイできます。さらに、SCLM Developer Toolkit によってビルドまたは制御されるすべてのコンポーネントは、カスタマイズ可能なデプロイ・スクリプトを使用して、配布することができます。セキュア・コピー (SCP) およびセキュア FTP (SFTP) コマンドを使用して EAR をリモート・ホストにコピーするために使用できるサンプルのスクリプトが用意されています。

デプロイメントのコアには、基本的に 2 つのスクリプトがあります。1 つ目のタイプのスクリプトはプロパティー・スクリプトで、ユーザーが変更できるタイプです。このスクリプトにはデプロイメント操作に使用するパラメーターのリストが含まれています。2 つ目はアクション・スクリプトで、デプロイメント操作に必要な各種ステップが含まれます。

デプロイメントは SCLM Developer Toolkit のクライアント・プラグインから開始され、デプロイメントのタイプはデプロイメント画面で適切なボタンを押して選択します。どのデプロイメント・アクションが選択されたかによって、プロパティー・スクリプトに取り込まれる内容が変わります。多くのスクリプトには、SCLM_ANTXML という名前のプロパティーがあり、対応するアクション・スクリプトのメンバー名が含まれます。Developer Toolkit は、生成されたプロパティー・スクリプトを取得して、アクション・スクリプトにそれをオーバーレイしてから、結果のアクション・スクリプトを呼び出します。

以下はサンプルの Ant デプロイメント・アクション・スクリプトです。このスクリプトは SBWBSAMP ライブラリーで提供されています。これらのサンプル・メンバーは SCLM 階層の SCLM タイプ J2EEBLD にコピーすることで、生成されたプロパティー・スクリプトから参照および使用できるようになります。生成されたプロパティー・スクリプトは、以下で参照される Ant XML デプロイメント・アクション・スクリプトをオーバーレイするプロパティー変数ファイルです。これらのスクリプトは TEXT または J2EEPART などのテキスト・タイプの言語で格納する必要があります。

メンバー名

説明

BWBDEPLA

WAS EAR デプロイメント。

BWBRDEPL

リモートの WAS EAR デプロイメント。

BWBSCOPY

セキュア・コピー・デプロイメント。SCP を使用して 1 つのホストから他のホストにビルド・オブジェクトをコピーします。

BWBSFTP

セキュア FTP デプロイメント。SFTP を使用して 1 つのホストから他のホストにビルド・オブジェクトをコピーします。

これらのビルド・スクリプトを複数のグループから使用できるようにするには、管理者は、スクリプトをビルドして、プロジェクトで有効な最上位のグループ・レベルにプロモートする必要があります。

生成されるスクリプトのタイプに応じて、実行される処理はわずかに異なります。

WebSphere Application Server (WAS) のデプロイメント

WebSphere Application Server (WAS) のデプロイメントでは、SCLM_ANTXML プロパティーは Ant アクション・スクリプトをポイントしません。代わりに、JACL アクション・スクリプトを参照します。また、z/OS の WAS に同梱されている wsadmin ツールを使用することもできます。wsadmin ツールでは、デプロイメント・プロセスをガイドするために JACL スクリプトが必要です。このデプロイ・メソッドを使用する場合は、デプロイメント・プロセスを呼び出す前に、JACL スクリプトを UNIX システム・サービスにインストールする必要があります。JACL スクリプトは z/OS の z/OS UNIX システム・サービスのファイル・システム・ディレクトリーにコピーしてください。JACL スクリプトを /etc/SCLMDT/CONFIG/scripts/deploy.jacl (ここで /etc/SCLMDT は SCLM Developer Toolkit のデフォルト etc ディレクトリー) として格納することをお勧めします。wsadmin ツールでは JACL スクリプトが ASCII フォーマットであると想定されます。

サンプルの JACL スクリプト BWBJACL (SBWBSAMP サンプル・ライブラリーにあります) を、z/OS UNIX システム・サービスのファイル・システム・ディレクトリーに ASCII フォーマットでコピーするには、次のステップを実行します。

- サンプル・ジョブ BWBJACLJ に含まれる説明をレビューし、適切にカスタマイズします。BWBJACLJ は SCLM Developer Toolkit の SBWBSAMP ライブラリーにあります。他の JACL の例は、WebSphere Application Server (WAS) のドキュメンテーションで参照できます。
- ジョブ BWBJACLJ を実行依頼します。これにより、サンプルの BWBJACL が ASCII に変換され、deploy.jacl というファイル名でカスタマイズされたディレクトリーにコピーされます。

クライアントのプラグインは、wsadmin ツール (wsadmin.sh) と JACL スクリプトがインストールされている、UNIX システム・サービスのディレクトリー・ロケーションと通信する必要があります。いずれのロケーションも「チーム (Team)」->「SCLM 設定 (SCLM Preferences)」->「ビルド・スクリプト・オプション (Build Script Options)」で構成できます。クライアント・プラグインはビルド可能なデプロイメント・スクリプトを生成するために使用されます。(デプロイメント・プロセスは、SCLM タイプ J2EEBLD に格納されているデプロイメント・スクリプトへのデプロイ機能要求によりトリガーされます)。

WAS デプロイメントまたはリモートの WAS デプロイメントでは、SCLM タイプ J2EEBLD に格納する必要のあるサンプルのアクション・スクリプトは、BWBDDEPLA および BWBRDEPL です。

SCLM から Unix システム・サービスへのデプロイメント

SCLM Developer Toolkit は、SCLM リポジトリに格納されているすべてのファイルを同一 LPAR 上にある z/OS UNIX システム・サービスのファイル・システムにデプロイするための手段を提供します。これにより、以下のセキュア・デプロイメントを使用して、SCLM によってビルドされたオブジェクトを、実行できる環境またはリモート・ホストにデプロイできる環境に簡単にデプロイする方法が提供されます。

このアクションのサンプル・アクション・スクリプトはありません。SCLM からメンバーを選択し、「SCLM メンバーを組み込む」ボタンを使用して、必要なプロパティー・スクリプトを生成します。これにより、選択した SCLM のロケーションから z/OS UNIX システム・サービスのファイル・システムで指定したディレクトリーに、ファイルがコピーされます。このディレクトリーが前もって存在していない場合、エラーが発生します。

セキュア・デプロイメント

このオプションはセキュア・コピー (SCP) とセキュア FTP (SFTP) コマンドを使用して、リモート・ホストにデプロイ可能なオブジェクトをコピーする手段を提供します。セキュア・デプロイのプロパティー・スクリプトと、「SCLM メンバーを組み込む」を併用すると、必要なファイルを SCLM 階層から選択して、z/OS UNIX システム・サービスのファイル・システムのロケーションにコピーしてから、セキュア・コピー (SCP) およびセキュア FTP (SFTP) コマンドを使用して、z/OS UNIX システム・サービスのファイル・システムのロケーションから宛先のマシンにコピーすることができます。

セキュア・デプロイメントでは、SCLM タイプ J2EEBLD に格納する必要があるサンプルのアクション・スクリプトは、BWBSCOPY および BWBSFTP です。

Developer Toolkit で ANT のデプロイ・スクリプトに SFTP (Secure File Transfer Protocol) または、SCP (Secure Copy Protocol) を使用するには、前提条件として IBM Ported Tools for z/OS (z/OS V1.4 およびそれ以降 - プログラム番号 5655-M23) の製品をインストールする必要があります。

IBM Ported Tools for z/OS とは z/OS プラットフォームでツールやアプリケーションを使用できるようにするためにデザインされた、無料のプログラム製品です。これらのアプリケーションは z/OS 環境で操作するために変更されています。IBM Ported Tools for z/OS は z/OS のライセンスをお持ちのお客様にのみご利用いただけます。z/OS 1.4 およびそれ以降でサポートされています。

以下の内容が含まれます。

- ネットワーク間でファイルをコピーするための SCP。これは RCP の代替手段です。
- 暗号化された SSH トランスポートにてファイルを転送するための SFTP。これは FTP に似た対話式のファイル転送プログラムです。

公開鍵認証

公開鍵認証により、Developer Toolkit のセキュア・デプロイメント操作の一部として、対話式ログオンの代わりに自動ログオンを使用できるようになります。

公開鍵認証が正しく機能するようにするには、代理ユーザー ID を使用してデプロイメントを実行するか、デプロイメントの機能を使用できるように各ユーザーを構成します。

ssh-agent と ssh-add を使用して、鍵ベースの認証を自動的に行うようセットアップする手順については、「IBM Ported Tools for z/OS User's Guide」を参照してください。SCLM Developer Toolkit 代理ユーザー ID を使用する方法については詳しくは、71 ページの『第 4 章 SCLM セキュリティー』を参照してください。

他のデプロイメント・オプション

独自の Ant スクリプトを作成して、さまざまな方法でデプロイメントを実行することもできます。独自のスクリプトでは、Ant <exec> タグを使用することで、z/OS UNIX システム・サービスのファイル・システムで使用する任意のプログラムを呼び出すことができます。このメソッドを使用すると、ビルド・スクリプトで FTP などのような他のプログラムを呼び出して、デプロイメントを実行できます。Ant スクリプトを作成する方法については、<http://ant.apache.org/> でオンラインの Ant ドキュメンテーションを参照してください。

ASCII または EBCDIC のストレージ・オプション

SCLM Developer Toolkit のプラグインから転送されたソース・ファイルは、ASCII または EBCDIC のいずれかで SCLM に格納できます。

通常、SCLM のすべてのソースは、z/OS の ISPF/SCLM から直接表示および編集できるように EBCDIC で格納されています。ホストからコードを直接参照したり、編集したりする必要がない場合は、元のクライアントの ASCII または UNICODE のコード・ページを使用して、ソースの格納先である SCLM にコードを直接格納する (バイナリー転送) ほうが便利な場合もあります。SCLM への格納、SCLM からの

インポートを行う大規模なプロジェクト、および ASCII から EBCDIC への言語変換が伴わない JAVA/J2EE ビルドでは、この方法によりパフォーマンスの改善が期待できます。

SCLM Developer Toolkit は、ファイルまたはメンバーに関連付けられた SCLM の言語を検査することで、ファイルがバイナリー転送か、ASCII から EBCDIC への変換を伴うかを判別します。さらに、SCLM Developer Toolkit では TRANSLATE.conf ファイルに TRANLANG キーワードを含む SCLM 言語のエントリが含まれるかどうかを検査します。

ASCII/EBCDIC 言語変換プログラム

SCLM

言語

変換プログラム 説明

JAVA	EBCDIC で格納された Java のソース・メンバー。サンプルの BWBTRAN1 を使用して作成。
SQLJ	EBCDIC で格納された SQLJ のメンバー。サンプル (BWBTRANS) を使用して作成。
JAVABIN	ASCII で格納された Java のソース・メンバー。サンプルの BWBTRAN1 を使用して作成。
J2EEPART	構文解析が不要で、EBCDIC で格納された任意の J2EE ファイル。サンプルの BWBTRAN1 を使用して作成。
J2EEBIN	構文解析が不要で、バイナリーまたは ASCII で格納された任意の J2EE ファイル。サンプルの BWBTRAN1 を使用して作成。
SQLJ	EBCDIC で格納された SQLJ のソース・メンバー。サンプル BWBTRANS を使用して作成。
SQLJBIN	ASCII で格納された SQLJ のソース・メンバー。サンプル BWBTRANS を使用して作成。
TEXT	構文解析が不要で、EBCDIC で格納されたデフォルトの TEXT 変換プログラム。サンプルの BWBTRAN1 を使用して作成。
BINARY	構文解析が不要なデフォルトの BINARY 言語変換プログラム。サンプルの BWBTRAN1 を使用して作成。

図 28. SCLM 言語変換プログラムと ASCII/EBCDIC

デフォルトを使用すると ASCII/EBCDIC 間の変換が想定されます。つまり、Eclipse プラグインで参照および編集されるファイルは、ISPF/SCLM を使用してホストから直接参照および編集することもできます。

プロジェクトのマイグレーションとインポート、およびビルドの実行には、ファイル変換が不要なため、ASCII の使用 (バイナリー転送) をお勧めします。これは ISPF/SCLM での編集が不要な場合に適しています。

使用する SCLM 言語変換プログラムに応じて、ソースを ASCII または EBCDIC のいずれかでビルドすることができます。

プラットフォーム間での使用可能度を上げるため、JAR、WAR、EAR などのすべてのデプロイ可能なファイルはビルドされます。こうすることで、ソースが EBCDIC で格納されているかどうかに関係なく、オブジェクトに含まれるすべてが ASCII タイプになります。

JAVA/J2EE ビルドの注意点: Java ソースが ASCII で格納されている場合は、ビルド・スクリプトで ENCODING プロパティ変数を使用して、ASCII コード・ページを指定し、Java ソースを正しくコンパイルする必要があります。

例:

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

呼び出された Ant スクリプトは Javac コマンドと ENCODING=ISO8859-1 とともに指定して、ASCII ソースをコンパイルします。デフォルトの ENCODING コード・ページは EBCDIC コード・ページの IBM-1047 です。

\$GLOBAL メンバー

JAVA/J2EE のビルド・プロセスの一部としてビルドを正常に実行するには、追加の情報が必要です。z/OS UNIX システム・サービスでビルドが実行されるため、Java の製品ロケーション、Ant の製品ロケーション、SCLM Developer Toolkit の構成ファイルのロケーション、作業域などの情報が必要になります。

さらに、\$GLOBAL メンバーはグループ固有のため、個々の SCLM の開発グループ用に、Ant または Java の異なるバージョンを使用しなければならない場合もあります。\$GLOBAL での環境変数の設定は、特定のビルド・スクリプト変数の設定で上書きされる場合があります。

| 注: ANT カスタマイズで指定した JAVA_HOME は、Java/J2EE プロジェクト・コン
| パイルのために \$GLOBAL で指定したすべての JAVA_BIN をオーバーライドし
| ます。

サンプル・メンバーの BWBGLOB は SBWBSAMP ライブラリーに用意されています。このサンプル・メンバーは、SCLM 階層の SCLM タイプ J2EEBLD にメンバー \$GLOBAL としてコピーし、TEXT (SISPMACS ライブラリーに含まれる言語変換プログラム FLM@TEXT で指定) などの有効な非構文解析言語を使用して保存する必要があります。

\$GLOBAL メンバーは現在、JAVA/J2EE ビルド・プロセスで以下の情報を使用できるようにします。

表 9. \$GLOBAL 変数

変数	説明
ANT_BIN	z/OS UNIX システム・サービスのファイル・システムの Ant ランタイム用ディレクトリー・パス 例: <property name="ANT_BIN" value="/u/antdirectory/Ant/apache-Ant-1.6.0/bin/Ant"/>

\$GLOBAL メンバー

表 9. \$GLOBAL 変数 (続き)

変数	説明
JAVA_BIN	z/OS UNIX システム・サービスのファイル・システムの Java コンパイル/ランタイム用ディレクトリー・パス 例: <pre><property name="JAVA_BIN" value="/u/javadirectory/IBM/J1.4/bin"/></pre>
CGI_DTWORK	SCLM Developer Toolkit の WORKAREA ディレクトリーのロケーション 例: <pre><property name="CGI_DTWORK" value="/var/SCLMDT"/></pre>
CGI_DTCONF	SCLM Developer Toolkit の CONFIG ディレクトリーのロケーション 例: <pre><property name="CGI_DTCONF" value="/etc/SCLMDT"/></pre>
CLASSPATH_JARS	Java のコンパイルに使用される z/OS UNIX システム・サービスのファイル・システムのクラスパス・ディレクトリー。このディレクトリーに配置されたすべての Jar はクラスパスで使用されます。 例: <pre><property name="CLASSPATH_JARS" value="/var/SCLMDT/CLASSPATH"/></pre>
TRANTABLE	ロング/ショートの名前変換を含む VSAM ファイル 例: <pre><property name="TRANTABLE" value="BWB.LSTRANS.FILE"/></pre>
DEBUG_MODE	Developer Toolkit で z/OS UNIX システム・サービスのファイル・システムから、Java/J2EE ビルド・ファイルを削除しないようにするには、「on」に設定します。 デバッグ目的で USS ファイル・システムにあるビルド済みの出力の構造を確認する場合、これは役立ちます。

これらの変数が SCLM プロジェクトのすべてのグループ・レベルで設定される場合は、階層の最上位レベルに単一の \$GLOBAL メンバーを作成するのは、有効な方法です。JAVA/J2EE のビルド変換プログラムを実行する場合、プログラムはビルドが実行されているグループ・レベルから階層をルックアップし、J2EEBLD タイプとして検出された最初の \$GLOBAL を使用します。

注: \$GLOBAL メンバーは有効な SCLM 保存済みメンバーとして格納する必要があります。そうすることで、階層のルックアップが実行できるようになります。

例えば、異なる開発グループで別の設定を必要としている場合は、各開発グループごとに \$GLOBAL メンバーを作成できます。

SITE とプロジェクト固有のオプション

SITE のインストール・レベルまたは固有の SCLM プロジェクト・レベルで特定の設定を行える機能が用意されています。現在、構成可能なオプションは、以下のとおりです。

- 必須変更コード・エントリー
- フォアグラウンド・ビルドとプロモートの使用不能化

- パッケージ承認システムの仕様。現在、IBM Breeze for SCLM が承認システムとしてサポートされています。
- バッチ・ビルド、ジョブ・カードのプロモートおよびマイグレーションの定義
- TRANSLATE.conf 構成ファイルの設定のオーバーライド
- プロジェクトのリスト・フィルター制限
- 双方向言語のデフォルト設定の定義

これらのオプションはすべて設定可能ですが、一部のみ設定することはできません。これらのオプションが設定されない場合は、プログラムによりデフォルトに設定されます。これらのオプションは SITE ファイルで設定可能で、それ以外のオプションは SCLM のプロジェクト固有のレベルで設定できます。その代わり SITE 固有のファイルは存在せず、オプションを設定できるのは SCLM のプロジェクト・レベルに限られます。IDE から入力した自身のジョブ・カードを使用して、ジョブ・カード情報をオーバーライドできます。

この機能は、z/OS UNIX システム・サービスのファイル・システムで、
/etc/SCLMDT/CONFIG/PROJECT ディレクトリーに特定のファイルを作成するか、インストール時に /CONFIG/PROJECT ディレクトリーを作成することで、有効にすることができます。このディレクトリーは、プロジェクトの初期化時にジョブ BWBINST1 を実行することで作成されます。

SITE 固有の値を設定する場合は、/PROJECT ディレクトリーに SITE.conf という名前のファイルを作成する必要があります。メンバー BWBSITE に含まれる SBWBSAMP ライブラリーに提供されているサンプルの SITE 構成ファイル。このメンバーをこのディレクトリーの SITE.conf という名前のファイルにコピーし、値を適宜調整します。次の図は、サンプルの SITE 構成ファイルを示します。

```
*****
*
* SCLM Developer Toolkit Site Specific option
*
*****
*
* Below are a number of site specific options used to
* determine the behavior of the Eclipse front-end.
* These can be overridden by creating a project specific
* options file for the SCLM project that overrides some
* or all of these options.
*
* SCM Approver processing applies to this project?
BUILDAPPROVER=NONE
PROMOTEAPPROVER=NONE
*
* Change Code entry on check-in is mandatory?
CCODE=N
*
*
* To allow promotion by architecture definition only,
* set the value of PROMOTEONLYFROMARCHDEF to Y
PROMOTEONLYFROMARCHDEF=N
*
* Foreground or On-line builds/promotes allowed for this project?
FOREGROUNDBUILD=Y
FOREGROUNDPROMOTE=Y
*
* Batch Build default jobcard
BATCHBUILD1=//SCLMBILD JOB (#ACCT),'SCLM BUILD',CLASS=A,MSGCLASS=X,
BATCHBUILD2=//          NOTIFY=,REGION=512M
BATCHBUILD3=//*
BATCHBUILD4=//*
*
* Batch Promote default jobcard
BATCHPROMOTE1=//SCLMPROM JOB (#ACCT),'SCLM PROMOTE',CLASS=A,MSGCLASS=X,
BATCHPROMOTE2=//          NOTIFY=&SYSUID,REGION=128M
BATCHPROMOTE3=//*
BATCHPROMOTE4=//*
*
* Batch Migrate default jobcard
BATCHMIGRATE1=//SCLMMIGR JOB (#ACCT),'SCLM MIGRATE',CLASS=A,MSGCLASS=X,
BATCHMIGRATE2=//          NOTIFY=&SYSUID,REGION=128M
BATCHMIGRATE3=//*
BATCHMIGRATE4=//*
*
* Batch Deployment default jobcard
BATCHDEPLOY1=//SCLMDPLY JOB (#ACCT),'SCLM DEPLOY',CLASS=A,MSGCLASS=X,
BATCHDEPLOY2=//          NOTIFY=&SYSUID,REGION=128M
BATCHDEPLOY3=//*
BATCHDEPLOY4=//*
*
* BUILD Security flag for SAF/RACF security call and possible Surrogate ID switch
BUILDSECURITY=N
*
* Project list flag if set to N will stop users selecting * as project filter
* This may avoid long user catalog searches for all SCLM projects
*
PROJECTLISTALL=Y
```

図 29. サンプルの SITE 固有の SCLM プロジェクト設定

プロジェクト固有の構成設定を指定して、これを使用して単一の SCLM プロジェクトを構成することもできます。これらは SITE.conf が存在する場合に、SITE 固有の値をオーバーライドします。プロジェクト固有の値を設定する場合は、/PROJECT ディレクトリーに project.conf という名前のファイル（ここで project は SCLM

のプロジェクト名を示します) を作成する必要があります。メンバー BWBPROJ に含まれる SBWBSAMP ライブラリーに提供されているサンプルのプロジェクト構成ファイル。このメンバーを /PROJECT ディレクトリーの project.conf という名前のファイルにコピーし、値を適宜調整します。次の図は、サンプルのプロジェクト構成ファイルを示します。

```
*
* ----- PROJECT SPECIFIC OPTIONS -----
*
* Below are a number of project specific options used to
* determine the behavior of the Eclipse front-end.
*
* These will override the SITE.CONF file
*
*
* SCM Approver processing applies to this project?
BUILDAPPROVER=BREEZE
PROMOTEAPPROVER=BREEZE
*
* Change Code entry on check-in is mandatory?
CCODE=Y
*
* Foreground or On-line builds/promotes allowed for this project?
FOREGROUNDBUILD=N
FOREGROUNDPROMOTE=N
*
* Batch Build default jobcard
BATCHBUILD1=//SCLMBILD JOB (#ACCT),'SCLM BUILD',CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID,
BATCHBUILD2=// MSGLEVEL=(1,1)
BATCHBUILD3=//*
BATCHBUILD4=//*
*
* Batch Promote default jobcard
BATCHPROMOTE1=//SCLMPROM JOB (#ACCT),'SCLM PROMOTE',CLASS=A,MSGCLASS=X,
BATCHPROMOTE2=// MSGLEVEL=(1,1),NOTIFY=&SYSUID
BATCHPROMOTE3=//*
BATCHPROMOTE4=//*
```

図 30. サンプルのプロジェクト固有の SCLM プロジェクト設定

オプション定義

すべてのオプションは任意です。オプションは製品によってデフォルト値に設定されます。オプションが SITE.conf または project.conf に指定されている場合は、そのオプションが使用されます。

表 10. SITE/プロジェクト・オプション

BUILDAPPROVER=approval product/ <u>NONE</u>	ビルド・プロセスに使用される承認製品の名前を指定します。現在サポートされている製品は、Breeze for SCLM のみです。デフォルトは NONE です。
---	---

表 10. SITE/プロジェクト・オプション (続き)

PROMOTEAPPROVER=approval product/ <u>NONE</u>	プロモート・プロセスに使用される承認製品の名前を指定します。現在サポートされている製品は、Breeze for SCLM のみです。 PROMOTEAPPROVER が BREEZE に設定されている場合は、プロモート中に Breeze 固有のフィールドが表示されます。デフォルトは NONE です。
CCODE= <u>N</u> /Y	コード・エントリーに変更を加えるには、必須フィールドのチェックインに Y を指定します。デフォルトは N で、この場合エントリー・コードの変更は必須ではありません。
FOREGROUNDBUILD= <u>Y</u> /N	フォアグラウンド・ビルドを制限するには、N を指定します。デフォルトは Y で、この場合フォアグラウンド・ビルドは許可されます。
FOREGROUNDPROMOTE= <u>Y</u> /N	フォアグラウンド・プロモートを制限するには、N を指定します。デフォルトは Y で、この場合フォアグラウンド・プロモートは許可されます。
BATCHBUILD1=Job card 1 BATCHBUILD2=Job Card 2 BATCHBUILD3=Job Card 3 BATCHBUILD4=Job Card 4	ビルド・プロセスにデフォルトのバッチ・ジョブ・カードを設定します。個々のプロジェクトでそれぞれのアカウント・コードまたはジョブ・クラスを使用するには、このオプションでプロジェクト固有のジョブ・カードを指定します。
BATCHPROMOTE1=Job card 1 BATCHPROMOTE2=Job card 2 BATCHPROMOTE3=Job card 3 BATCHPROMOTE4=Job card 4	プロモート・プロセスにデフォルトのバッチ・ジョブを設定します。個々のプロジェクトでそれぞれのアカウント・コードまたはジョブ・クラスを使用するには、このオプションでプロジェクト固有のジョブ・カードを指定します。
BATCHMIGRATE1=Job card 1 BATCHMIGRATE2=Job card 2 BATCHMIGRATE3=Job card 3 BATCHMIGRATE4=Job card 4	マイグレーション・プロセスにデフォルトのバッチ・ジョブを設定します。個々のプロジェクトでそれぞれのアカウント・コードまたはジョブ・クラスを使用するには、このオプションでプロジェクト固有のジョブ・カードを指定します。
BUILDSECURITY= <u>Y</u> /N	SAF/RACF のセキュリティー呼び出しと、可能性のある代理 ID の切り替えを呼び出すには、Y を指定します。この詳細については、71 ページの『第 4 章 SCLM セキュリティー』を参照してください。
PROMOTESecurity= <u>Y</u> /N	SAF/RACF のセキュリティー呼び出しと、可能性のある代理 ID の切り替えを呼び出すには、Y を指定します。この詳細については、71 ページの『第 4 章 SCLM セキュリティー』を参照してください。

表 10. SITE/プロジェクト・オプション (続き)

DEPLOYSECURITY=Y/N	SAF/RACF のセキュリティー呼び出しと、可能性のある代理 ID の切り替えを呼び出すには、Y を指定します。この詳細については、71 ページの『第 4 章 SCLM セキュリティー』を参照してください。
ASCII=ASCII codepage	TRANSLATE.conf ファイルで指定された ASCII コード・ページをオーバーライドする、ASCII コード・ページを指定します。例: ASCII=UTF-8
EBCDIC=EBCDIC codepage	TRANSLATE.conf ファイルで指定された EBCDIC コード・ページをオーバーライドする、EBCDIC コード・ページを指定します。例: EBCDIC=IBM-420
TRANLANG=SCLM Language	TRANSLATE.conf で指定された TRANLANG パラメーターのリストに、追加する TRANLANG パラメーターを指定します。例: TRANLANG=DOC
NOTRANLANG=SCLM Language	TRANSLATE.conf で指定された SCLM プロジェクトで有効な TRANLANG のリストから、あらかじめ指定された TRANLANG を削除するために NOTRANLANG キーワードを使用します。例: NOTRANLANG=JAVA
LOGLANG=SCLM Language	TRANSLATE.conf で指定された LOGLANG パラメーターのリストに、追加する LOGLANG パラメーターを指定します。例: LOGLANG=DOC
NOLONGLANG=SCLM Language	TRANSLATE.conf で指定された SCLM プロジェクトで有効な LOGLANG のリストから、あらかじめ指定された LOGLANG を削除するために NOLONGLANG キーワードを使用します。例: NOLONGLANG=COBOL

表 10. SITE/プロジェクト・オプション (続き)

BIDIPROP=LANG=SCLM Language/* TextOrient=LTR/RTL TextType=Visual/ Logical SymetricSwap=On/Off NumericSwap=On/Off	BIDIPROP キーワードを使用して、BiDirectional 言語を SCLM 言語のデフォルトに指定します。LANG= はすべての SCLM 言語または特定の SCLM 言語のいずれかに設定できます。双方向言語サポートは Rational Developer for System z でのみサポートされています。詳しくは、「Rational Developer for System z Host Configuration Guide」を参照してください。
PROJECTLISTALL=Y	プロジェクトのリスト・フラグが N に設定されていると、ユーザーがプロジェクト・フィルターとして * を選択することを停止します。これにより、全 SCLM プロジェクトの長時間のユーザー・カタログ検索を回避できます。

TRANSLATE.conf のオーバーライドの併用例

TRANSLATE.conf ファイルは SCLM Developer Toolkit のインストール時に適用される、コード・ページのサポート設定と、SCLM の言語サポートのデフォルト設定をセットアップします。追加または変更できるデフォルトのサンプル・リストを次に示します。

```
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
```

異なるタイプのデータを、場合によっては異なる各国語で格納している、個々の SCLM プロジェクトでは、このデフォルト設定がオーバーライドされる可能性があります。例えば、SCLM プロジェクト SCLMPRJ1 のプロジェクト構成ファイルに、以下の TRANSLATE.conf のオーバーライド設定が含まれているとします。

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

これはソースの言語変換をアラビア語のコード・ページのペアにするよう、コード・ページを設定します。さらに、DOC の SCLM 言語が TRANSLATE.conf からデフォルトとして追加されます。

SCLM プロジェクト SCLMPRJ2 に、以下の異なる TRANSLATE.conf 設定が含まれているとします。

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

これはソースの言語変換をヘブライ語のコード・ページのペアにするよう、コード・ページを設定します。さらに、DOC と XLS の SCLM 言語が TRANSLATE.conf からデフォルトとして追加されます。ただし、この場合、TRANSLATE.conf のデフォルト設定は削除されます。このことはそれ程必要ではなく、追加の言語指定があっても問題にはなりません。しかしこれは、固有の SCLM プロジェクトに必要な SCLM 言語のみを含むようにプロジェクトをセットアップする方法を示しています。

BIDIPROP のオーバーライドの併用例

SITE.conf ファイルに指定された BIDIPROP の値は、SCLM プロジェクト固有の project.conf ファイルで指定された任意の BIDIPROP 値によりオーバーライドされる可能性があります。例えば、以下のように SITE.conf で設定されているとします。

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

これはすべての SCLM 言語を指定された設定に設定します。ここで、以下のように ADMIN10.conf ファイルで設定できます。

```
*
* 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
```

BIDIPROP のオーバーライドの併用例

これらの設定は JAVA と COBOL の言語定義を行う SITE.conf の設定をオーバーライドします。すべての他の言語では SITE.conf に定義された設定がデフォルトで設定されます。

第 4 章 SCLM セキュリティー

SCLM Developer Toolkit は、ビルド、プロモート、およびデプロイ機能に対して、必要に応じてオプションのセキュリティー機能を提供します。

- SCLM Developer Toolkit 構成ファイルで設定されるセキュリティー・フラグ、および SAF/RACF 規則によって制御されるセキュリティー。
- ある機能に対してセキュリティー・フラグが設定され、要求元のユーザー ID が SAF/RACF 権限検査を通れば、SCLM ビルド、プロモート、またはデプロイは、要求元のユーザー ID のもとで続行するか、その SAF/RACF 規則で定義された場合はオプションとして代理ユーザー ID のもとで実行できます。

ビルド/プロモート/デプロイ・セキュリティー・フラグおよびプロセス・フロー

サイト/プロジェクト構成ファイルでビルド、プロモート、デプロイ・セキュリティー・フラグを設定できます。このフラグは、その機能および代理ユーザー ID のもとの可能な処理に対して追加の SAF セキュリティー検査を指定します。

- サイト/プロジェクト構成ファイルを読み取って、ビルド・セキュリティー・オプションがオンに設定されているかどうかを確認します。

BUILDSECURITY = Y

- サイト/プロジェクト構成ファイルを読み取って、プロモート・セキュリティー・オプションがオンに設定されているかどうかを確認します。

PROMOTESecurity = Y

- サイト/プロジェクト構成ファイルを読み取って、デプロイ・セキュリティー・オプションがオンに設定されているかどうかを確認します。

DEPLOYSECURITY = Y

セキュリティー・フラグが設定されている場合:

- 機能要求からサブパラメーターを読み取った後、変換プログラムは SCLM セキュリティー・モジュールを呼び出して、定義済みの規則に照らして SAF/RACF ユーザー権限を検査します。
- 許可されている場合は、その規則に関連する SAF/RACF アプリケーション・データが代理ユーザー ID (suid=xxxxxxx) として読み取られます。代理ユーザー ID が存在する場合は、現在のタスクの有効ユーザー ID が新しい代理ユーザー ID に切り替わり、代理ユーザー ID が存在しない場合は、発信ユーザー ID のもとで処理が続行されます。

「ビルド、プロモート (Build, Promote)」パネルからデータ・セット命名規則を選択できます。

- 許可されていない場合は、ビルド、プロモート、またはデプロイでセキュリティー・エラー・メッセージとともに RC=8 が戻されます。ビルド、プロモート、またはデプロイ処理は取り消されます。

セキュリティ規則および代理ユーザー ID

SCLM ビルド、プロモート、およびデプロイ機能のセキュリティは、SAF/RACROUTE セキュリティ・インターフェースを使用して定義します。このインターフェースをサポートすることが確認されているセキュリティ製品は RACF、ACF2、および TOP SECRET です。

以下の例は、定義および許可コマンド RDEFINE および PERMIT を使用する RACF をモデルとしたものです。SAF をサポートする他のセキュリティ製品には、同等の定義のための独自のコマンドがあります。これらの定義については、該当するセキュリティ製品の資料を参照してください。

- RDEFINE コマンドを使用して、クラス記述子テーブルで指定されたクラスに属するすべてのリソースを RACF に対して定義します。RDEFINE コマンドは、リソースのプロファイルを RACF データベースに追加して、そのリソースへのアクセスを制御します。
- PERMIT コマンドは、特定のリソースへのアクセスを許可されているユーザーとグループのリストを維持管理するために使用されます。標準アクセス・リストには、そのリソースへのアクセスを許可されているユーザー ID とグループ名、およびそれぞれに付与されているアクセス・レベルが含まれます。
- SCLM 機能の機能プロファイルを XFACILIT クラスに対して定義します。
- セキュリティ管理者は、RDEFINE コマンドを使用して必要な RACF プロファイルを定義し、PERMIT コマンドを使用してアクセスを許可します。

ビルド規則形式

ビルドの SCLM プロファイル定義の形式を次に示します。

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

フィールド・サブパラメーターの詳細:

- SCLM: SCLM 機能プロファイルを示します
- BUILD: SCLM からのビルド機能を示します
- Project: SCLM プロジェクト、または、すべてのプロジェクトを表す場合は *
- Projdef: 代替プロジェクト定義 (デフォルト・プロジェクト)、または、すべての代替プロジェクトを表す場合は *
- Group: ビルドする SCLM グループ、または、すべてのグループを表す場合は *
- Type: SCLM タイプ、または、すべてのタイプを表す場合は *
- Member: ビルドする SCLM メンバー、または、すべてのメンバーを表す場合は *

プロモート規則形式

プロモートの SCLM プロファイル定義の形式を次に示します。

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

フィールド・サブパラメーターの詳細:

- SCLM: SCLM 機能プロファイルを示します
- PROMOTE: SCLM からのプロモート機能を示します
- Project: SCLM プロジェクト、または、すべてのプロジェクトを表す場合は *
- Projdef: 代替プロジェクト定義 (デフォルト・プロジェクト)、または、すべての代替プロジェクトを表す場合は *

- Group: ビルドする SCLM グループ、または、すべてのグループを表す場合は *
- Type: SCLM タイプ、または、すべてのタイプを表す場合は *
- Member: プロモートする SCLM メンバー、または、すべてのメンバーを表す場合は *

デプロイ規則形式

デプロイメントの SCLM プロファイル定義の形式を次に示します。

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

フィールド・サブパラメーターの詳細:

- SCLM: SCLM 機能プロファイルを示します
- DEPLOY: SCLM からのデプロイ機能を示します
- Server: ターゲット・デプロイメント・サーバー - デプロイ Ant スクリプトの SERVER_NAME、または、すべてのサーバーを表す場合は *
- Application: ターゲット WAS アプリケーション名 - Ant スクリプトの APPLICATION_NAME、または、すべてのアプリケーションを表す場合は *
- Node: ターゲット WAS ノード名 - Ant スクリプトの NODE_NAME、または、すべてのノードを表す場合は *
- Cell: ターゲット WAS セル名 - デプロイ Ant スクリプトの CELL_NAME、または、すべてのセルを表す場合は *
- Project: デプロイする EAR を含む SCLM プロジェクト、または、すべてのプロジェクトを表す場合は *
- Projdef: 代替プロジェクト定義 (デフォルト・プロジェクト)、または、すべての代替プロジェクトを表す場合は *
- Group: デプロイ可能な EAR が保管されている SCLM グループ、または全グループの場合は *
- Type: デプロイ可能な EAR が保管されている SCLM タイプ、または全タイプの場合は *

SAF/RACF ビルド、プロモート、デプロイ、およびプロファイルに関する規則

セキュリティー管理者が定義するクラス XFACILIT を使用するビルド・プロファイル規則の例を次に示します。 SCLM.BUILD.* 規則を SCLM.PROMOTE.* 規則で置き換えることにより、プロモート規則と同じサンプルを使用できます。

注: 標準 RACF 規則の場合と同様、特定のパラメーターの代わりに汎用「*」フィールドを使用できます。適切な SAF/RACF ルールと一致する場合、そのルールに基づく読み取り権限が付与されていれば、ビルド・アクセスが許可されます。この規則のデフォルト・アクセスは NONE → UACC(NONE) になります。アクセス規則は PERMIT コマンドによって定義されます。

次の規則は、project=TESTPROJ でグループ・レベル prod のビルドについて、すべてのメンバーを保護します。また、代理ユーザー ID はアプリケーション・データ APPLDATA('SUID=xxxxx') に基づいた規則で保管されます。

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

この例は SCLM ビルド・プロファイルを定義しています。

- プロジェクト = TESTPROJ

SAF/RACF ビルド、プロモート、デプロイ、およびプロファイルに関する規則

- 代替プロジェクト定義 = TESTPROJ
- SCLM グループ = PROD
- SCLM タイプ = すべてのタイプ
- メンバー = すべてのメンバー

注: PMEANEY の代理ユーザー ID はアプリケーション・データに保管されます。

次の例は、上記の例で個々のユーザーおよびユーザー・グループに対して定義されたセキュリティ・アクセス権を示しています。これもセキュリティ管理者によって定義されます。

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

ワイルドカード文字を使用すれば、PERMIT は元の RDEFINE プロファイルと一致するので、ユーザー HOGES がプロジェクト TESTPROJ およびグループ PROD から任意のメンバーをビルドできるようにします。一致規則の「APPLICATION DATA」には代理ユーザー ID が保管されているため、ビルドは代理ユーザー ID (この場合は PMEANEY) のもとで処理されます。

セキュリティ管理者が定義するクラス XFACILIT を使用するデプロイメント・プロファイルの例を次に示します。

```
RDEFINE  
XFACILIT SCLM.DEPLOY.srvery.testapp.node1.cell1.TESTPROJ.TESTPROJ.PROD.J2EEDEP  
UACC(NONE)
```

これは WAS の SCLM デプロイ・プロファイルを定義しています。

サーバー名 = srvery
アプリケーション名 = testapp
ノード名 = node1
セル名 = cell1

SCLM プロジェクトの詳細:

プロジェクト = TESTPROJ
代替プロジェクト定義 = TESTPROJ
SCLM グループ = PROD
SCLM タイプ = J2EEDEP

次の 2 つの例は、上記の例で個々のユーザーおよびユーザー・グループに対して定義されたセキュリティ・アクセス権を示しています。これもセキュリティ管理者によって定義されます。プロファイルの UACC 権限は ACCESS(NONE) になります。

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

ワイルドカード文字を使用すれば、PERMIT は元の RDEFINE プロファイルと一致するので、ユーザー HOGES が WAS サーバー=srvery、アプリケーション=testapp、ノード=node1、セル=cell1、SCLM プロジェクト=TESTPROJ、プロジェクト定義=TESTPROJ、グループ=PROD、およびタイプ=J2EEDEP でデプロイできるようにします。

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

SAF/RACF ビルド、プロモート、デプロイ、およびプロファイルに関する規則

これは元の RDEFINE プロファイルと一致するので、RACF グループ J2EEGRP に属するユーザーは上記のサーバーで同じ SCLM プロジェクト詳細からデプロイすることができます。

第 5 章 CRON 開始のビルドおよびプロモート

大部分のビルドとプロモートは Developer Toolkit クライアントを介して開始されますが、z/OS UNIX システム・サービス・ファイル・システム内でビルドおよびプロモート構成ファイルをセットアップし、これらのビルドまたはプロモートを UNIX システム・サービス内の CRON (タイム) サービスを介して開始することができます。この方法を使用すれば、関連するビルドおよびプロモート・パラメーターが z/OS UNIX システム・サービス・ファイル・システム構成ファイルから読み取られ、SCLM 処理のために Developer Toolkit ホスト・コンポーネントに渡されるので、SCLM Developer Toolkit クライアントは必要なくなります。

以下は、CRON 開始のビルドおよびプロモートを提供する SCLM Developer Toolkit サンプルの説明です。これらのサンプルは、インストール済みの Developer Toolkit SBWBSAMP データ・セットに入っています。

サンプル・メンバー

説明

- BWBCRON1** この REXX サンプルは SCLM Developer Toolkit ホスト・インターフェースを呼び出し、関数仮パラメーターを渡します。デフォルトでは関数プロセスからの出力は STDOUT に表示されますが、この出力を z/OS UNIX システム・サービス・ファイル・システム内のファイルまたはログにリダイレクトすることもできます。
- このサンプルは、z/OS UNIX システム・サービス・ファイル・システムの任意のディレクトリーにコピーして実行できます。サンプル内の詳細説明に従ってサンプルをカスタマイズする必要があります。
- この REXX サンプルは、サンプル BWBCRONB (ビルドの場合) またはサンプル BWBCRONP (プロモートの場合) からの入力を使用して実行する必要があります。
- BWBCRONB** この REXX サンプルは、モジュール BWBCRON1 に渡されるビルド・パラメーター入カストリングをセットアップします。
- このサンプルは、すべての必須ビルド・パラメーターを更新するようにカスタマイズする必要があります。
- このサンプルは、ユーザー指定の z/OS UNIX システム・サービス・ファイル・システム・ディレクトリー (必要に応じて名前の変更可能) にコピーして、サンプル BWBCRON1 とともに実行する必要があります。
- BWBCRONP** この REXX サンプルは、モジュール BWBCRON1 に渡されるプロモート・パラメーター入カストリングをセットアップします。
- このサンプルは、すべての必須プロモート・パラメーターを更新するようにカスタマイズする必要があります。
- このサンプルは、ユーザー指定の z/OS UNIX システム・サービス・ファイル・システム・ディレクトリー (必要に応じて名前の変更可能) にコピーして、サンプル BWBCRON1 とともに実行する必要があります。

図 31. サンプル CRON メンバー

STEPLIB および PATH に関する要件

システム共通プロファイル (/etc/profile) またはユーザー・プロファイル (/u/userid/.profile) に存在する PATH および STEPLIB 変数は、CRON ジョブ (\$PATH) を検索し、SCLM Developer Toolkit データ・セットが LINKLIST に入っていない場合は SCLM Developer Toolkit モジュール (\$STEPLIB) を検索するように設定する必要があります。

例:

サンプル BWBCRON1 および BWBCRONB は、テスト・ディレクトリー /var/SCLMDT/CRONJOBS にコピーされます。次の z/OS UNIX システム・サービス・ファイル・システム PATH および STEPLIB 変数が、/etc/profile に設定されています。

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

CRON ビルド・ジョブの実行

CRON ジョブが PATH 変数に追加されると、出力を `parameter_exec` から `processing_exec` にパイピングすることによって CRON ジョブを実行できます。このとき、出力を出力ログ・ファイルに送ることができます。

構文

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

「|」は z/OS UNIX システム・サービスのパイプ記号です。

呼び出し例:

提供されたサンプル名を使用し、次のように CRON ビルド `exec` を呼び出すことができます。

```
BWBCRONB | BWBCRON1 >bwbcrnb.log
```

さらに、このサンプル・ビルド実行を、月曜から金曜までの午前 7 時 30 分に実行するように CRONTAB ファイルに追加できます。

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

使用可能な CRON サービスと CRONTAB 形式について詳しくは、次のマニュアルを参照してください。

- z/OS UNIX システム・サービス コマンド解説書
- z/OS UNIX システム・サービス 計画

あるいは、z/OS UNIX システム・サービスのもとでオンライン・マニュアル・ヘルプ (man) を使用してください。

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

CRON ビルド・ジョブのサンプル

以下は、SBWBSAMP ライブラリーで提供される BWBCRON1 および BWBCRONB ジョブのサンプルです。


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

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

BWBINT

EXIT
```

図 32. サンプル CRON ビルド Exec

```

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

```

図 33. サンプル・ビルド・パラメーター・ファイル

付録 A. SCLM の概要

IBM SCLM Developer Toolkit は、Eclipse で作成された分散アプリケーションを、SCLM (IBM z/OS ソース・コード管理システム) を使用して管理、ビルドできるようにする手段を提供します。

分散およびメインフレーム・ユーザーが使用する言語とツールは、これらのユーザーが使用する環境によって異なります。両方の環境の主要概念を確認して理解すれば、これらの言語とツールを統合構造にうまく組み込むことができます。

Developer Toolkit は、アプリケーション構造の観点から見ると一連の Eclipse プラグインで、HTTP トランスポートと RSE トランスポートの両方を使用可能にする、対応する z/OS ホスト・コードを備えています。操作の面から見ると、Eclipse デベロッパーはワークスペース・プロジェクトを SCLM に登録します。プロジェクト内のファイルは、SCLM プロジェクトに追加し、チェックインおよびチェックアウトし、必要に応じてビルドおよびデプロイすることができます。これらのサービスはすべて、「チーム・アクション (Team Actions)」メニューを介して実行されます。SCLM 管理者の立場からすると、管理者はプロジェクト、タイプ、言語、および関連するビルド変換プログラムを作成できます。変更コードや許可コードなどのフィーチャーは、要件によって異なります。

SCLM の概念

Java/J2EE の開発者の観点からすれば、SCLM の説明には以下の概念が役立ちます。

ファイル命名

z/OS ファイル・システムは、8 文字の長さのファイル名のみをサポートします。Developer Toolkit インターフェースは、通常の Java ロング・ネーム規則のサポートを可能にする変換サービスを提供します。命名上の制約に従わなければならない SCLM 固有のファイルがあります。これらのファイルは主として、44 ページの『J2EE ARCHDEF』で説明されている ARCHDEF 構造に関係しています。

タイプ

SCLM プロジェクトで保管される各ファイル (SCLM の用語ではメンバーと呼ぶ) は、データ・セットに保管されます。ファイルが保管されているデータ・セットは、SCLM タイプによって識別されます。タイプはデータ・セット名の一部であり、SCLM は SCLM Project.Group.Type. によってタイプと関連する言語に関係づけられます。SCLM プロジェクトでは、多数のタイプを定義することができます。これらのタイプにより、同じ名前の 2 つのファイルを同じ SCLM プロジェクトに保管することができます。各プロジェクトは多数のタイプを含むことができます。タイプの使用を適用すれば、1 つの SCLM プロジェクトに複数の Eclipse プロジェクトを保管できます。これは、各 IDE プロジェクトが潜在的に .project および

.classpath ファイルを持っている場合でも可能です。タイプを使用してこれらのファイルを区別しておかないと、それぞれのファイルのコピーは SCLM 内で 1 つのみになります。

SCLM タイプの定義は、SCLM 管理者が行います。SCLM とプロジェクトを共有するときは、SCLM にオブジェクトを保管するときに使用するタイプを知っておく必要があります。

言語

ファイルを SCLM に追加するときは、一定の言語定義を使用してそのファイルを保管する必要があります。言語の定義も SCLM 管理者が行います。この定義は、ホスト・システムとの間でファイルをやり取りするときに SCLM の動作を制御します。言語定義を使用すれば、特定のファイル・タイプについて、ロング・ネーム変換を行うか、バイナリー・オブジェクトとして保管するか、ASCII または EBCDIC (ネイティブ z/OS エンコード) に変換するかを定義することができます。例えば、JAVABIN という言語定義は、ロング・ネームに変換されるバイナリー・オブジェクトに関連することがあります。あるいは、WEBHTML はロング・ネームに変換して ASCII で保管するテキスト・ファイルを表すように定義できます。言語定義の数はプロジェクトごとに定義されます。保管されたファイルを SCLM から正しく取得できるようにするためには、使用する言語を理解しておく必要があります。言語はまた、SCLM によるオブジェクトのビルド方法も定義します。

SCLM のプロパティ

SCLM の制御下にあるファイルには、いくつかのプロパティが関連付けられます。これらのプロパティは事実上、IDE ファイルと対応する SCLM プロパティ間のマッピング・メカニズムです。SCLM に対してサービス要求が出されると、適切なサービス・パラメーターを形成するために、このデータが読み取られます。これらのプロパティは、Eclipse から SCLM 制御のメンバーを強調表示すると、「プロパティ (Properties)」メニューで確認できます。

SCLM プロジェクト構造

SCLM とプロジェクトを共有するときは、自分の所属する開発グループも指定する必要があります。SCLM プロジェクト構造は実際は階層構造です。標準的な階層は次のとおりです。

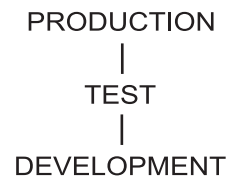


図 34. 複数タイプ

コードは最初、DEVELOPMENT レベルで保管されます。ビルドとテストが正常に完了すると、TEST までプロモートされます。テストが正常に完了すると、PRODUCTION にプロモートされます。これは一般に、開発済み製品を表します。実動レベルのコードで問題点が検出されると、その問題点を修正するために編集しなければならないファイルが開発レベルまで下位コピーされ、ビルド・プロセスが

再び開始されます。アプリケーションを構成するすべてのコードが開発レベルまで下位コピーされるわけではありません。SCLM は、ビルドを構成するエレメント、およびそのエレメントが保管されているレベルを記録します。

開発レベルには複数のグループが存在する場合があります。このため、開発レベルで保管されたコードを分離することができます。SCLM はまた、さまざまな開発グループに保管されたコードの動作をプロモート能力の観点から判別するコントロールも提供します。

ARCHDEF

IDE プロジェクトの構造は一般に、1 つ以上の IDE プロジェクトから構成されています。各 IDE プロジェクトを異なる SCLM タイプで保管することで、この構造は維持管理されます。したがって、ARCHDEF ファイルは事実上、IDE プロジェクトを構成するファイルを定義します。各 SCLM プロジェクトには複数の ARCHDEF があります。ARCHDEF は他の ARCHDEF を参照できるので、この複数 IDE プロジェクト構造をビルド・プロセスに対して定義することができ、ARCHDEF は SCLM に対してビルド・リストを定義するための主要な手段となります。この最も近い例は Make プロセスです。ARCHDEF は、外部 JAR またはクラスのロケーションを指定できるビルド・スクリプトを指定するほかに、ビルドを構成するファイルのリストも行います。詳しくは、オンライン・ヘルプ・システムの「ユーザー・マニュアル」セクションを参照してください。

JAVA/J2EE の概念

IDE プロジェクトがワークスペースに作成されると、プロジェクト説明ファイルが自動的に生成され、**.project** という名前で保管されます。この XML 文書には、プロジェクトに関する「ビルダー」または「ネーチャー」の説明が含まれます。「ビルダー」とは、プロジェクト・コンテンツに基づいてビルド状態を作成するインクリメンタル・プロジェクト・ビルダーのことです。プロジェクトのコンテンツが変わると、このファイルが更新されます。「ネーチャー」とは、与えられたプロジェクトと特定のプラグインまたはフィーチャーとの間の関連を定義し管理するものです。

.classpath ファイルはパスを記述するファイルで、このパスは、IDE プロジェクト内のソース・コードが参照する外部 JAR またはクラスを検出するために使用されます。SCLM Developer でのビルド時の同等な機能は、Ant ビルド・スクリプト内の CLASSPATH_JARS ディレクティブで定義されます。このディレクティブは、IDE プロジェクト内のソース・コードが参照する外部 JAR またはクラスを検出するために使用される、z/OS ホスト上のパスを記述します。

.classpath と **.project** の両方が IDE プロジェクト構成を保存するために使用されるので、IDE プロジェクト構成は別のワークスペースでも再作成できます。このため、この両者を IDE プロジェクトの一部として SCLM にチェックインすることをお勧めします。

プロジェクト開発の重要な一面は (特に J2EE プロジェクトでは)、さまざまな形式のアプリケーション実行可能ファイルを多数作成できることです。Java プロジェクト実行可能ファイルは、多くの場合、JAR、WAR、RAR、または EAR ファイルとしてパッケージされます。

JAR ファイルは一般に、標準 Java アプリケーションまたは Enterprise Java Beans (EJB) に関係します。

WAR ファイルは、Web アプリケーション用に作成されます。一般に、このファイルは Java サーブレット、JSP、および HTML ファイルで構成されます。WAR アプリケーションは、多くの場合、Web ベース・アプリケーションのフロントエンドです。このアプリケーションは、特定のサービスのために EJB など他の JAR も参照できます。各 WAR ファイルには **web.xml** ファイルが含まれます。このファイルは、WAR アプリケーションが使用する Java、HTML、およびライブラリー・サービスに関する構成を記述するものです。

RAR ファイル開発は現在、Developer Toolkit ではサポートされていません。

EAR ファイルはエンタープライズ・アプリケーションを表します。このアプリケーションは、JAR ファイルと WAR ファイルで構成されます。J2EE 言語では、EAR ファイルの作成は、その構成要素である JAR ファイルと WAR ファイルの集合です。この方法により、実際には特定のコンポーネント (JAR/WAR) で構成される EAR アプリケーションを作成できます。このアプリケーションの実際の構成は、**application.xml** ファイルに記述されています。EAR ファイル自身は独立型の実行可能オブジェクトではありません。EAR ファイルは、Websphere Application Server (WAS) などの J2EE コンテナにインストールする必要があります。EAR ファイルのインストールは**デプロイメント**と呼ばれます。デプロイした EAR アプリケーションには WAS 環境を介してアクセスできます。デプロイメントはビルドとは別のプロセスです。デプロイメントは、EAR アプリケーションの物理的インストールを必要とします。

したがって、J2EE アプリケーションを開発するときは、WAR ファイルや JAR ファイルなどの独立したコンポーネントを多数開発する必要があることがあります。これらのファイルは EAR ファイルに統合されます。そうすると、EAR ファイルは、操作のために J2EE コンテナ (例えば、WAS) にデプロイする準備が整います。

Eclipse ワークスペース内では、プロジェクトは事実上近接しています。つまり、IDE 環境内では、プロジェクトはパッケージ化に際して容易に他の IDE プロジェクトを効果的に参照できます。SCLM 内では、各 IDE プロジェクト (例えば、WAR、JAR、および EAR プロジェクト) は単一の SCLM プロジェクトにマップする必要があります。各プロジェクトは異なる SCLM タイプを使用して区別されます。その理由は、多数の IDE プロジェクト

(.project、.classpath、web.xml、application.xml など) で共通のファイル名が使用されるため、別々のタイプを使用することで同じ名前のパーツが同じ SCLM プロジェクトに存在できるからです。マッピングについて詳しくは、53 ページの『J2EE プロジェクトの SCLM へのマッピング』を参照してください。

SCLM の観点からは、EAR アプリケーションの開発は、ハイ・レベルの ARCHDEF 構造を使用することで最も参照しやすくなります。SCLM 内では、ハイ・レベル ARCHDEF (多くの SCLM プロジェクトではパッケージと呼ばれる) は、ARCHDEF 構造の頂点の後に EAR アプリケーションと EAR アプリケーションを構成する下位参照 (WAR および JAR ファイル) が続きます。この構造により、上位と下位レベルの両方でビルドを使用でき、さらにフル・ビルドや条件付き

ビルドも使用できます。このように、ARCHDEF は、SCLM プロジェクト内の J2EE プロジェクト・エレメントも定義できる手段を提供します。

付録 B. SQLJ Support

SQLJ は Java の言語拡張です。Java プログラマーが、プログラムの中にデータベース・コミュニケーションを含めることを可能にするいくつかのテクノロジーの 1 つです。SQLJ は、一般に JDBC のような動的な同等機能を上回る、静的 SQL や組み込み型 SQL を生成する手段を提供します。

SCLM Developer Toolkit では、DB2 を使用して SQLJ 対応の Java プログラムをビルドすることを可能にするサンプル・スクリプトも提供されます。

この章を読み終えた後で読者は、SQLJ の本質を理解し、SCLM Developer Toolkit を使用するときその知識をどう適用するかを理解することが目標です。

SQL とは ?

SQL は *Structured Query Language* の頭文字です。SQL はオープン言語であり、Relational Database Management System (RDMS) におけるデータの照会、追加、削除、および変更に使用されます。

この言語が最初に実装されたのは、1970 年代初期の IBM データベース製品である System R でした。それ以降 SQL は成長を続け、(ANSI および ISO により) 標準化され、さまざまなデータベース・システム上でさまざまな特色を発揮しています。

DB2 とは ?

DB2 は人気の高いデータベース・システムで、伝統的にメインフレーム・プラットフォーム向けですが、その後、多くのその他のシステムにも拡大されています。DB2 は z/OS の関係データベース管理システムのデファクト・スタンダードです。

DB2 UDB バージョン 8 は、SCLM Developer Toolkit のビルド・スクリプトが基準とするバージョンです。この章で DB2 に言及している場合は、具体的には DB2 UDB バージョン 8 を指します。

JDBC とは ?

JDBC は *Java Database Connectivity* の略語です。Java 開発では、これはよく知られており、データベースとの相互作用を実装するために一般的に使用される技術です。JDBC はコール・レベルの API で、SQL ステートメントが文字列として API に渡されることを意味し、JDBC が RDMS 上でのそれらの実行を制御します。この結果、これらの文字列の値は実行時に変更され、JDBC を動的に処理します。

JDBC プログラムの実行は、SQLJ 同等機能より低速ですが、この方法の利点には、「一度書けばどこでも呼び出せる (Write Once, call Anywhere)」として知られる概念があります。これは、実行時まで相互作用が必要ないので、JDBC プログラムは非常にポータブルであり、最小の負担で異なるシステム間で利用できることです。

SQLJ とは ?

SQLJ は、Java アプリケーションでデータベース・トランザクションに使用される言語拡張です。静的および組み込み型 SQLJ を生成します。この用語は SQL - *Structured Query Language* および Java を意味する J で構成されます。

ランタイムに実行される SQL ステートメントは、プログラムがアSEMBルされたときに認識されるので、SQLJ は静的 です。JDBC と対比すると、実行するクエリーをいつでも変更できます。

SQLJ は組み込み 型です。バインド時に直列化形式のプログラムの SQL ステートメントがデータベースに提供されるからです。データベースはこの直列化データを使用して、参照する表に対する最適化されたアクセス・パスを決定します。JDBC では、ランタイムにアプリケーションからステートメントを受け取るまでは、データベースはどのステートメントを実行するかを決定する手段がありません。したがってランタイムにアクセス・パスを決定する必要があります。これにより、SQLJ を使用することによって回避されるオーバーヘッドが発生します。

JDBC と SQLJ の比較

この表は、読むことを推奨されている Redbook 「DB2 UDB for z/OS Version 8: *Everything You Ever Wanted to Know, ... and More*」の 5.2 章にある資料に基づいています。

	SQLJ (static)	JDBC (dynamic)
パフォーマンス	一般的には、静的 SQL が動的 SQL より高速です。ランタイムにのみ、プログラムの実行に先行してパッケージやプランの許可がチェックされる必要があるからです。	動的 SQL ステートメントでは、SQL ステートメントが解析され、表やビューの許可がチェックされ、最適化パスが決定される必要があります。
許可	SQLJ では、アプリケーションの所有者はプランやパッケージに関する EXECUTE 権限を付与し、権限を付与された者は書かれているとおりにそのアプリケーションを実行する必要があります。	JDBC では、アプリケーションの所有者は、アプリケーションによって使用される、基礎となるすべての表に関する特権を付与します。これらの特権のを付与された者は、これらの特権で許可されているどんなことでも実行できます。例えば、もともと許可が与えられていたアプリケーションの外部でもこれらの表を使用することができます。アプリケーションでは、ユーザーが実行できることを制御することはできません。

デバッグ	SQLJ は API ではなく、言語拡張です。これは、SQLJ ツールがプログラムの中の SQL ステートメントを認識し、プログラム開発過程で構文や権限の妥当性をチェックすることを意味します。	JDBC は純粋なコール・レベル API です。これは、Java コンパイラーが SQL ステートメントについて何も認識せず、単にメソッド呼び出しに対する引数として出現することを意味します。ステートメントの 1 つにエラーがあると、ランタイムにデータベースがエラーを検知するまで、エラーを認識できません。
モニタリング	SQLJ では、優れたシステム・モニターおよびパフォーマンス報告の機能が利用できます。静的 SQL パッケージでは、任意の時点で実行中のプログラムの名前を知ることができます。この機能は、さまざまなアプリケーションでの CPU 使用量、ロックの問題（デッドロックやタイムアウト）などを調べる場合に特に有効です。	現在実行中のプログラム名を SQLJ 内で識別できる場合は、JDBC を使用して、同じプログラムを介するすべてのトランザクションが発生します。これにより、モニターと問題領域の検出が難しくなります。
冗長性	SQLJ ステートメントは純粋に SQL 構文で作成されており、Java 方式による折り返しの必要がないので、プログラム自体は読みやすく、保守も容易です。また、JDBC で明示的に作成する必要のある一部の定形コードは SQLJ で自動的に生成されるので、SQLJ で書かれたプログラムは、同等の JDBC プログラムより短縮される傾向があります。	JDBC では、すべての SQL ステートメントは API 呼び出しで折り返す必要があるもので、わかりにくく、冗長なコードになりがちです。

直列化プロファイルとは？

SQLJ で書かれたコードは、拡張子が .sqlj のファイルに配置されます。プログラム準備の最初のステップ（詳細は後述）では、.sqlj ファイルは SQLJ 変換プログラムに送られます。

変換プログラムは 2 つのタイプの出力を生成します。うち 1 つは Java ソース・コード (.java) です。このソース・コードは、明らかに .sqlj ファイル内のコードの Java 実装です。

第 2 の出力タイプは直列化プロファイル (.ser) です。このファイルは .sqlj ファイルからの全 SQL ステートメントを直列化形式で含んでいます。このプロファイルはランタイムにプログラムに対して使用可能になる必要があり、RDMS へバインドするために使用されます。

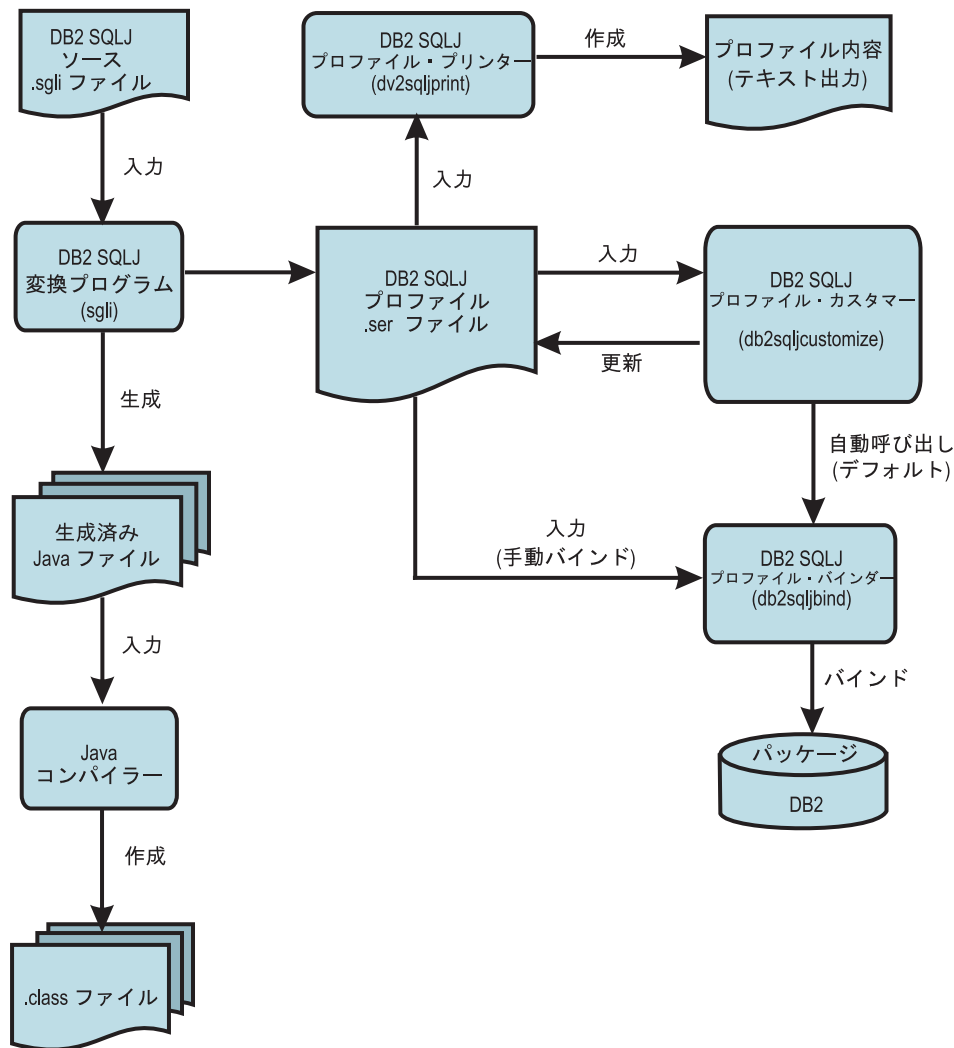
DBRM とは ?

DBRM は *Database Request Module* の略語です。これは、プログラム内での SQL ステートメントの伝統的な DB2 直列化表現です。例えば、プログラムは COBOL で書かれることがあります。このプログラムは、特定の DB2 サブシステムにバインドするのに使用される DBRM を生成するために、DB2 によって事前処理されます。

SQLJ ではプロセスが多少異なり、DB2 UDB バージョン 8 では互換モード という用語で呼ばれます。ユーティリティ *db2sqljcustomize* には、DBRM が生成する原因となるオプションのコマンド行引数が備わっている可能性があります。この DBRM は従来の方法、例えば SCLM ユーザー出口で呼び出される REXX スクリプトを使用して DB2 にバインドされます。

SQLJ プログラム準備

SQLJ プログラムをビルドするために SCLM Developer Toolkit をどのように使用するかを説明する前に、まず手動プロセスを調べます。このプロセスは SQLJ の DB2 実装が目的であり、3 つのコマンド *sqlj*、*db2sqljcustomize* および *db2bind* を使用します。バインド・ステップは、オプションで *db2sqljcustomize* で実行され、*db2bind* が必ず必要ということではありません。



変換

SQLJ 変換プログラム (*SCLM* 言語変換プログラムと混同しないでください) は、SQLJ ソース・ファイルを入力として使用し、Java ソース・コード (.java files) および直列化プロファイル (.ser files) を生成します。

SQLJ 言語自体は本書では説明しません。SQLJ コードの作成について参照する場合は、<http://www.sql.org> を調べてください。

.sqlj ファイルごとに生成される直列化プロファイルの数は、SQLJ コードの中で参照される接続コンテキスト・クラスの数に依存します。各ファイルに対して 1 つの直列化プロファイルが生成されます。

複数の SQLJ ソース・ファイルが単一の接続コンテキスト・クラスを参照するだけなので、単一の直列化プロファイルが生成されます。直列化プロファイルは、ソース・ファイルで参照される順序にしたがって名前が付けられます。名前は次の形式です。

progrname_SJProfileX.ser

この場合:

Progrname	プログラムの名前。これは、入力ソース・ファイル名から拡張子 .sqlj を削除したものです。
X	現行クラスの索引を表す整数です。索引付けはゼロを基準にします。参照される最初の接続コンテキスト・クラスはプロファイル 0 を生成し、次がプロファイル 1 という順序で続きます。

例:

入力: Customer.sqlj (1 つの接続コンテキスト・クラスへの参照)

出力: Customer.java
Customer_SJProfile0.ser

オプションで、引数 -compile=true が sqlj に渡された場合、
Customer.class

カスタマイズ

直列化プロファイルが生成されると、次はそれらのカスタマイズです。DB2 バージョン 8 でこれを行うためのコマンドは *db2sqljcustomize* ですが、前のバージョンでは *db2profcc* でした。カスタマイザーの個々の呼び出しと、SQLJ 変換プログラムの呼び出しは対応している必要があります。つまり、単一の変換プログラム呼び出しで 5 つのプロファイルを生じた場合、これらの 5 つのプロファイルは、単一のプロファイル・カスタマイザーへの呼び出しに対する入力として送付される必要があります。別の方法として、各ユーティリティの 1 つの呼び出しに対して、個々のプログラム名を関連付ける方法があります。プログラム名は、入力のソース・ファイル名から拡張子 .sqlj を削除したものと同じであることに注意してください。

カスタマイズでは、直列化プロファイルにランタイムで使用する DB2 固有の情報を追加します。自動バインドなどのその他のオプションは、コマンド行切り替えを通じて構成されます。DB2 レガシー・バージョンを使用している場合、または *gendbrm* および *dbrmdir* フラグを *db2sqljcustomize* に指定した場合は、DBRM ファイルが生成されます。このファイルは、後でデータベースにバインドする際に使用されます。DB2 UDB 8+ の Universal ドライバーを使用すると、DBRM の生成が不要となり、代わりに、SQLJ 変換プログラムにより生成される直列化プロファイルを使用してバインドします。

バインド

バインドは SQLJ プログラム準備プロセスの最後のステップです。DB2 バージョン 8 でバインドに使用するコマンドは `db2sqljbind` であり、`db2sqljcustomize` を実行するとき、自動的にバインドする方法もあります。バインドは、直列化された SQL ステートメントのために DB2 表へのアクセス・パスを構築するステップです。これらのステートメントは、DBRM または直列化プロファイルのいずれかの形式で使用可能です。

デフォルトでは、4 つのパッケージが作成され、各分離レベルに対して 1 つずつ対応します。バインドには、DBRM を使用する従来の手法と、代わりに直列化プロファイルを使用する新たな Universal 手法があります。

SCLM DT タイプと変換プログラム

SCLM タイプと変換プログラムを説明する前に、*SCLM* 言語変換プログラム、または単に *SCLM* 変換プログラム と、*sqlj* との区別を定義する必要があります。

SCLM では、定義された言語にはすべて、その言語をどう処理するかを理解させる変換プログラムが必要です。これは、SQLJ 変換プログラムとは異なります。*sqlj* はコマンド行ユーティリティであり、SQLJ ソース・ファイルを取り出し、直列化プロファイルと Java ソース・コードを生成するものです。

この区別が明確になったら、SQLJ ビルド・プロセスと関連する *SCLM* タイプ と *SCLM* 変換プログラム について説明します。

SQLJ に対する SCLM 変換プログラムが用意されたら、SCLM に保管されるすべての SQLJ ソース・コードの言語タイプとして割り当てる必要があります。この新規変換プログラムには、別の SCL タイプを定義する必要があります。SQLJ 用の SCLM 変換プログラムは JAVA 変換プログラムに似ていますが、SCLM 出力タイプ SQLJSER および DBRMLIB に対応する追加の IOTYPE 定義を含んでいます。カスタマイズ・ステップの一環として DBRM ファイルの生成が不要な場合、この DBRMLIB IOTYPE は SQLJ 言語定義から削除しても構いません。

プロジェクト定義の中で、管理者は、新規の SCLM 変換プログラムおよび追加タイプを定義し生成する必要があります。

<i>SQLJSER</i>	これは、変換とカスタマイズのステップで作成され、カスタマイズされた、生成済み直列化プロファイルのファイル (.ser ファイル) を保管するために必要です。この SCLM タイプのデータ・セットは、 <code>recfm=VB, lrecl=256</code> として定義することを推奨します。
<i>DBRMLIB</i>	これは、カスタマイズ・ステップで作成した、生成済みの DBRM ファイルを含むタイプです。このタイプは、DB2 バインド・プロセスの一部として生成される DBRM ファイルを使用するお客様にのみ必要です。この SCLM タイプのデータ・セットは、 <code>recfm=FB, lrecl= 80</code> として割り振る必要があります。

ビルド・プロセスの調整

柔軟性を最大に保つため、SQLJ ビルド・プロセスは大幅にカスタマイズ可能であり、さまざまなサイトの構成、および *sqlj* および *db2sqljcustomize* に渡す必要のあるパラメーターの任意の組み合わせに対応します。

この章では、SCLM Developer Toolkit の SQLJ 実装の背景にある概念を説明します。この章を読んだ後、ユーザーがサイトの要件に合わせてビルド・プロセスをカスタマイズできるようになることが目標です。

SQLJ 変換およびプロファイル・カスタマイズの間、SCLM Developer Toolkit はコマンド *sqlj* および *db2sqljcustomize* で使用されるのと同じ Java クラスを直接呼び出します。SCLM DT 変換およびカスタマイズ・プロセスに提供される引数はまったく同一です。各コマンドのすべてのコマンド行引数に関する詳細な説明は、「DB2 Universal Database user guide」を参照してください。

ビルド・スクリプトの調整

「SCLM に追加」ウィザードを使用済みと仮定すると、SQLJ プログラムのビルド・スクリプトには Archdef と同じメンバー名が与えられます。例えば、*sqlj* プロジェクトの Archdef が次の場合は、以下ようになります。

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

ビルド・スクリプトは次の場所で見つけられます。

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

このビルド・スクリプトの中にマスター・ビルド・スクリプトへの参照があります。これはプロパティの中にあります。*Developer Toolkit* に同梱されるビルド・スクリプトは、JAR プロジェクトでは *BWBSQLB*、EJB プロジェクトでは *BWBSQLBE* です。この値を変更する必要はありません。変換およびカスタマイズのステップでリストされるほとんどの構成は、このファイル上にあるままです。

sqlj プロパティ

次の表にリストされている各プロパティ・リストは、*BWBSQLB* ビルド・スクリプトに表示されます。このプロパティは XML 形式であり、次のようになります。

スクリプトを構成するには、関連するすべてのプロパティの値を変更する必要があります。

名前	値	説明
sqlj.exec	/etc/SCLMDT/bwbsqlc.rex	sqlj & db2sqljcustomize exec ルーチン bwbsqlc.rex の場所 を指定します。(サンプル BWBSQLC) デフォルトで、 このサンプルは SCLM DT インストール・ディレクトリ ーに配置されているはずで す。
sqlj.class	sqlj.tools.Sqlj	sqlj クラス名を指定します。 これは、 <i>sqlj</i> ユーティリティー で呼び出されるクラスの 名前です。この値を変更する 必要はほとんどありません。
sqlj.bin	/db2path/bin	sqlj スクリプトが存在する db2 sqlj bin ディレクトリー の場所を指定します。
sqlj.cp	/db2path/jcc/classes/sqlj.zip	クラスパスに含める sqlj.zip の場所を指定します。
sqlj.arg	-compile=false status linemap=NO db2optimize	sqlj プロセスのためのグロー バル・プロパティー引数を下 に指定します。

db2sqljcustomize プロパティー

次の表にリストされている各プロパティー・リストは、BWBSQLB ビルド・スクリプトに表示されます。このプロパティーは XML 形式であり、次のようになります。

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

スクリプトを構成するには、関連するすべてのプロパティーの値を変更する必要があります。

名前	値	説明
sqljdb2cust.class	com.ibm.db2.jcc.sqlj.Customizer	sqlj db2 カスタマイズ のクラス名を指定しま す。この値を変更する 必要はほとんどありま せん。
db2sqljcust.cp	/db2path/jcc/classes/db2jcc.jar: ./SRC: /db2path/jcc/classes/db2jcc_license_cisuz.jar	カスタマイズ・ユーテ ィリティーのクラスパ ス設定。完全修飾パス 名を XML で指定し ます。
db2sqljcust.arg	-automaticbind NO -onlinecheck YES -staticpositioned YES -bindoptions â ISOLATION(CS)â -genDBRM	カスタマイズ・ユーテ ィリティーに指定する 一般的な引数。

db2sqljcust.propfile	user.properties	動的なプロパティー値のスクリプトを決定するためにユーザー・プロパティーに渡される一時的なプロパティー・ファイル名。デフォルトのままに構いません。
db2sqljcust.userpgm	スクリプトをバイパスする場合は NONE。 それ以外は、ユーザー・スクリプトの完全修飾パス名およびファイル名を指定します。	このスクリプトは、カスタマイズ・ユーティリティの直前に実行されます。カスタマイズ・ユーティリティへの入力として使用されるプロパティー・ファイルを動的に更新します。

カスタム・ユーザー・スクリプト

SCLM Developer Toolkit で提供される SQLJ ビルド・スクリプトは、DB2 UDB v8 互換モードで動作するように設計されています。このモードは、直列化プロファイル経由でバインドするだけでなく、DBRM を通じて DB2 の概念をサポートします。直列化プロファイルを使用するために、BWBSQLB への変更が必要です。これはサブトピック『直列化プロファイルのバインドにおける使用』で説明しています。

バインド方法とは別に、ビルド・プロセスをサイトに適合させるために、独自のデータベース環境、分離ポリシー、およびその他の要因に合わせて、*sqlj* および *db2sqljcustomize* に対する引数をカスタマイズする必要性が生じる可能性があります。これらの引数に対応する動的プロパティーを決定するために、独自のスクリプトを作成する必要性も考えられます。例えば、入力ファイル名に関連するわかりやすいパッケージ名を作成する場合です。

SCLM Developer Toolkit は独自のカスタマイズ・スクリプトを指定することによりこれを可能にします。ANT XML ビルド・プロセス上では、すべてが「プロパティー」、つまり名前と値のペアを指定した XML の「プロパティー」エレメントの概念に基づいて動作します。例えば、ビルド・スクリプト BWBSQLB の *db2sqljcustomize* ステップでは、*db2sqljcustomize* に提供されるグローバル・コマンド行引数は *db2sqljcust.arg* という名前と、デフォルト値の *-automaticbind NO -onlinecheck YES -staticpositioned YES -bindoptions "ISOLATION(CS)" genDBRM lang=EN-AU* という設定で、プロパティー・エレメントに定義されます。

提供する引数を変更したい場合は、プロパティーの値を変更するためにビルド・スクリプトを編集すること（これにより設定がグローバルに変更される場合があります）、および自分のカスタマイズ・スクリプトをプロセスにフックすることの両方が可能です。

カスタム・プロパティ・スクリプトをフックするには、スクリプトの名前を `db2sqljcust.userpgm` に、さらに書き込みたいプロパティ・ファイルの名前を `db2sqljcust.propfile` に設定します。

`db2sqljcust.userpgm` に指定したスクリプトは、`db2sqljcustomize` プロセスの直前に実行されます。スクリプトは、`db2sqljcust.userpgm` に指定したプロパティ・ファイルを動的に更新します。ビルド・プロセスにより、動的に更新されたプロパティ・ファイルのプロパティと、ビルド・スクリプトに既に定義されているプロパティの両方が連結された状態で、このプロパティ・ファイルは `db2sqljcustomize` プロセスへの入力として使用されます。

`db2sqljcust.userpgm` に指定したスクリプトは、それを実行するとき次の引数が提供されます。

引数	説明
Basedir	ベース・ディレクトリー (ワークスペース・ディレクトリー)
Propfile	作成・更新するプロパティ・ファイルの名前。注意: 作成するプロパティ・ファイルは、 <code>basedir'/propfile</code> である必要があります。
Sqljf	ファイル名のリスト。 <code>db2sqljcustomize</code> で処理される直列化プロファイル (.ser) を表します。

プロパティは、行あたり 1 つのプロパティ宣言を含み、次の形式でファイルに設定する必要があります。

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

例えば、次のようにします。

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

カスタム・ルーチンはファイルごとに一度、呼び出されます。最後に、引数プロパティを使用して `db2sqljcustomize` 呼び出しのために必要な引数の文字列を構築します。例えば、次のようにします。

```
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}
```

バインド [DBRM]

従来の DB2 は、この目的で データベース要求モジュール または DBRM を使用します。フラグ `gendbrm` が設定されている場合、DBRM は、`db2sqljcustomize` コマ

ンドによって生成されます。このフラグが設定されていない場合、このコマンドは、直列化プロファイル経由でバインドするものと仮定して DBRM を生成しません。

このパラメーターを設定すると、SCLM Developer Toolkit は生成済みの DBRM をピックアップして、将来使用するために SCLM に保管します。この技法を使用する 1 つの利点は、ビルド/コピー出口などの SCLM ユーザー出口で DB2 バインドを容易に実行できることです。

ビルド/コピー・ユーザー出口では、更新済みオブジェクト・リストが自動的に提供されるので、変更されたモジュールのみを選択して再バインド可能であり、不要なバインドによる非効率を回避できます。

DBRM のバインドを構成するには 4 つのステップがあります。

1. *sqlj* の適切な引数を設定します。

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

引数	説明
compile=false	このオプションを「false」に設定すると、sqlj 変換プログラムは、生成する Java ソースを自動的にコンパイルすることを回避します。SCLM Developer Toolkit は、生成されたソースをビルド・プロセスで使用するの、このオプションを常に「false」に設定することが推奨されます。
linemap=no	Java 例外の行番号を、SQLJ ソース・ファイル (.sqlj ファイル) の行番号と一致させるかどうか、または SQLJ 変換プログラムのファイル (.java ファイル) によって生成される Java ソース・ファイルと一致させるかどうかを指定します。これは .class ファイルを必要とします。したがって <i>compile=false</i> と一緒に使用する場合は、 <i>no</i> に設定する必要があります。
status	SQLJ プロセスの直接のステータス表示を印刷します。

2. *gendbrm* を含め、*db2sqljcustomize* の適切な引数を設定します。

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

引数	説明
automaticbind no	「no」に設定すると、カスタマイザーはカスタマイズの終了時点でバインドを実行しません。
onlinecheck yes	<i>url</i> パラメーターで指定したシステム上でオンライン検査を実行します。 <i>url</i> が提供されている場合は、デフォルトで「yes」。そうでない場合は「no」です。
Bindoptions ISOLATION(CS)	バインダーに単一のパッケージの作成を指示します(カーソル固定)。 <i>singlepkgname</i> と連結して使用されます (動的に設定)。
gendbrm	カスタマイザーに DBRM ファイルの生成を指示します。

3. ユーザー・スクリプトを構成します。

ユーザー・プログラムの場所を **BWBSQLB** に設定します。これは、動的プロパティを計算するために使用する **rex** スクリプトがどこにあるかをビルド・プロセスに知らせます。

動的に構成したい大きなプロパティは、*singlepkgname* です。これはバインド先のパッケージの名前であり、各プログラムはそれぞれ固有の名前を付けようとしています。この単純な例では、プログラム名の最初の 8 文字を使用します。

4. DBRM をバインドするためのビルド出口を作成します。ビルド・コピー出口が推奨されます。

カスタマイズ・ステップで *singlepkgname* を使用しているので、パッケージの名前は **DBRM** の名前と同じになります。

バインド [直列化プロファイル]

SQLJ プログラムをバインドするために、新たに推奨する手法はバインドに直列化プロファイル (.ser ファイル) を使用することです。直列化プロファイルは **DBRM** と同じ機能 (プログラムの中で直列化イメージのステートメントを提供する) を実行するので、このアプローチは不可避です。

ビルド・スクリプト **BWBSQLB** にわずかな変更を加えて、代わりにこの手法を使用するよう **SCLM Developer Toolkit** を構成できます。変更の方法は簡単で、*db2sqljcustomize* に設定する引数を *gendbrm* コマンド行の切り替えを削除するように変更し、*automaticbind* を「YES」に変更します。

直列化プロファイルのバインドを構成するには、3 つのステップがあります。

1. *sqlj* に適切な引数を設定します。

sqlj 変換プログラムには、直列化プロファイルのバインドに固有なコマンド行引数はありません。ただし、この特別な例の引数の設定は次のとおりです。

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

引数	説明
compile=false	このオプションを「false」に設定すると、 <i>sqlj</i> 変換プログラムは、生成する Java ソースを自動的にコンパイルすることを回避します。 SCLM Developer Toolkit は、生成されたソースをビルド・プロセスで使用するので、このオプションを常に「false」に設定することが推奨されます。
linemap=no	Java 例外の行番号を、 SQLJ ソース・ファイル (.sqlj ファイル) の行番号と一致させるかどうか、または SQLJ 変換プログラムのファイル (.java ファイル) によって生成される Java ソース・ファイルと一致させるかどうかを指定します。これは .class ファイルを必要とします。したがって <i>compile=false</i> と一緒に使用する場合は、 <i>no</i> に設定する必要があります。
status	SQLJ プロセスの直接のステータス表示を印刷します。

2. `db2sqljcustomize` に適切な引数を設定します。

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

引数	説明
automaticbind yes	「yes」に設定すると、カスタマイザーはカスタマイズの終了時点でバインドを実行します。「no」に設定すると、コマンド <code>db2bind</code> とは別にバインドを実行する必要があります。
onlinecheck yes	<code>url</code> パラメーターで指定したシステム上でオンライン検査を実行します。 <code>url</code> が提供されている場合は、デフォルトで「yes」。そうでない場合は「no」です。

3. ユーザー・スクリプトを構成します。

ユーザー・プログラムの場所を `BWBSQLB` に設定します。これは、動的プロパティを計算するために使用する `rex` スクリプトがどこにあるかを、ビルド・プロセスに知らせます。

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

付録 C. ロング/ショート・ネーム変換テーブル

現在のところ、コア SCLM では、8 文字を超えるファイル (メンバー) 名のコード・ストレージを使用できません。

Java や他の PC クライアント・コードのコードは、元々かなり長い名前をもち、パス情報を名前の一部として統合 (パッケージ化) する場合さえあります。このため、8 文字を超える名前付き部分をもつコードは、ロング/ショート・ネーム変換ユーティリティを通して、これらの部分を名前の長さが 8 文字以下の SCLM に保管できるようにする必要があります。

ロング・ネームからショート・ネームへの変換テーブルには、ショート・ネーム (SCLM に保管) に対応するロング・ネーム (実名) が保管されます。これらのテーブルは SCLM によって制御およびアクセスされ、VSAM データ・セットに保存されます。この機能は、z/OS V1.4 以降用の APAR OA11426 を扱う PTF 付きで SCLM に導入されています。z/OS V1.8 以降の場合、この PTF は必要ありません。

変換アルゴリズムは、以下の手順に従っています。

1. 変換接頭部は、ロング・プログラム/ファイル名 (すなわち、マルチ・パッケージング形式で / 文字の後の最後のファイル名) の最初の 2 文字 (大文字) からなります。最初の 2 文字がホスト・メンバー名の接頭部として無効な場合 (無効な特殊文字を含んでいるため) 接頭部は XX となります。単一の英字による名前 (/u/test/A や /u/test/A.java) など特殊なケースにも XX の接頭部が割り当てられます。
2. 最後の 6 文字は、数值的に、変換テーブルで使用可能な次位の連番を割り当てられます。

例:

ロング・ネーム	SCLM PDS または PDSE でのショート・ネーム
com/ibm/workbench/testprogram.java	TE000001
source/plugins/Phantom/.classpath	XX000001

SCLM 変換プログラムの技術要約

SCLM プログラム FLMLSTRN は、VSAM 変換テーブルの読み取りおよび更新のために作成されました。SCLM Developer Toolkit は、このプログラムを使用して相関関係のあるロング・ネームとショート・ネームを読み取り、更新します。

変換テーブルを保管するために使用される VSAM ファイルは、可変長 KSDS で、さらに代替の索引およびパスが定義されています。SCLM では、この VSAM ファイルを割り振るためにサンプル・ジョブが提供されています。

VSAM クラスターの内部構造は次のとおりです。

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

ロング/ショート・ネーム変換用の VSAM ファイルを割り振るためのサンプル JCL は、19 ページの『ステップ 6: ロング/ショート・ネーム・テーブル VSAM ファイルの構成』にあります。

注: SCLM 変換テーブルの機能呼び出しに関する以下の技術情報は、参照用にのみ提供されるだけで、SCLM Developer Toolkit の機能を有効にする上では必要ありません。

プログラム FLMLSTRN は、表 11 にリストされたパラメーターの 1 つを使用して ISPF SELECT サービスで呼び出されます。

構文

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

呼び出し例:

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

表 11. ロング/ショート・ネーム変換パラメーター

キーワード・レコード	処理	説明
FINDLONG	単一	一定のショート・ネームに対応するロング・ネームを検索します。
FINDSHORT	単一	一定のロング・ネームに対応するショート・ネームを検索します。
TRANSLATE	単一	ショート・ネームが存在すればそれを検索し、存在しない場合は新規ショート・ネームを割り振ります。
MIGRATE	複数	複数のロング・ネームを検索します。
IMPORT	複数	複数のショート・ネームを検索します。

単一ロング/ショート・ネーム・レコード処理

FINDLONG 処理

- DD LSTRANS に割り振られた VSAM クラスターが読み取りモードで開かれます。
- ショート・ネームが ISPF 変数 FLMLSSHR から取り出されて、VSAM ファイルを読み取るために使用されます。
- レコードが見つからない場合、ロング・ネームが見つからなかったことを示すメッセージが ISPF 変数 FLMLSERR 経由で戻されます。
- ロング・ネームが見つかった場合、ISPF 変数 FLMLSLNG で戻されます。
- VSAM クラスターが閉じられます。

FINDSHORT 処理

- DD LSTRNPTH に割り振られた VSAM パスが読み取りモードで開かれます。
- ロング・ネームが ISPF 変数 FLMLSLNG から取り出されます。
- ロング・ネームの最後の 50 バイトがパスを読み取るために使用されます。
- レコードが戻されない場合、ショート・ネームが見つからなかったことを示すメッセージが ISPF 変数 FLMLSERR 経由で戻されます。
- レコードが戻された場合、VSAM レコードのロング・ネームが ISPF 変数 FLMLSLNG 内のロング・ネームと比較されます。
- 一致しない場合、ls_lngname_key が一致しなくなるか、ロング・ネームが見つかるまで、VSAM レコードの読み取りと比較が続けられます。

注: ls_lngname_key では、重複が許されます。同じ ls_lngname_key で異なるロング・ネームをもつ VSAM レコードが存在できるためです。

- ショート・ネームが見つかった場合、ISPF 変数 FLMLSSHR で戻されます。
- VSAM パスが閉じられます。

TRANSLATE 処理

処理は、FINDSHORT の場合と同じです。

- ショート・ネームが見つかる、それ以上の処理は行われません。
- ショート・ネームが見つからなかった場合、DD LSTRANS に割り振られた VSAM クラスターが更新モードで開かれます。
- ファイル名は、ロング・ネーム中の最後の「/」または「\」を見つけることで判別されます。
- ファイル名の最初の 2 バイトが、番号を含む VSAM ファイル接頭部レコードを検索するために使用されます。
- ファイル接頭部と番号でショート・ネームが生成されます (例えば、PR000123)。
- 生成されたショート・ネーム (PR000123) は、VSAM ファイルを検査して、そのショート・ネームが使用中であるかどうかを判別するために使用されます。
- 使用中の場合、接頭部番号が増やされ、ショート・ネームがもう一度検査されます。
- この処理は、使用中でないショート・ネームが見つかるまで続けられます。
- 接頭部レコードが更新された後、新規の変換レコードが追加されます。
- ショート・ネームが ISPF 変数 FLMLSSHR で戻されます。
- VSAM クラスターが閉じられます。

複数ロング/ショート・ネーム・レコード処理

MIGRATE および IMPORT は、多数のロング・ネームを変換する場合 (MIGRATE) または多数のショート・ネームを検索する場合 (IMPORT) にパフォーマンスを改善するために導入された機能です。

両方の機能 (MIGRATE と IMPORT) とともに、LRECL=1036 (DD LSTRNPRC として割り振り) を指定された可変のブロック化順次ファイルを読み取ります。

複数ロング/ショート・ネーム・レコード処理

呼び出し前、このファイルには、呼び出される機能によって、ショート・ネームまたはロング・ネームが正しい形式で、正しい列に入ります。

呼び出し後、LSTRNPRC にはショート・ネームと関連するロング・ネームの両方が入ります。

ファイルの形式は次のとおりです。

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

IMPORT 処理

- DD LSTRANS に割り振られた VSAM クラスターが読み取りモードで開かれ、DD LSTRNPRC に割り振られた処理ファイルが更新用に開かれます。
- 処理ファイルのそれぞれのレコードでは、ショート・ネームが VSAM 変換ファイルを読み取るために使用されます。レコードが見つかり、処理ファイルがロング・ネームで更新されます。
- VSAM クラスター/処理ファイルが閉じられます。

MIGRATE 処理

- DD LSTRANS に割り振られた VSAM クラスターが読み取りモードで開かれ、DD LSTRNPRC に割り振られた処理ファイルが更新用に開かれます。
- 処理ファイルのそれぞれのレコードでは、ロング・ネームが VSAM ファイルを読み取るために使用されます。レコードが見つかり、処理ファイルのレコードが対応するショート・ネームで更新され、見つからない場合、LSTRANS が更新モードで開かれ、新規のロング/ショート名前項目が追加され、新規に生成されたショート・ネームが LSTRNPRC ファイルに戻されます。
- VSAM クラスター/処理ファイルが閉じられます。

次に、ロング/ショート・ネーム変換処理を呼び出すためのサンプルの REXX を示します。

```

/* REXX *****/
/* Sample to translate long name to a short name */
/*****/
Address TSO
"FREE FI(LSTRANS)"
"FREE FI(LSTRNPTH)"
"ALLOC DD(LSTRANS) DA('BWB.LSTRANS.FILE') SHR REUSE"
"ALLOC DD(LSTRNPTH) DA('BWB.LSTRANS.FILE.PATH') SHR REUSE"
/* Create short name for long name com/ibm/phantom.txt */
FLMLSLNG = "com/ibm/phantom.txt"
Address ISPEXEC "VPUT (FLMLSLNG) PROFILE"
Address ISPEXEC "SELECT PGM(FLMLSTRN) PARM(TRANSLATE)"
LSRC=RC
If LSRC > 0 Then
Do
    Address ISPEXEC "VGET (FLMLSERR,FLMLSER1) PROFILE"
    Say "LS ERROR LINE 1 ==>" FLMLSERR
    Say "LS ERROR LINE 2 ==>" FLMLSER1
    Return
End
Else
Do
    Address ISPEXEC "VGET (FLMLSSHR,FLMLSLNG) PROFILE"
    Say " Shortname = " FLMLSSHR
    Say " Longname = " FLMLSLNG
End
Address TSO
"FREE FI(LSTRANS)"
"FREE FI(LSTRNPTH)"

```

図 35. 変換モジュール呼び出し用のサンプル REXX

参考文献

本書で参照している資料:

Domino Go Webserver for OS/390 概説およびインストール, SD88-7879

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

対話式システム生産性向上機能 (ISPF) SCLM
プロジェクト管理および開発者のガイド *z/OS*,
SC88-8960

対話式システム生産性向上機能 (ISPF) ソフト
ウェア構成およびライブラリー管理機能
(SCLM) リファレンス *z/OS*, SC88-8961

z/OS 情報ロードマップ, SA88-8518

*z/OS UNIX システム・サービス コマンド解説
書*, SA88-8641

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

z/OS UNIX システム・サービス 計画,
GA88-8639

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-8711
東京都港区六本木 3-2-12
法務・知的財産
知的財産権ライセンス渉外

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

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

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

以下の保証は、国または地域の法律に沿わない場合は、適用されません。IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態を提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

商標

以下は、International Business Machines Corporation の米国およびその他の国における商標です。

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

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

Windows は Microsoft Corporation の米国およびその他の国における商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

用語集

[ア行]

アーキテクチャー定義 (architecture definition). アプリケーションのコンポーネントを概念単位に編成する手段。アプリケーションの構成を定義する SCLM の方法。アプリケーションのコンポーネントの組み合わせ方法、およびビルド機能とプロモート機能の両方を実行するためのコンポーネントの使用方法を記述する。アーキテクチャー定義は、コンポーネントをアプリケーション、サブアプリケーション、およびロード・モジュールにグループ化するために使用される。

アーキテクチャー・メンバー (architecture member). 個々のソフトウェア・コンポーネント (他のアーキテクチャー・メンバーの集合であることもある) を、アプリケーションの他のソフトウェア・コンポーネントとの関係を指定することで定義する。

アカウンティング・レコード (accounting record). SCLM 制御データ・レコード。SCLM の制御下にあるメンバーの統計情報、ヒストリカル情報、および従属情報を含む。

アンロック (unlock). 以前ロックアウトされたメンバーを更新できるようにすること (通常はプロモートと関連付けられる)。

アンロック・サービス (unlock service). メンバーの開発グループに対する制約を除去する (アンロックする)。

エンタープライズ・アーカイブ (Enterprise Archive). 特殊化されたタイプの JAR ファイルで、J2EE 標準によって定義され、J2EE アプリケーションを J2EE アプリケーション・サーバーにデプロイするために使用される。EAR ファイルには、個々の Web アプリケーションの EJB コンポーネント、デプロイメント記述子、および Web アーカイブ (WAR) ファイルが含まれる。

[カ行]

階層 (hierarchy). グループをランク順に編成したもので、各グループはその上のグループに従属する。

開発グループ (development group). 階層の最低レベルにあるグループ。このグループは、このグループにプロモートする下位グループを持たないエンド・ノードを表す。

許可コード (authcode). SCLM が使用する識別子で、階層内のメンバーを更新しプロモートする権限を制御するもの。このコードを使用すれば、モジュール衝突 (変更点のオーバーレイ) のリスクを冒さずに並行開発が可能である。

許可コード (authorization code). 「許可コード (authcode)」を参照。

グループ (group). SCLM 論理命名規則において同じ上位修飾子を持つプロジェクト・データ・セットの集合。

言語定義 (language definition). SCLM 機能 PARSE、VERIFY、BUILD、COPY、および PURGE のために実行される変換プログラムの集合を指定する。言語定義は、言語属性が FLMLANGL マクロの LANG キーワードの値と一致する SCLM ライブラリーのメンバーに対して実行される各変換プログラムについて、1 つの FLMLANGL マクロとその後に続く FLMTRNSL マクロとから構成される。

コンテナ (container). J2EE において、コンポーネントにライフ・サイクル管理、セキュリティ、デプロイメント、およびランタイム・サービスを提供するエンティティ。各タイプのコンテナ (EJB、Web、JSP、サーブレット、アプレット、およびアプリケーション・クライアント) もコンポーネント固有のサービスを提供する。

コンポーネント (component). アプリケーションに関連する入力または出力メンバーで、アプリケーションの全メンバーまたは 1 メンバーを構成するもの。

[サ行]

サービス (service). コマンドやプログラミング・インターフェースを介して使用できる SCLM 機能。

スコープ (scope). ビルドまたはプロモートによって処理される (例えば、検証、コピー、コンパイル、またはパージされる) メンバー (アーキテクチャー定義を含む) の集合。

双方向 (bi-di) (bidirectional (bi-di)). 数字 (左から右に書かれる) を除き、一般に右から左に書かれるアラビア語やヘブライ語などの添え字に関する用語。

双方向属性 (bidirectional attribute). テキスト・タイプ、テキスト方向、数値スワッピング、および対称スワッピング。

【タ行】

代替プロジェクト定義 (alternate project definition). デフォルト・プロジェクト定義とは異なるバージョンのプロジェクト環境を提供するプロジェクト定義。

タイプ (type). プロジェクト区分データ・セットの SCLM 命名規則の 3 番目の修飾子。通常は、プロジェクト階層として維持管理されるデータの種別を識別する。タイプの例として、SOURCE、OBJECT、および LOAD がある。

【ハ行】

バージョン (version). メンバーが前の時点で存在していたときのコピー。

バージョン管理 (versioning). メンバーのバージョンの取得を可能にする機能。これは、変更を取り消すのに役立つ。

パースペクティブ (perspective). ワークベンチ内のリソースのさまざまな側面を表示するビューのグループ。ワークベンチ・ユーザーは、手元のタスクに応じてパースペクティブを切り替え、パースペクティブ内のビューおよびエディターのレイアウトをカスタマイズできる。

ビルド (build). 言語定義で指定された変換プログラムの呼び出しを通じて入力を入力に変換すること。ビルド時に呼び出される変換プログラムの例として、コンパイラー、プリプロセッサ、およびリンケージ・エディターがある。

ビルド・マップ (build map). ビルド時にデータベースの完全分析を含む内部データ・レコード。参照されたすべてのメンバーの名前と、各メンバーの最終変更日およびバージョン番号を含む。

プロジェクト (project). 単一の高位修飾子のもとに統合された SCLM データベースを表すライブラリーの集合。

プロジェクト管理担当者 (project administrator). SCLM プロジェクトを維持管理する担当者。

プロジェクト定義 (project definition). SCLM ライブラリー構造、プロジェクト制御情報、および言語定義を定義する。プロジェクト定義は、SCLM が実行時に使用するロード・モジュールである。プロジェクト定義のソース・コードはマクロで構成されている。

プロジェクト定義データ (project definition data). SCLM プロジェクトの作成と制御に使用されるプロジェクト定義と言語定義。

プロモート (promote). アプリケーションまたはそのコンポーネントをプロジェクト階層内のあるレベルから次のレベルに移動すること。開発グループからプロモーションすると、正常にプロモートされた編集可能メンバーのロックが除去される。

変換プログラム (translator). 実行のために SCLM から制御権を受け取るロード・モジュール、CLIST、または REXX プログラム。変換プログラムの名前は、FLMTRNSL マクロの COMPILE キーワードの値として指定される。変換プログラムの例として、コンパイラー、アセンブラー、リンケージ・エディター、テキスト・プロセッサ、DB2® プリプロセッサ、CICS® プリプロセッサ、ユーティリティ、およびカスタマー・ツールがある。

変更コード (change code). 8 文字の識別子で、SCLM が制御するメンバーの更新や変更の理由を示すために使用される。

【マ行】

マイグレーション (migrate). ソフトウェア・コンポーネントを SCLM に登録すること。これには、コンポーネント言語の識別と、場合により変更コードおよび許可コードが含まれる。

マイグレーション (migration). メンバーを SCLM の制御下に置くこと。マイグレーションでは、メンバーをロックし、要求された言語に従ってメンバーを解析し、情報をアカウンティング・データベースに保管する。マイグレーション・ユーティリティを使用して、SCLM への変換時などに多数のメンバーをプロジェクトのデータベースに入力できる。

メンバー (member). SCLM データベースの個々のエレメントで、単一のデータ型のソフトウェア・コンポーネントを表す。

【ラ行】

ライブラリー (library). z/OS における区分データ・セット。

レベル (level). 階層の特定の層で、グループから構成され、同等のランクを持つもの。

ロック (lock). ユーザーがメンバーをロックすると、そのメンバーを変更できるのはそのユーザーのみとなる。その他すべてのユーザーは、そのメンバーがプロモート

またはアンロックされるまで、そのメンバーを変更できない。メンバーがロックされると、許可コードが指定される。2 人のユーザーが同じパーツを変更する必要がある場合は、2 つの異なる許可コードを使用できる。

E

EAR. 「エンタープライズ・アーカイブ (Enterprise Archive)」を参照。

EJB. 「Enterprise JavaBeans」を参照。

Enterprise JavaBeans. JavaBeans は、サブルーチンのように再利用可能なオブジェクトである。EJB はこれをさらに進めたもので、プラットフォームに依存しないように設計されている。

I

IDE プロジェクト (IDE project). Eclipse IDE で開発されたプロジェクト。

J

J2EE. Java 2 プラットフォームの Enterprise Edition。コンポーネント・ベースの多層エンタープライズ・アプリケーションを開発するための標準を定義する。

J2EE プロジェクト (J2EE project). Java/J2EE に特に関係のある Eclipse IDE で開発されたプロジェクト。

JAR. Java アーカイブ (Java Archive)。

Java エンタープライズ (Java Enterprise). Java プログラム向けまたは Java プログラムに関する情報を格納するためのファイル・フォーマット。

S

SCLM 管理者 (SCLM administrator). SCLM プロジェクトを維持管理する担当者。

W

WAR. 「Web アーカイブ (Web Archive)」を参照。

Web アーカイブ (Web Archive). Web タイプのアプリケーション向けまたは Web タイプのアプリケーションに関する情報を格納するためのファイル・フォーマット。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

依存関係

クラスパス 41

インストール

概要 xiii

確認 23, 28

Ant 21

インストール・ディレクトリー

構成ファイル用に推奨 6

受け渡しディレクティブ、httpd.conf ファイル内 12

オーバーライド、TRANSLATE.conf 68

オプション

構成可能 62

サンプルの SITE 固有のファイル 63

サンプルのプロジェクト固有のファイル 65

定義 65

プロジェクト 62

ASCII のストレージ 59

BATCHBUILDn 66

BATCHMIGRATEn 66

BATCHPROMOTEn オプション 66

BUILDAPPROVER 65

BUILDSECURITY 66

CCODE 66

EBCDIC のストレージ 59

FOREGROUNDBUILD 66

FOREGROUNDPROMOTE 66

PROMOTESECURITY 66

SITE 62

[カ行]

階層ファイル・システム

参照: z/OS UNIX システム・サービス・ファイル・システム

概念

言語 84

タイプ 83

ファイル命名 83

プロジェクト構造 84

プロパティー 84

ARCHDEF 85

鍵認証 59

カスタマイズ

確認 23, 28

Ant 21

ISPF 構成ファイル 7

ISPF.conf 7

キーワード

LOGLANG 8, 9

TRANLANG 9

規則

SAF/RACF

デプロイ 73

ビルド 73

プロファイル 73

プロモート 73

規則形式

デプロイ 73

ビルド 72

プロモート 72

機能

IMPORT 105

MIGRATE 105

クラスパスの依存関係 41

グローバル変数

ANT_BIN 61

CGI_DTWORk 62

CLASSPATH_JARS 62

DEBUG_MODE 62

JAVA_BIN 62

TRANTABLE 62

言語 84

言語変換プログラム 60

BINARY 60

BWBTRAN1 37

BWBTRAN2 37

BWBTRAN3 37

BWBTRANJ 37

J2EEPART 60

JAVA/J2EE サポート 37

SCLM、J2EEOBJ 42

SCLM、J2EEPART 40

TEXT 60

コード・ページ

参照: ASCII/EBCDIC 変換

公開鍵認証 59

構成に関する考慮事項 5

構成ファイル

カスタマイズ 7

プロジェクト 65

プロモート 77

HTTP サーバー 12

構成ファイル (続き)

TRANSLATE 10

[サ行]

サンプル

BWBGLOB 61

BWBSITE 63

CRON 開始のビルドおよびプロモート 77

サンプル・メンバー

BWBCRON1 78

BWBCRONB 78

BWBCRONP 78

ショート・ネーム

参照: ロング/ショート・ネーム

処理

FINDLONG 104

FINDSHORT 105

IMPORT 106

MIGRATE 106

TRANSLATE 105

推奨される J2EE プロジェクト (およびその他のプロジェクト) の SCLM へのマッピング 54

スケルトン、ビルド 52

ストレージ・オプション

参照: オプション

セキュリティ規則

代理ユーザー ID 72

セキュリティ・フラグ 71

セットアップ JCL、実行 5

ソフトウェア要件 4

[タ行]

タイプ 83

代理ユーザー ID

セキュリティ規則 72

ディレクティブ

Exec

httpd.conf ファイル内 12

PASS

接続に失敗した場合の確認 25

Pass

httpd.conf ファイル内 12

ディレクトリー 63

CONFIG 5

LOGS 5

PROJECT 6

WORKAREA 5

デプロイ規則形式 73
デプロイメント
オプション 59
セキュア 58
SCLM Developer Toolkit 56
SCLM から Unix システム・サービス
へ 58
WebSphere Application Server
(WAS) 57
デプロイメント・オプション 59

[ナ行]

認証
公開鍵 59

[ハ行]

ビルド
構成ファイル 77
CRON 開始 77
ビルド規則形式 72
ビルド/プロモート/デプロイ
セキュリティ・フラグ 71
プロセス・フロー 71
ビルド・スケルトン 52
ファイル、CRONTAB 79
ファイル命名 83
プロジェクト構成ファイル、サンプル 65
プロジェクト構造 84
プロセス・フロー
ビルド/プロモート/デプロイ 71
プロパティ 84
プロモート
構成ファイル 77
CRON 開始 77
プロモート規則形式 72
変換
言語変換プログラム 60
使用中のコード・ページ 8
パーツの変換 10
必須ソースまたはコンポーネント 40
標準外コード・ページ変換 13
ホストへの変換 9
ロング/ショート・ファイル名 103
Ant テキスト・ファイルおよびスクリ
プトの変換 22
ASCII または EBCDIC で格納された
ファイル 59
参照: ASCII/EBCDIC 変換
ポート
参照: ポート番号
ポート番号
デフォルトの HTTP サーバー 11
変更 11

ポート番号 (続き)
ロケーションの URL で使用される
24
HTTP サーバーへの接続の検査に使用
される 27
httpd.conf ファイル内 12

[マ行]

命名、ファイル 83
メンバー
BWBHTTPC 6
BWBPROJ
サンプルのプロジェクト構成ファイ
ルの保持 65

[ヤ行]

要件
RACF 5
STEPLIB 78
要件、PATH 78

[ラ行]

ライセンスのお問い合わせ 111
リソース・アクセス制御機能 (RACF)
参照: RACF
リモート・システム・エクスプローラー、
構成 16
ロング/ショート・ネーム
単一レコード処理 104
テーブル VSAM ファイルの構成 19
複数レコード処理 105
変換テーブル 103
変換パラメーター 104
変換ユーティリティ 103
ロング/ショート・ネーム変換
REXX サンプル 106
ロング/ショート・ファイル名
参照: ロング/ショート・ネーム
ロング・ネーム
参照: ロング/ショート・ネーム

A

Ant
インストール 21
カスタマイズ 21
初期化のテスト 23
ASCII から EBCDIC への変換 22
Web アドレス 21

ANT_BIN グローバル変数
z/OS UNIX システム・サービスのファ
イル・システム・ディレクトリー・
パス 61
ARCHDEF 85
ARCHDEF SCLM タイプ 43
ASCII
参照: ASCII/EBCDIC 変換
ASCII コード・ページ 67
ASCII のストレージ・オプション 59
ASCII/EBCDIC 変換
言語変換プログラム 60
使用中のコード・ページ 8
パーツの変換 10
必須ソースまたはコンポーネント 40
標準外 8
標準外コード・ページ変換 13
ホストへの変換 9
Ant テキスト・ファイルおよびスクリ
プトの変換 22
ASCII または EBCDIC で格納された
ファイル 59
参照: ASCII/EBCDIC 変換
参照: IBM-1047
参照: ISO8859-1

B

BATCHBUILDn オプション 66
BATCHMIGRATEn オプション 66
BATCHPROMOTEn オプション 66
BIDIPROP オーバーライド 69
BIDIPROP 言語 68
BINARY 言語変換プログラム 60
BUILDAPPROVER オプション 65
BUILDSECURITY オプション 66
BWBCPANT
Ant インストール・メンバー 22
BWBCRON1 サンプル・メンバー 78
BWBCRONB サンプル・メンバー 78
BWBCRONP サンプル・メンバー 78
BWBGLOB サンプル 61
BWBHTTPC メンバー 6
BWBHTTPE メンバー 6
BWBINST1 インストール JCL
実行されるタスク 5
BWBINST1 サンプル構成ファイル 12
BWBPROJ メンバー
サンプルのプロジェクト構成ファイ
ルの保持 65
BWBSITE サンプル 63
BWBTRAN1 言語変換プログラム 37
BWBTRAN2 言語変換プログラム 37
BWBTRAN3 言語変換プログラム 37
BWBTRANJ 言語変換プログラム 37

BWBTRANT

サンプル変換スクリプト 22

C

CCODE オプション 66

CGI_DTWORk グローバル変数

インストール・ホーム・ディレクトリ
ー 62

CLASSPATH_JARS グローバル変数

z/OS UNIX システム・サービスのファ
イル・システムのクラスパス・ディ
レクトリ 62

CODEPAGE キーワード 8

CONFIG ディレクトリ、作成 5

CONFIG/PROJECT ディレクトリ

参照: PROJECT ディレクトリ

CRON

ビルド・ジョブのサンプル 79

ビルド・ジョブの実行 79

CRON 開始のビルド 77

CRON 開始のプロモート 77

CRONTAB ファイル 79

D

DEBUG_MODE グローバル変数

VSAM ファイル 62

DEPLOYSECURITY オプション 67

E

EBCDIC

参照: ASCII/EBCDIC 変換

EBCDIC コード・ページ 67

EBCDIC のストレージ・オプション 59

Eclipse ペースのクライアント

インストール 31

前提条件 31

Exec ディレクティブ、httpd.conf ファイ

ル内 12

F

FINDLONG 処理 104

FINDSHORT 処理 105

FLMLSTRN、SCLM 変換プログラム 103

FOREGROUNDBUILD オプション 66

FOREGROUNDPROMOTE オプション

66

H

HTTP サーバー

環境ファイル

カスタマイズ 13

ロケーション 12

既存のサーバーのカスタマイズ 14

構成 11

サンプル構成ファイル 6, 12

始動 15

接続に失敗した場合の確認 25

接続のテスト 27

デフォルトのポート構成 11

トレース 15

ポート 12

ユーザー ID を所有する 14

ログオン・プロンプト 24

JCL/STARTED TASK のカスタマイズ

13

HTTP のインストール 23

httpd.conf

カスタマイズ 12

サンプル HTTP 構成ファイル 12

CONFIG ディレクトリへのコピー

6

httpd.env

カスタマイズ 13

CONFIG ディレクトリへのコピー

6

I

IBM z/OS HTTP サーバー

参照: HTTP サーバー

IBM-1047

デフォルトの EBCDIC コード・ペー
ジ 8

IMPORT 機能

ロング/ショート・ネーム・レコードの
処理の改善 105

IMPORT 処理 106

ISO8859-1

デフォルトの ASCII コード・ページ
8

ISPF 構成ファイル

カスタマイズ 7

ISPF.conf

カスタマイズ 7

ロケーション 7

IVP

インストールとカスタマイズを確認す
るための実行 23, 28

J

J2EE

サンプル言語変換プログラム 37

サンプル・スクリプト 50

参照: JAVA/J2EE

J2EE ARCHDEF

サンプル 45

J2EE プロジェクト、マッピング 53

J2EE プロジェクトのマッピング 53

J2EEANT SCLM 言語変換プログラム 41

J2EEBIN

言語変換プログラム 60

SCLM 言語変換プログラム 40

J2EEBLD SCLM タイプ 42

J2EEEAR SCLM タイプ 44

J2EEJAR SCLM タイプ 44

J2EELIST SCLM タイプ 43

J2EEOBJ SCLM 言語変換プログラム 42

J2EEPART SCLM 言語変換プログラム

40

J2EEPART 言語変換プログラム 60

J2EEWAR SCLM タイプ 44

Java

必要なレベル 5

参照: JAVA/J2EE

JAVA SCLM 言語変換プログラム 40

JAVA 言語変換プログラム 60

JAVABIN SCLM 言語変換プログラム 41

JAVABIN 言語変換プログラム 60

JAVACLAS SCLM タイプ 43

JAVALIST SCLM タイプ 43

JAVA/J2EE

言語変換プログラム 37

言語変換プログラム・モジュールへの
アクセス 37

サポートする SCLM タイプ 42

サンプルのプロジェクト定義 37

ビルド 39

ビルドの要約 38

Ant XML ビルド・スケルトン 52

Ant のインストールとカスタマイズ
21

SCLM メンバー形式 44

WORKAREA の使用 6

Java/J2EE SCLM タイプ 44

JAVA/J2EE の概念 85

JAVA_BIN

z/OS UNIX システム・サービスのファ
イル・システム・ディレクトリ・
パス 62

JAVA_HOME 環境変数 22

L

LOGS ディレクトリー
作成 5
読み取り/書き込みアクセス権限 14
LONGLANG キーワード 9
LONGLANG=SCLM 言語 67

M

MIGRATE 機能
ロング/ショート・ネーム・レコードの
処理の改善 105
MIGRATE 処理 106

N

NOLONGLANG=SCLM 言語 67
NOTRANLANG=SCLM 言語 67

P

PASS ディレクティブ、接続に失敗した場
合の確認 25
PATH、要件 78
PROJECT 63
PROJECT ディレクトリー
オプション用のストレージ域 63
作成 6
project.conf の作成 65
project.conf、作成 65
PROMOTEAPPROVER オプション 66
PROMOTESECURITY オプション 66

R

RACF
考慮事項 14
個々のユーザー要件 5
ユーザー ID の OMVS セグメントの
作成 14
REXX
必要なソフトウェア 14
ビルド・パラメーター入カストリン
グ・サンプルのセットアップ 78
プロモート・パラメーター入カストリ
ング・サンプルのセットアップ 78
ホスト・インターフェース呼び出しサ
ンプル 78
ロング/ショート・ネーム変換のサンプ
ル 106
CRON ビルド exec サンプル 79
RSE のインストール、確認 28

S

SBWBSAMP ライブラリー
サンプルの変換プログラムのソース
37
内容 77
BWBGLOB サンプル 61
BWBPROJ サンプル 65
BWBSITE サンプル 63
JAVA/J2EE Ant XML のビルド・スケ
ルトン 52
SCLM
概念 83
概要 83
SCLM Developer Toolkit
インストール 3
カスタマイズ 3
SCLM カスタマイズ
SCLM 管理者による 37
SCLM 管理者
SCLM カスタマイズ 37
SCLM セキュリティ 71
SCLM タイプ 42
ARCHDEF 43
J2EEBLD 42
J2EEEAR 44
J2EEJAR 44
J2EELIST 43
J2EEWAR 44
SCLM の概念 83
SCLM の概要 83
SCLM の言語定義 40
SCLM ファイル・フォーマット
J2EE Ant 48
J2EE ARCHDEF 44
\$GLOBAL 44
SCLM 変換プログラム
技術要約 103
SCLM メンバー形式 44
SITE.conf
作成 63
SITE 構成ファイル 63
SMP/E のインストール xiv
STEPLIB に関する要件 78

T

TEXT 言語変換プログラム 60
TRANLANG キーワード 9
TRANLANG=SCLM 言語 67
TRANSLATE 構成ファイルの例 10
TRANSLATE 処理 105
TRANSLATE.conf
オーバーライド 68
カスタマイズ 8
作成 6

TRANSLATE.conf (続き)
内容 8
ロケーション 7
TRANTABLE グローバル変数
VSAM ファイル 62

U

UNIX システム・サービス
パイプ記号 79
CRON を介したビルドおよびプロモー
トの開始 77
USS
参照: UNIX システム・サービス

W

WORKAREA ディレクトリー
作成 5
目的 6
読み取り/書き込みアクセス権限 6

Z

z/OS
ソフトウェア要件 4
HTTP サーバー
参照: HTTP server
UNIX システム・サービス
JAVA_HOME 環境変数 22
UNIX システム・サービス・ファイ
ル・システム
ビルドおよびプロモート構成ファイ
ルのセットアップ 77
変数 78
UNIX システム・サービス・ファイ
ル・システムのマウント・ポイント
接続に失敗した場合の確認 25
参照: UNIX システム・サービス

[特殊文字]

\$GLOBAL メンバー
異なる設定の作成 62
すべてのグループの最上位レベルに設
定 62
CLASSPATH_JARS パラメーター 41
JAVA/J2EE ビルド・プロセスへの情報
の提供 61



プログラム番号: 5655-S72

Printed in Japan

SC88-4670-01



日本アイ・ビー・エム株式会社
〒106-8711 東京都港区六本木3-2-12