

バージョン 6.0.1



マイグレーション・ガイド

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、『特記事項』に記載されている情報をお読みください。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

<http://www.ibm.com/jp/manuals/main/mail.html>

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

<http://www.ibm.com/jp/manuals/> の「ご注文について」をご覧ください。

(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： SC10-4211-01
WebSphere Integration Developer
Migration Guide
Version 6.0.1

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1刷 2006.4

この文書では、平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財)日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

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

© Copyright IBM Japan 2006

目次

WebSphere Integration Developer を使 用したアプリケーションのマイグレーショ ン	1
PDF 版	1
チュートリアル: WebSphere Integration Developer へ のマイグレーション	1
WebSphere Integration Developer へのマイグレーショ ン	1
WebSphere InterChange Server からの WebSphere Process Server へのマイグレーション	2

WebSphere MQ Workflow から WebSphere Integration Developer へのマイグレーション	33
WebSphere Studio Application Developer Integration Edition から WebSphere Integration Developer へのソース成果物のマイグレーション . .	39
特記事項	111

WebSphere Integration Developer を使用したアプリケーションのマイグレーション

WebSphere® Integration Developer Version 6.0 には、既存の環境をマイグレーションするために必要なツールが用意されています。

以下のトピックでは、WebSphere Integration Developer にマイグレーションする場合の概念、参照、および段階的指示について説明します。

PDF 版

このマイグレーション情報は、PDF 形式でも入手できます。

本書は、PDF ファイルでも入手できます。

PDF ファイルを表示するには Adobe Acrobat が必要です。このソフトウェアのフリー版は、www.adobe.com で入手できます。

チュートリアル: WebSphere Integration Developer へのマイグレーション

このチュートリアルでは、既存のサービス・プロジェクトをマイグレーションする方法、およびそのサービス・プロジェクトを WebSphere Integration Developer を使用してモジュール・プロジェクトにマイグレーションする方法について説明します。

チュートリアルには 1 つのレッスンがあり、ムービー・フォーマットになっています。

- レッスン 1: サービス・プロジェクトのマイグレーションでは、既存のサービス・プロジェクトを取り出して、それをモジュール・プロジェクトにマイグレーションする方法が示されます。

以下のリンクをクリックするとチュートリアルが起動します。

注: 以下のリンクは、この Web サイトでは動作しません。リンクが正しく機能するには、WebSphere> Integration Developer がインストール済みでなければなりません。

チュートリアル: WebSphere Integration Developer へのマイグレーション

WebSphere Integration Developer へのマイグレーション

WebSphere Integration Developer Version 6.0 には、既存の環境をマイグレーションするために必要なツールが用意されています。

以下のトピックでは、WebSphere Integration Developer にマイグレーションする場合の概念、参照、および段階的指示について説明します。

WebSphere InterChange Server からの WebSphere Process Server へのマイグレーション

WebSphere InterChange Server から WebSphere Process Server へのマイグレーションは、以下の機能を介してサポートされます。

注: WebSphere Process Server のこのリリースでのマイグレーションに関連する制限事項についての詳細は、リリース情報を参照してください。

- 以下のものから呼び出せる、マイグレーション・ツールを介したソース成果物の自動マイグレーション。
 - WebSphere Integration Developer の「ファイル」→「インポート」メニュー
 - WebSphere Integration Developer の初期画面
 - 「WebSphere Process Server - First Steps」からのマイグレーション・ウィザード
 - **reposMigrate** コマンド行ユーティリティ
- 多くの WebSphere InterChange Server API のランタイムでのネイティブ・サポート
- 現行の WebSphere Business Integration Adapter テクノロジーに対するサポート。これにより、既存のアダプターが WebSphere Process Server と互換性を持つことになります。

ソース成果物のマイグレーションがサポートされているにしても、広範囲にわたる分析とテストを行い、結果として得られるアプリケーションが、WebSphere Process Server で期待されたとおりに機能するかどうか、あるいはマイグレーション後の再設計を必要とするかどうかを判断することが推奨されます。この推奨事項は、WebSphere InterChange Server とこのバージョンの WebSphere Process Server との機能的な同等性についての以下の制限に基づいています。このバージョンの WebSphere Process Server には、以下の WebSphere InterChange Server 機能に相当するもののサポートがありません。

- アクセス・インターフェース
- アダプターで開始される同期要求応答
- タイムアウト付き同期送信サービス呼び出し
- Async_in サービス呼び出し
- 分離
- イベント順序付け
- ホット・デプロイメント/動的更新
- セキュリティー - 監査
- セキュリティー - きめ細かな RBAC
- グループ・サポート
- スケジューラー - 一時停止操作
- セキュリティー記述子はマイグレーションされません
- マップおよびコラボレーション・テンプレートのマイグレーション時のカスタム XML Snippet 変換はサポートされません

WebSphere InterChange Server のサポート対象マイグレーション・パス

WebSphere Process Server マイグレーション・ツールは、WebSphere InterChange Server のバージョン 4.2.2、4.2.3、および 4.3 からのマイグレーションをサポートします。

バージョン 4.2.2 以前の WebSphere InterChange Server リリースはすべて最初に、バージョン 4.2.2、4.2.3、または 4.3 にマイグレーションしてから、WebSphere Process Server にマイグレーションする必要があります。

WebSphere InterChange Server からのマイグレーションの準備

WebSphere InterChange Server から WebSphere Process Server にマイグレーションする前に、環境が適切に準備されていることを最初に確認する必要があります。WebSphere Process Server には、WebSphere InterChange Server からマイグレーションするために必要なツールが用意されています。

これらのマイグレーション・ツールは、以下から呼び出すことができます。

- WebSphere Integration Developer の「ファイル」→「インポート」メニュー
- WebSphere Integration Developer の初期画面
- 「WebSphere Process Server - First Steps」からのマイグレーション・ウィザード

マイグレーション・ツールへの入力は、WebSphere InterChange Server からエクスポートされたリポジトリ JAR ファイルです。このため、これらのオプションのいずれかからマイグレーション・ツールにアクセスする前に、最初に以下を行う必要があります。

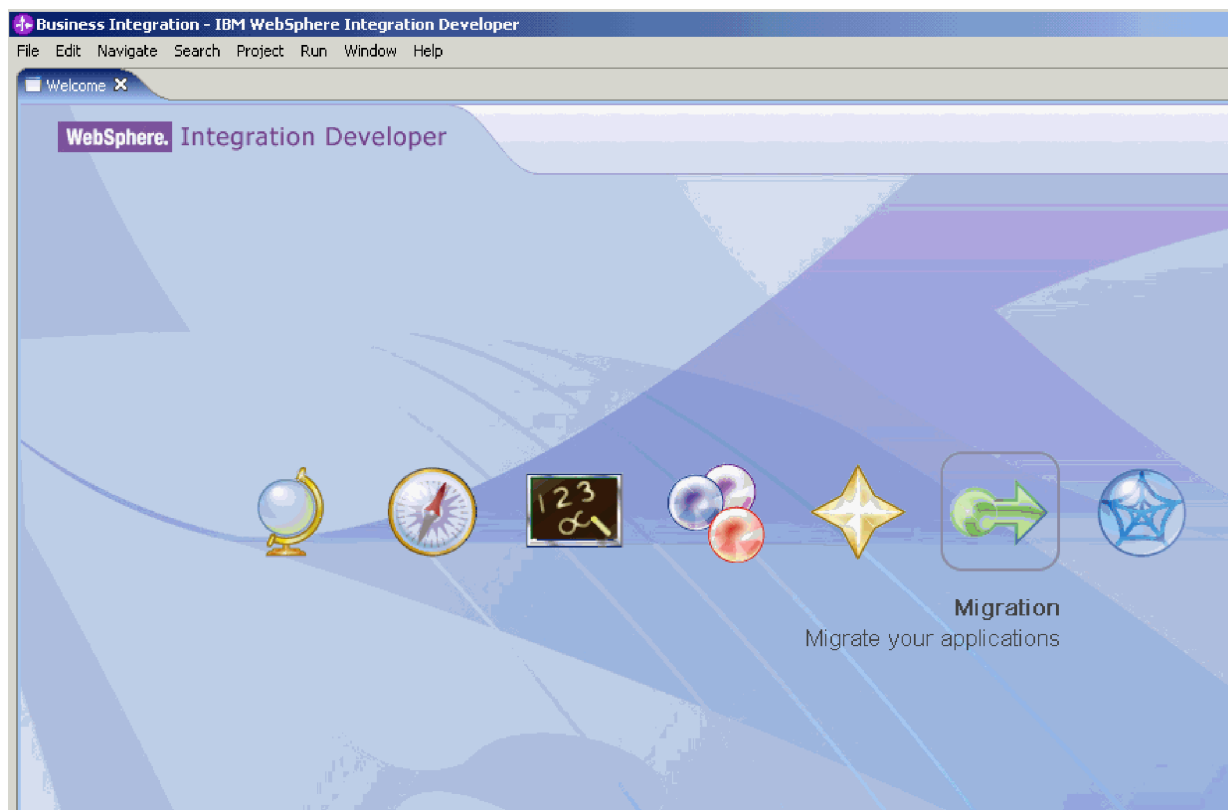
1. WebSphere Process Server にマイグレーション可能な WebSphere InterChange Server のバージョンを実行していることを確認します。トピック『WebSphere InterChange Server のサポート対象マイグレーション・パス』を参照してください。
2. WebSphere InterChange Server のドキュメンテーションで説明されているように、WebSphere InterChange Server **repos_copy** コマンドを使用して、WebSphere InterChange Server からリポジトリ JAR ファイルにソース成果物をエクスポートします。この JAR ファイルは、マイグレーション・ツールの入力になります。

マイグレーション・ウィザードを使用した WebSphere InterChange Server のマイグレーション

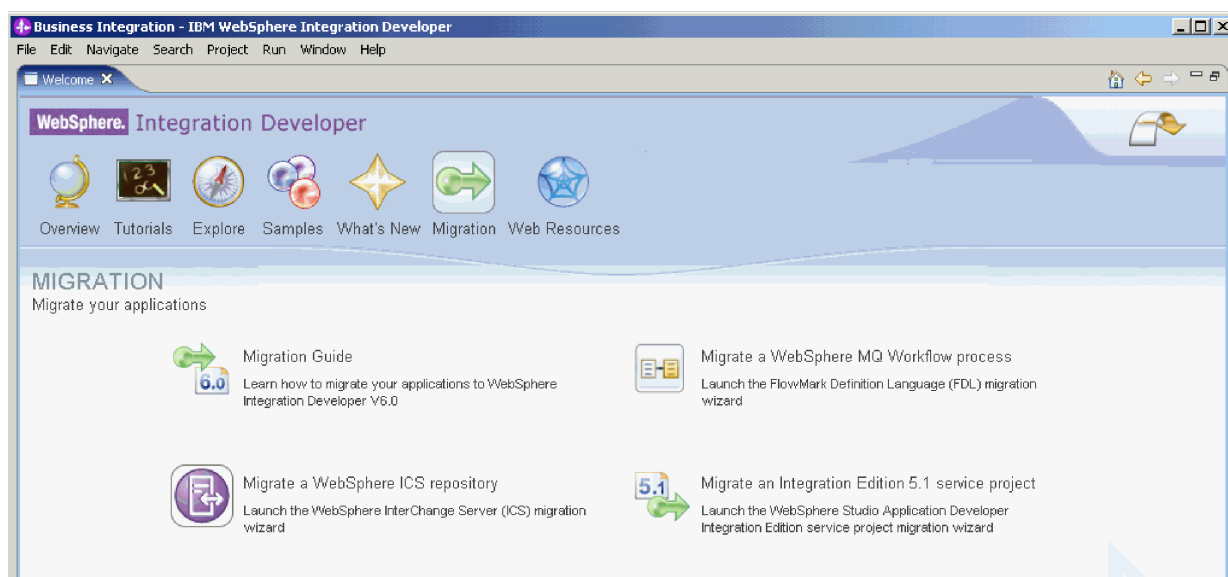
WebSphere Integration Developer マイグレーション・ウィザードを使用して、既存の WebSphere InterChange Server 成果物をマイグレーションできます。

マイグレーション・ウィザードを使用して WebSphere InterChange Server 成果物をマイグレーションするには、以下のステップに従います。

1. ウェルカム・ページから、 をクリックしてマイグレーション・ページを開きます。

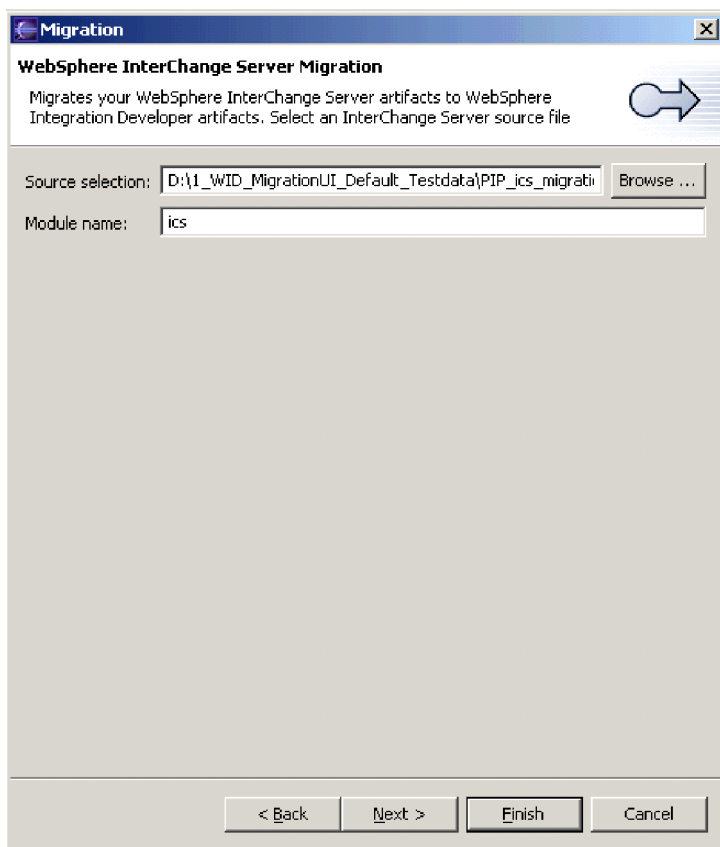


2. 「マイグレーション」ページから、「WebSphere ICS リポジトリをマイグレーションする」オプションを選択します。

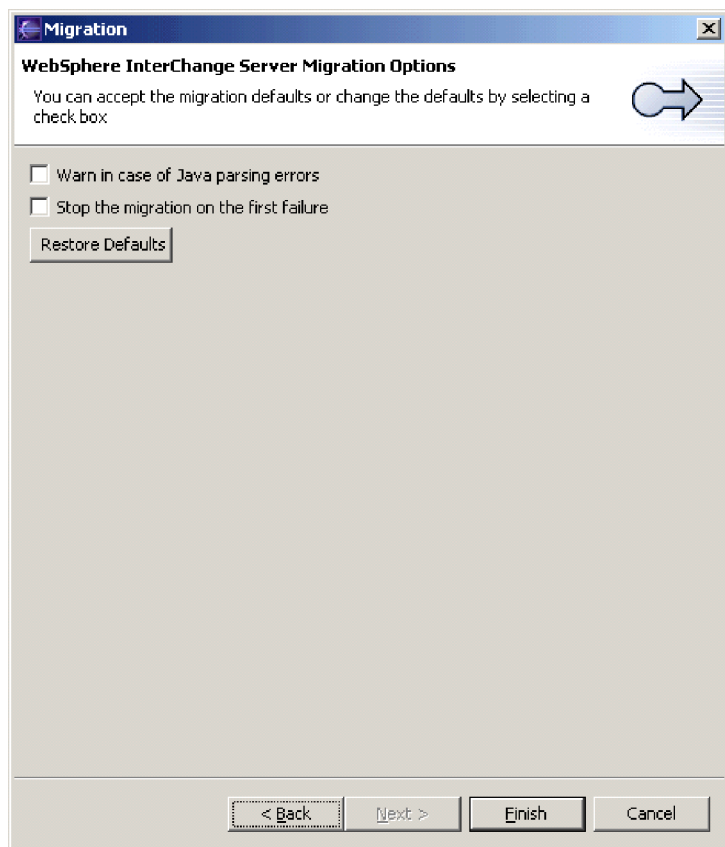


マイグレーション・ウィザードは、「ファイル」→「インポート」メニュー・オプションから起動したり、「**WebSphere InterChange Server JAR ファイル**」を選択して「次へ」をクリックすることで起動したりすることもできます。

3. マイグレーション・ウィザードがオープンします。「参照」ボタンをクリックしてファイルに移動し、ソース・ファイルの名前を「ソース」選択フィールドに入力します。関連するフィールドにモジュール名を入力します。「次へ」をクリックします。
- 4 IBM WebSphere Integration Developer: マイグレーション・ガイド



4. 「マイグレーション・オプション」ウィンドウが開きます。ここでマイグレーションのデフォルトを受け入れるか、チェック・ボックスを選択してオプションを変更することができます。



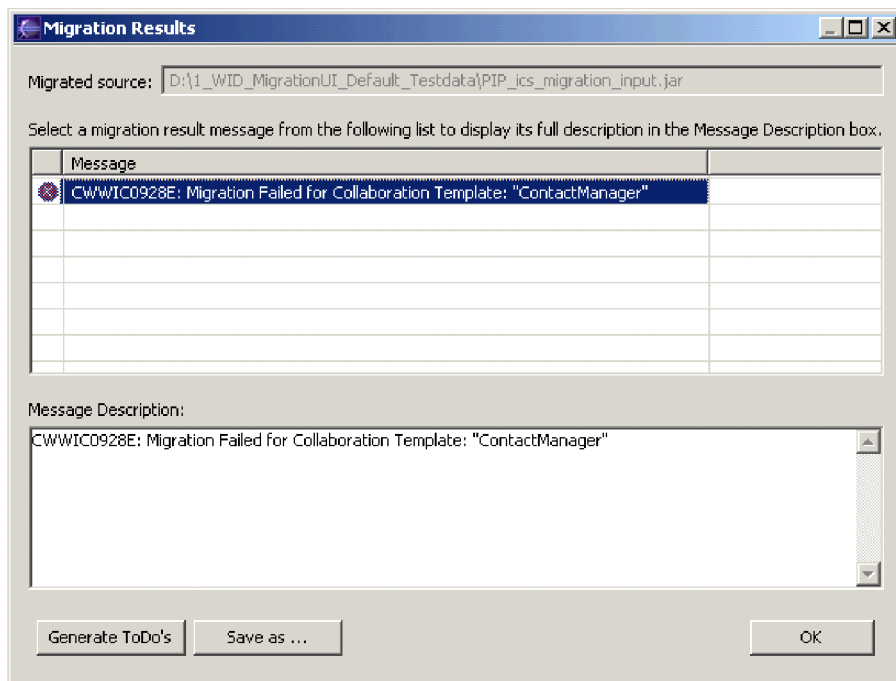
「終了」をクリックします。

WebSphere InterChange Server マイグレーションの検査

WebSphere InterChange Server の jar ファイルのマイグレーション中にエラーが報告されなかった場合は、成果物のマイグレーションが正常に行われています。マイグレーション・ウィザードが正常に完了しなかった場合は、エラー・メッセージ、警告メッセージ、情報メッセージのリストが表示されます。これらのメッセージを使用して、WebSphere InterChange Server マイグレーションを検査することができます。

注: WebSphere InterChange Server から WebSphere Process Server へのマイグレーションは複雑なので、本番に移行する前に、WebSphere Process Server で実行しているアプリケーションを広範囲に渡りテストし、予想通りに機能することを確認してください。

マイグレーション・ウィザードが完了すると、次のページが表示されます。



「マイグレーション結果」ウィンドウにマイグレーション・メッセージのリストが表示されます。メッセージをクリックして、詳細説明を確認します。必要な場合は、「タスクの生成」ボタンをクリックして、このメッセージ・リストから修正の必要な項目のリストを作成できます。

WebSphere InterChange Server からのマイグレーション失敗時の作業

WebSphere InterChange Server からのマイグレーションが失敗した場合、これに対処するには 2 つの方法があります。

注: 最初は、WebSphere InterChange Server により詳しくなるために、最初のオプションを選択してください。ただし、WebSphere Process Server とその新しい成果物に慣れてきたら、WebSphere Integration Developer でマイグレーションした成果物を修復するよう選択することもできます。

1. エラーの性質上許可される場合は、WebSphere InterChange Server ツール・セットを使用して WebSphere InterChange Server 成果物を調整してから、JAR ファイルを再度エクスポートし、マイグレーションを再試行できます。
2. WebSphere Integration Developer で成果物を編集して、マイグレーションした WebSphere Process Server 成果物のエラーを修正できます。

マイグレーション・ツールが処理する WebSphere InterChange Server 成果物

マイグレーション・ツールは、一部の WebSphere InterChange Server 成果物を自動的にマイグレーションできます。

以下の成果物をマイグレーションできます。

- **ビジネス・オブジェクト:** これは WebSphere Process Server のビジネス・オブジェクトになります
- **マップ:** これは WebSphere Process Server のマップになります
- **関係:** これは WebSphere Process Server の関係およびロールになります
- **コラボレーション・テンプレート:** これは WebSphere Process Server の BPEL および WSDL になります
- **コラボレーション・オブジェクト:** これは WebSphere Process Server の SCA 成果物になります

- **コネクタ定義:** これは WebSphere Process Server の SCA 成果物になります

マイグレーション・ツールは、以下の WebSphere InterChange Server の成果物/リソースについて、WebSphere Process Server 内のリソースを自動的に構成できます。

- DBConnection プール
- 関係データベース
- スケジューラー・エントリー

マイグレーション・ツールは、以下の WebSphere InterChange Server 成果物を処理しません。

- ベンチマーク成果物

サポートされる WebSphere InterChange Server の API

WebSphere Process Server および WebSphere Integration Developer で提供される WebSphere InterChange Server のソース成果物マイグレーション・ツールのほかに、WebSphere InterChange Server で提供されていた多くの API もサポートされています。マイグレーション・ツールは、マイグレーション時にできる限り多くのカスタム・スニペット・コードを保持することで、これらの WebSphere InterChange Server の API とともに機能します。

注: これらの API は、マイグレーションされた WebSphere InterChange Server のアプリケーションが Process Server の新しい API を使用するように変更されるまで、そのアプリケーションのサポートのためだけに提供されています。これらの WebSphere InterChange Server の API はいずれも推奨されないものであり、将来のリリースで除去されます。

Process Server でサポートされる WebSphere InterChange Server の API を下にリストします。これらの API は、WebSphere InterChange Server で提供される機能と同等の機能を Process Server でも提供します。これらの API の機能的な説明については、WebSphere InterChange Server のドキュメンテーションを参照してください。

BusObj

Collaboration/

- BusObj(DataObject)
- BusObj(String)
- BusObj(String, Locale)
- copy(BusObj)
- duplicate():BusObj
- equalKeys(BusObj):boolean
- equals(Object):boolean
- equalsShallow(BusObj):boolean
- exists(String):boolean
- get(int):Object
- get(String):Object
- getBoolean(String):boolean
- getBusObj(String):BusObj
- getBusObjArray(String):BusObjArray
- getCount(String):int
- getDouble(String):double

- `getFloat(String):float`
- `getInt(String):int`
- `getKeys():String`
- `getLocale():java.util.Locale`
- `getLong(String):long`
- `getLongText(String):String`
- `getString(String):String`
- `getType():String`
- `getValues():String`
- `getVerb():String`
- `isBlank(String):boolean`
- `isKey(String):boolean`
- `isNull(String):boolean`
- `isRequired(String):boolean`
- `keysToString():String`
- `set(BusObj)`
- `set(int, Object)`
- `set(String, boolean)`
- `set(String, double)`
- `set(String, float)`
- `set(String, int)`
- `set(String, long)`
- `set(String, Object)`
- `set(String, String)`
- `setContent(BusObj)`
- `setDefaultAttrValues()`
- `setKeys(BusObj)`
- `setLocale(java.util.Locale)`
- `setVerb(String)`
- `setWithCreate(String, BusObj)`
- `setWithCreate(String, BusObjArray)`
- `setWithCreate(String, Object)`
- `toString():String`
- `validData(String, boolean):boolean`
- `validData(String, BusObj):boolean`
- `validData(String, BusObjArray):boolean`
- `validData(String, double):boolean`
- `validData(String, float):boolean`
- `validData(String, int):boolean`
- `validData(String, long):boolean`

- validData(String, Object):boolean
- validData(String, String):boolean

BusObjArray

Collaboration/

- addElement(BusObj)
- duplicate():BusObjArray
- elementAt(int):BusObj
- equals(BusObjArray):boolean
- getElements():BusObj[]
- getLastIndex():int
- max(String):String
- maxBusObjArray(String):BusObjArray
- maxBusObjs(String):BusObj[]
- min(String):String
- minBusObjArray(String):BusObjArray
- minBusObjs(String):BusObj[]
- removeAllElements()
- removeElement(BusObj)
- removeElementAt(int)
- setElementAt(int, BusObj)
- size():int
- sum(String):double
- swap(int, int)
- toString():String

BaseDLM

DLM/

- BaseDLM(BaseMap)
- getDBConnection(String):CwDBConnection
- getDBConnection(String, boolean):CwDBConnection
- getName():String
- getRelConnection(String):DtpConnection
- implicitDBTransactionBracketing():boolean
- isTraceEnabled(int):boolean
- logError(int)
- logError(int, Object[])
- logError(int, String)
- logError(int, String, String)
- logError(int, String, String, String)
- logError(int, String, String, String, String)

- `logError(int, String, String, String, String, String)`
- `logError(String)`
- `logInfo(int)`
- `logInfo(int, Object[])`
- `logInfo(int, String)`
- `logInfo(int, String, String)`
- `logInfo(int, String, String, String)`
- `logInfo(int, String, String, String, String)`
- `logInfo(int, String, String, String, String, String)`
- `logInfo(String)`
- `logWarning(int)`
- `logWarning(int, Object[])`
- `logWarning(int, String)`
- `logWarning(int, String, String)`
- `logWarning(int, String, String, String)`
- `logWarning(int, String, String, String, String)`
- `logWarning(int, String, String, String, String, String)`
- `logWarning(String)`
- `raiseException(RuntimeEntityException)`
- `raiseException(String, int)`
- `raiseException(String, int, Object[])`
- `raiseException(String, int, String)`
- `raiseException(String, int, String, String)`
- `raiseException(String, int, String, String, String)`
- `raiseException(String, int, String, String, String, String)`
- `raiseException(String, int, String, String, String, String, String)`
- `raiseException(String, String)`
- `releaseRelConnection(boolean)`
- `trace(int, int)`
- `trace(int, int, Object[])`
- `trace(int, int, String)`
- `trace(int, int, String, String)`
- `trace(int, int, String, String, String)`
- `trace(int, int, String, String, String, String)`
- `trace(int, int, String, String, String, String, String)`
- `trace(int, String)`
- `trace(String)`

CwDBCConnection

CwDBCConnection/

CxCommon/

- beginTransaction()
- commit()
- executePreparedSQL(String)
- executePreparedSQL(String, Vector)
- executeSQL(String)
- executeSQL(String, Vector)
- executeStoredProcedure(String, Vector)
- getUpdateCount():int
- hasMoreRows():boolean
- inTransaction():boolean
- isActive():boolean
- nextRow():Vector
- release()
- rollback()

CwDBConstants

CwDBConnection/

CxCommon/

- PARAM_IN - 0
- PARAM_INOUT - 1
- PARAM_OUT - 2

CwDBStoredProcedureParam

CwDBConnection/

CxCommon/

- CwDBStoredProcedureParam(int, Array)
- CwDBStoredProcedureParam(int, BigDecimal)
- CwDBStoredProcedureParam(int, boolean)
- CwDBStoredProcedureParam(int, Boolean)
- CwDBStoredProcedureParam(int, byte[])
- CwDBStoredProcedureParam(int, double)
- CwDBStoredProcedureParam(int, Double)
- CwDBStoredProcedureParam(int, float)
- CwDBStoredProcedureParam(int, Float)
- CwDBStoredProcedureParam(int, int)
- CwDBStoredProcedureParam(int, Integer)
- CwDBStoredProcedureParam(int, java.sql.Blob)
- CwDBStoredProcedureParam(int, java.sql.Clob)
- CwDBStoredProcedureParam(int, java.sql.Date)
- CwDBStoredProcedureParam(int, java.sql.Struct)
- CwDBStoredProcedureParam(int, java.sql.Time)
- CwDBStoredProcedureParam(int, java.sql.Timestamp)

- CwDBStoredProcedureParam(int, Long)
- CwDBStoredProcedureParam(int, String)
- CwDBStoredProcedureParam(int, String, Object)
- getParamType():int getValue():Object

DtpConnection

Dtp/

CxCommon/

- beginTran()
- commit()
- executeSQL(String)
- executeSQL(String, Vector)
- executeStoredProcure(String, Vector)
- getUpdateCount():int
- hasMoreRows():boolean
- inTransaction():boolean
- isActive():boolean
- nextRow():Vector
- rollback()

DtpDataConversion

Dtp/

CxCommon/

- BOOL_TYPE - 4
- CANNOTCONVERT - 2
- DATE_TYPE - 5
- DOUBLE_TYPE - 3
- FLOAT_TYPE - 2
- INTEGER_TYPE - 0
- LONGTEXT_TYPE - 6
- OKTOCONVERT - 0
- POTENTIALDATALOSS - 1
- STRING_TYPE - 1
- UNKNOWN_TYPE - 999
- getType(double):int
- getType(float):int
- getType(int):int
- getType(Object):int
- isOKToConvert(int, int):int
- isOKToConvert(String, String):int
- toBoolean(boolean):Boolean
- toBoolean(Object):Boolean

- toDouble(double):Double
- toDouble(float):Double
- toDouble(int):Double
- toDouble(Object):Double
- toFloat(double):Float
- toFloat(float):Float
- toFloat(int):Float
- toFloat(Object):Float
- toInteger(double):Integer
- toInteger(float):Integer
- toInteger(int):Integer
- toInteger(Object):Integer
- toPrimitiveBoolean(Object):boolean
- toPrimitiveDouble(float):double
- toPrimitiveDouble(int):double
- toPrimitiveDouble(Object):double
- toPrimitiveFloat(double):float
- toPrimitiveFloat(int):float
- toPrimitiveFloat(Object):float
- toPrimitiveInt(double):int
- toPrimitiveInt(float):int
- toPrimitiveInt(Object):int
- toString(double):String
- toString(float):String
- toString(int):String
- toString(Object):String

DtpDate

Dtp/

CxCommon/

- DtpDate()
- DtpDate(long, boolean)
- DtpDate(String, String)
- DtpDate(String, String, String[], String[])
- addDays(int):DtpDate
- addMonths(int):DtpDate
- addWeekdays(int):DtpDate
- addYears(int):DtpDate
- after(DtpDate):boolean
- before(DtpDate):boolean
- calcDays(DtpDate):int

- calcWeekdays(DtpDate):int
- get12MonthNames():String[]
- get12ShortMonthNames():String[]
- get7DayNames():String[]
- getCWDate():String
- getDayOfMonth():String
- getDayOfWeek():String
- getHours():String
- getIntDay():int
- getIntDayOfWeek():int
- getIntHours():int
- getIntMilliseconds():int
- getIntMinutes():int
- getIntMonth():int
- getIntSeconds():int
- getIntYear():int
- getMaxDate(BusObjArray, String, String):DtpDate
- getMaxDateBO(BusObj[], String, String):BusObj[]
- getMaxDateBO(BusObjArray, String, String):BusObj[]
- getMinDate(BusObjArray, String, String):DtpDate
- getMinDateBO(BusObj[], String, String):BusObj[]
- getMinDateBO(BusObjArray, String, String):BusObj[]
- getMinutes():String
- getMonth():String
- getMSSince1970():long
- getNumericMonth():String
- getSeconds():String
- getShortMonth():String
- getYear():String
- set12MonthNames(String[], boolean)
- set12MonthNamesToDefault()
- set12ShortMonthNames(String[])
- set12ShortMonthNamesToDefault()
- set7DayNames(String[])
- set7DayNamesToDefault()
- toString():String
- toString(String):String
- toString(String, boolean):String

DtpMapService

Dtp/

CxCommon/

- runMap(String, String, BusObj[], CxExecutionContext):BusObj[]

DtpSplitString

Dtp/

CxCommon/

- DtpSplitString(String, String)
- elementAt(int):String
- firstElement():String
- getElementCount():int
- getEnumeration():Enumeration
- lastElement():String
- nextElement():String
- prevElement():String
- reset()

DtpUtils

Dtp/

CxCommon/

- padLeft(String, char, int):String
- padRight(String, char, int):String
- stringReplace(String, String, String):String
- truncate(double):int
- truncate(double, int):double
- truncate(float):int
- truncate(float, int):double
- truncate(Object):int
- truncate(Object, int):double

IdentityRelationship

relationship/

utilities/

crossworlds/

com/

- addMyChildren(String, String, BusObj, String, Object, CxExecutionContext)
- deleteMyChildren(String, String, BusObj, String, CxExecutionContext)
- deleteMyChildren(String, String, BusObj, String, Object, CxExecutionContext)
- foreignKeyLookup(String, String, BusObj, String, BusObj, String, CxExecutionContext)
- foreignKeyXref(String, String, String, BusObj, String, BusObj, String, CxExecutionContext)
- maintainChildVerb(String, String, String, BusObj, String, BusObj, String, CxExecutionContext, boolean, boolean)

- maintainCompositeRelationship(String, String, BusObj, Object, CxExecutionContext)
- maintainSimpleIdentityRelationship(String, String, BusObj, BusObj, CxExecutionContext)
- updateMyChildren(String, String, BusObj, String, String, String, String, CxExecutionContext)

MapExeContext

Dtp/

CxCommon/

- ACCESS_REQUEST - "SUBSCRIPTION_DELIVERY"
- ACCESS_RESPONSE - "ACCESS_RETURN_REQUEST"
- EVENT_DELIVERY - "SUBSCRIPTION_DELIVERY"
- getConnName():String
- getGenericBO():BusObj
- getInitiator():String
- getLocale():java.util.Locale
- getOriginalRequestBO():BusObj
- SERVICE_CALL_FAILURE - "CONSUME_FAILED"
- SERVICE_CALL_REQUEST - "CONSUME"
- SERVICE_CALL_RESPONSE - "DELIVERBUSOBJ"
- setConnName(String)
- setInitiator(String)
- setLocale(java.util.Locale)

Participant

RelationshipServices/

Server/

- Participant(String, String, int, BusObj)
- Participant(String, String, int, String)
- Participant(String, String, int, long)
- Participant(String, String, int, int)
- Participant(String, String, int, double)
- Participant(String, String, int, float)
- Participant(String, String, int, boolean)
- Participant(String, String, BusObj)
- Participant(String, String, String)
- Participant(String, String, long)
- Participant(String, String, int)
- Participant(String, String, double)
- Participant(String, String, float)
- Participant(String, String, boolean)
- getBoolean():boolean
- getBusObj():BusObj
- getDouble():double

- getFloat():float
- getInstanceId():int
- getInt():int
- getLong():long
- getParticipantDefinition():String
- getRelationshipDefinition():String
- getString():String INVALID_INSTANCE_ID
- set(boolean)
- set(BusObj)
- set(double)
- set(float)
- set(int)
- set(long)
- set(String)
- setInstanceId(int)
- setParticipantDefinition(String)
- setRelationshipDefinition(String)
- setParticipantDefinition(String)
- setRelationshipDefinition(String)

Relationship

RelationshipServices/

Server/

- addMyChildren(String, String, BusObj, String, Object, CxExecutionContext)
- addParticipant(Participant):int
- addParticipant(String, String, boolean):int
- addParticipant(String, String, BusObj):int
- addParticipant(String, String, double):int
- addParticipant(String, String, float):int
- addParticipant(String, String, int):int
- addParticipant(String, String, int, boolean):int
- addParticipant(String, String, int, BusObj):int
- addParticipant(String, String, int, double):int
- addParticipant(String, String, int, float):int
- addParticipant(String, String, int, int):int
- addParticipant(String, String, int, long):int
- addParticipant(String, String, int, String):int
- addParticipant(String, String, long):int
- addParticipant(String, String, String):int
- create(Participant):int
- create(String, String, boolean):int

- create(String, String, BusObj):int
- create(String, String, double):int
- create(String, String, float):int
- create(String, String, int):int
- create(String, String, long):int
- create(String, String, String):int
- deactivateParticipant(Participant)
- deactivateParticipant(String, String, boolean)
- deactivateParticipant(String, String, BusObj)
- deactivateParticipant(String, String, double)
- deactivateParticipant(String, String, float)
- deactivateParticipant(String, String, int)
- deactivateParticipant(String, String, long)
- deactivateParticipant(String, String, String)
- deactivateParticipantByInstance(String, String, int)
- deactivateParticipantByInstance(String, String, int, boolean)
- deactivateParticipantByInstance(String, String, int, BusObj)
- deactivateParticipantByInstance(String, String, int, double)
- deactivateParticipantByInstance(String, String, int, float)
- deactivateParticipantByInstance(String, String, int, int)
- deactivateParticipantByInstance(String, String, int, long)
- deactivateParticipantByInstance(String, String, int, String)
- deleteMyChildren(String, String, BusObj, String, CxExecutionContext)
- deleteMyChildren(String, String, BusObj, String, Object, CxExecutionContext)
- deleteParticipant(Participant)
- deleteParticipant(String, String, boolean)
- deleteParticipant(String, String, BusObj)
- deleteParticipant(String, String, double)
- deleteParticipant(String, String, float)
- deleteParticipant(String, String, int)
- deleteParticipant(String, String, long)
- deleteParticipant(String, String, String)
- deleteParticipantByInstance(String, String, int)
- deleteParticipantByInstance(String, String, int, boolean)
- deleteParticipantByInstance(String, String, int, BusObj)
- deleteParticipantByInstance(String, String, int, double)
- deleteParticipantByInstance(String, String, int, float)
- deleteParticipantByInstance(String, String, int, int)
- deleteParticipantByInstance(String, String, int, long)
- deleteParticipantByInstance(String, String, int, String)

- `getNewID(String):int`
- `maintainCompositeRelationship(String, String, BusObj, Object, CxExecutionContext)`
- `maintainSimpleIdentityRelationship(String, String, BusObj, BusObj, CxExecutionContext)`
- `retrieveInstances(String, boolean):int[]`
- `retrieveInstances(String, BusObj):int[]`
- `retrieveInstances(String, double):int[]`
- `retrieveInstances(String, float):int[]`
- `retrieveInstances(String, int):int[]`
- `retrieveInstances(String, long):int[]`
- `retrieveInstances(String, String):int[]`
- `retrieveInstances(String, String, boolean):int[]`
- `retrieveInstances(String, String, BusObj):int[]`
- `retrieveInstances(String, String, double):int[]`
- `retrieveInstances(String, String, float):int[]`
- `retrieveInstances(String, String, int):int[]`
- `retrieveInstances(String, String, long):int[]`
- `retrieveInstances(String, String, String):int[]`
- `retrieveInstances(String, String[], boolean):int[]`
- `retrieveInstances(String, String[], BusObj):int[]`
- `retrieveInstances(String, String[], double):int[]`
- `retrieveInstances(String, String[], float):int[]`
- `retrieveInstances(String, String[], int):int[]`
- `retrieveInstances(String, String[], long):int[]`
- `retrieveInstances(String, String[], String):int[]`
- `retrieveParticipants(String, int):Participant[]`
- `retrieveParticipants(String, String, int):Participant[]`
- `retrieveParticipants(String, String[], int):Participant[]`
- `updateMyChildren(String, String, BusObj, String, String, String, String, CxExecutionContext)`
- `updateParticipant(String, String, BusObj)`
- `updateParticipantByInstance(Participant)`
- `updateParticipantByInstance(String, String, int)`
- `updateParticipantByInstance(String, String, int, BusObj)`

UserStoredProcedureParam

Dtp/

CxCommon/

- `UserStoredProcedureParam(int, String, Object, String, String)`
- `getParamDataTypeJavaObj():String`
- `getParamDataTypeJDBC():int`
- `getParamIndex():int`
- `getParamIOType():String`

- `getParamName():String`
- `getParamValue():Object`
- `setParamDataJavaObj(String)`
- `setParamDataJDBC(int)`
- `setParamIndex(int)`
- `setParamIOType(String)`
- `setParamName(String)`
- `setParamValue(Object)`

In StoredProcedureParam

- `PARAM_TYPE_IN` - "IN"
- `PARAM_TYPE_OUT` - "OUT"
- `PARAM_TYPE_INOUT` - "INOUT"
- `DATA_TYPE_STRING` - "String"
- `DATA_TYPE_INTEGER` - "Integer"
- `DATA_TYPE_DOUBLE` - "Double"
- `DATA_TYPE_FLOAT` - "Float"
- `DATA_TYPE_BOOLEAN` - "Boolean"
- `DATA_TYPE_TIME` - "java.sql.Time"
- `DATA_TYPE_DATE` - "java.sql.Date"
- `DATA_TYPE_TIMESTAMP` - "java.sql.Timestamp"
- `DATA_TYPE_BIG_DECIMAL` - "java.math.BigDecimal"
- `DATA_TYPE_LONG_INTEGER` - "Long"
- `DATA_TYPE_BINARY` - "byte[]"
- `DATA_TYPE_CLOB` - "Clob"
- `DATA_TYPE_BLOB` - "Blob"
- `DATA_TYPE_ARRAY` - "Array"
- `DATA_TYPE_STRUCT` - "Struct"
- `DATA_TYPE_REF` - "Ref"

BaseCollaboration

Collaboration/

- `BaseCollaboration(com.ibm.bpe.api.ProcessInstanceData)`
- `AnyException` - "AnyException"
- `AppBusObjDoesNotExist` - "BusObjDoesNotExist"
- `AppLogOnFailure` - "AppLogOnFailure"
- `AppMultipleHits` - "AppMultipleHits"
- `AppRequestNotYetSent` - "AppRequestNotYetSent"
- `AppRetrieveByContentFailed` - "AppRetrieveByContent"
- `AppTimeOut` - "AppTimeOut"
- `AppUnknown` - "AppUnknown"

- `AttributeException` - `"AttributeException"`
- `existsConfigProperty(String):boolean`
- `getConfigProperty(String):String`
- `getConfigPropertyArray(String):String[]`
- `getCurrentLoopIndex():int`
- `getDBConnection(String):CwDBConnection`
- `getDBConnection(String, boolean):CwDBConnection` `getLocale():java.util.Locale`
- `getMessage(int):String`
- `getMessage(int, Object[]):String`
- `getName():String`
- `implicitDBTransactionBracketing():boolean`
- `isCallerInRole(String):boolean`
- `isTraceEnabled(int):boolean`
- `JavaException` - `"JavaException"`
- `logError(int)`
- `logError(int, Object[])`
- `logError(int, String)`
- `logError(int, String, String)`
- `logError(int, String, String, String)`
- `logError(int, String, String, String, String)`
- `logError(int, String, String, String, String, String)`
- `logError(String)`
- `logInfo(int)`
- `logInfo(int, Object[])`
- `logInfo(int, String)`
- `logInfo(int, String, String)`
- `logInfo(int, String, String, String)`
- `logInfo(int, String, String, String, String)`
- `logInfo(int, String, String, String, String, String)`
- `logInfo(String)`
- `logWarning(int)`
- `logWarning(int, Object[])`
- `logWarning(int, String)`
- `logWarning(int, String, String)`
- `logWarning(int, String, String, String)`
- `logWarning(int, String, String, String, String)`
- `logWarning(int, String, String, String, String, String)`
- `logWarning(String)`
- `not(boolean):boolean` `ObjectException` - `"ObjectException"`
- `OperationException` - `"OperationException"`

- raiseException(CollaborationException)
- raiseException(String, int)
- raiseException(String, int, Object[])
- raiseException(String, int, String)
- raiseException(String, int, String, String)
- raiseException(String, int, String, String, String)
- raiseException(String, int, String, String, String, String, String)
- raiseException(String, String)
- ServiceCallException - "ConsumerException"
- ServiceCallTransportException - "ServiceCallTransportException"
- SystemException - "SystemException"
- trace(int, int)
- trace(int, int, Object[])
- trace(int, int, String)
- trace(int, int, String, String)
- trace(int, int, String, String, String)
- trace(int, int, String, String, String, String)
- trace(int, int, String, String, String, String, String)
- trace(int, String)
- trace(String)
- TransactionException - "TransactionException"

CxExecutionContext

CxCommon/

- CxExecutionContext()
- getContext(String):Object
- MAPCONTEXT - "MAPCONTEXT"
- setContext(String, Object)

CollaborationException

Collaboration/

- getMessage():String
- getMsgNumber():int
- getSubType():String
- getText():String
- getType():String
- toString():String

Filter

crossworlds/ com/

- Filter(BaseCollaboration)
- filterExcludes(String, String):boolean
- filterIncludes(String, String):boolean
- recurseFilter(BusObj, String, boolean, String, String):boolean
- recursePreReqs(String, Vector):int

Globals

**crossworlds/
com/**

- Globals(BaseCollaboration)
- callMap(String, BusObj):BusObj

SmartCollabService

**crossworlds/
com/**

- SmartCollabService()
- SmartCollabService(BaseCollaboration)
- doAgg(BusObj, String, String, String):BusObj
- doMergeHash(Vector, String, String):Vector
- doRecursiveAgg(BusObj, String, String, String):BusObj
- doRecursiveSplit(BusObj, String):Vector
- doRecursiveSplit(BusObj, String, boolean):Vector
- getKeyValues(BusObj, String):String
- merge(Vector, String):BusObj
- merge(Vector, String, BusObj):BusObj
- split(BusObj, String):Vector

StateManagement

**crossworlds/
com/**

- StateManagement()
- beginTransaction()
- commit()
- deleteBO(String, String, String)
- deleteState(String, String, String, int)
- persistBO(String, String, String, String, BusObj)
- recoverBO(String, String, String):BusObj
- releaseDBConnection()
- resetData()
- retrieveState(String, String, String, int):int
- saveState(String, String, String, String, int, int, double)
- setDBConnection(CwDBConnection)

- updateBO(String, String, String, String, BusObj)
- updateState(String, String, String, String, int, int)

EventKeyAttrDef

EventManagement/

CxCommon/

- EventKeyAttrDef()
- EventKeyAttrDef(String, String)
- public String keyName
- public String keyValue

EventQueryDef

EventManagement/

CxCommon/

- EventQueryDef()
- EventQueryDef(String, String, String, String, int)
- public String nameConnector
- public String nameCollaboration
- public String nameBusObj
- public String verb
- public int ownerType

FailedEventInfo

EventManagement/

CxCommon/

- FailedEventInfo()
- FailedEventInfo(String x6, int, EventKeyAttrDef[], int, int, String, String, int)
- public String nameOwner
- public String nameConnector
- public String nameBusObj
- public String nameVerb
- public String strTime
- public String strMessage
- public int wipIndex
- public EventKeyAttrDef[] strbusObjKeys
- public int nKeys
- public int eventStatus
- public String expirationTime
- public String scenarioName
- public int scenarioState

CwBiDiEngine

- BiDiBOTransformation(BusinessObject, String, String, boolean):BusinessObj

- BiDiBusObjTransformation(BusObj, String, String, boolean):BusObj
- BiDiStringTransformation(String, String, String):String

WebSphere InterChange Server XML からの WebSphere Process Server DataObject のマップ

レガシー・アダプターを使用して WebSphere Process Server に接続する場合、下のアルゴリズムを使用すると、どのようにして WebSphere Process Server DataObject が WebSphere InterChange Server XML から作成されたのかについて、詳しく理解することができます。この情報には、データ値が配置された場所や、WebSphere InterChange Server で使用されていたデータ値を置換するために選択されたデータ値が示されます。

一般

- ChangeSummary に verb を設定する場合、設定はすべて **markCreate/Update/Delete** API を使用して行われます。
- ChangeSummary/EventSummary に verb を設定する場合、**Create**、**Update**、および **Delete** verb が ChangeSummary に設定され、それ以外の verb はすべて EventSummary に設定されます。
- ChangeSummary から verb を取得する場合:
 - isDelete が true の場合、verb は **Delete** になります。

注: isDelete が true の場合、isCreate の値は無視されます。このような状況が発生する可能性があるのは、作成のマークが付けられてから DataObject がハブに入り、その時点で DataObject が削除された場合、またはハブで作成と削除の両方が行われた場合です。

- isDelete が false で、isCreate が true の場合、verb は **Create** になります。
- isDelete が false で、isCreate も false の場合、verb は **Update** になります。
- ロギングを使用可能にしている場合、意図した **Update** ではなく **Create** として DataObject が識別されることを避けるには、次の操作を行う必要があります。
 - DataObject の作成中はロギングを中断する。
 - DataObject の更新の場合にロギングを再開する (または、**markUpdated** API を使用する)。

ロード (Loading)

「ロード中」では、WebSphere InterChange Server ランタイム XML が WebSphere Business Integration BusinessGraph AfterImage インスタンスにロードされます。

- 該当する BusinessGraph のインスタンスが作成されます。
- ChangeSummary ロギングがオンになり、後でこのロギングをオンにしてエントリーをクリアすることがないようにします。
- 不要な情報が ChangeSummary に入力されないようにするため、ChangeSummary ロギングが一時停止されます。
- 最上位の BusinessObject の属性が DataObject に作成されます (下のセクション『属性処理』を参照してください)。
- 最上位の BusinessObject に子の BusinessObject がある場合、再帰的に処理が行われます。
- 子の BusinessObject の属性が DataObject に作成されます (下のセクション『属性処理』を参照してください)。
- 最上位の BusinessObject の verb が BusinessGraph の最上位の verb に設定され、要約内にも設定されます。

- 子の BusinessObject の verb が要約内に設定されます。

保管 (Saving)

「保管中」では、WebSphere Business Integration BusinessGraph AfterImage インスタンスが WebSphere InterChange Server ランタイム XML に保管されます。入力された BusinessGraph が AfterImage ではない場合、例外が throw されます。

- WebSphere InterChange Server XML 文書のインスタンスが作成されます。
- ChangeSummary 内で削除されたすべての DataObject が、元の位置に存在するかのように処理されます。

注: この削除の取り消しプロセスでは、リストの順序は保持されません。

- 最上位の BusinessObject の verb が BusinessGraph の最上位の verb に設定されます。
- 最上位の BusinessObject の属性が DataObject から設定されます (下のセクション『属性処理』を参照してください)。
- DataObject に子の DataObject がある場合、再帰的に処理が行われます。
- 子の DataObject の verb が次のように処理されます。
 - EventSummary または ChangeSummary に子の DataObject の verb がない場合、verb は "" (空のストリング) に設定されます。
 - verb が EventSummary にあって ChangeSummary にない場合、この verb が使用されます。
 - verb が ChangeSummary にあって EventSummary にない場合、この verb が使用されます。
 - verb が ChangeSummary と EventSummary の両方にある場合、ChangeSummary の値は EventSummary の値に優先して選択されるため、ChangeSummary の verb が使用されます。
- 子の DataObject の属性が BusinessObject に設定されます (下のセクション『属性処理』を参照してください)。

属性処理

- 下で説明されていない値はすべて ASIS としてロード/保管されます。
- ObjectEventId は EventSummary にロードされ、EventSummary から保管されます。
- **CxBlank** および **CxIgnore** の場合:
 - 変換の WebSphere Business Integration BusinessObject の側では、**CxBlank** および **CxIgnore** は次のように設定/識別されます。
 - **CxIgnore** - NULL の Java™ 値で設定解除/設定を行う
 - **CxBlank** - 下の表にあるように従属値を入力する
 - 変換の WebSphere InterChange Server XML の側では、**CxBlank** および **CxIgnore** は次のように設定/識別されます。

表 1. CxBlank および CxIgnore の設定

型	CxIgnore	CxBlank
整数	Integer.MIN_VALUE	Integer.MAX_VALUE
浮動	Float.MIN_VALUE	Float.MAX_VALUE
倍精度	Double.MIN_VALUE	Double.MAX_VALUE
ストリング/日付/長形式テキスト	"CxIgnore"	""
子の BusinessObject	(空の要素)	N/A

WebSphere InterChange Server マイグレーション・プロセスのベスト・プラクティス

以下のガイドラインは、WebSphere InterChange Server 向けのインテグレーション成果物の開発に役立つことを目的としています。これらのガイドラインに従うことで、WebSphere InterChange Server 成果物の WebSphere Process Server に容易にマイグレーションできます。

これらの推奨事項は、新しいインテグレーション・ソリューションの開発のためのガイドとしてのみ使用されることを意図したものです。既存の内容は、これらのガイドラインに従っていない場合もあります。また、これらのガイドラインから逸脱しなければならない場合があることも理解してください。これらの場合には、成果物のマイグレーションに必要な再作業の量を最小化するために、逸脱の範囲を制限するように注意を払う必要があります。ここで概説されるガイドラインは、WebSphere InterChange Server 成果物一般の開発向けのベスト・プラクティスを網羅しているわけではないという点に注意してください。それよりむしろ、将来、成果物をマイグレーションする場合の容易さに影響する考慮事項に範囲を限定しています。

一般的開発

ほとんどのインテグレーション成果物に広く適用される考慮事項がいくつかあります。一般に、ツールによって提供される機能を活用し、ツールによって強制されるメタデータ・モデルに準拠する成果物は、最も円滑にマイグレーションされます。また、大幅な拡張と外部依存を持つ成果物は、マイグレーション時に、より多くの手動介入を必要とする傾向にあります。

以下のリストでは、将来のマイグレーションを容易にするために役立つ、WebSphere InterChange Server ベースのソリューションの一般的開発についてのベスト・プラクティスを要約します。

- リアルタイムの自動化されたプロセスのインテグレーション・ソリューションに対して、WebSphere InterChange Server を使用する
- システムおよびコンポーネントの設計を文書化する
- 開発ツールを使用してインテグレーション成果物を編集する
- ツールおよび Java Snippet を使用するルールを定義するために、ベスト・プラクティスを活用する

自明のことではありますが、インテグレーション・ソリューションが WebSphere InterChange Server によって提供されるプログラミング・モデルおよびアーキテクチャーに準拠していることが重要です。これは、リアルタイムの自動化されたプロセスのインテグレーション・ソリューションに最も適しています。また、WebSphere InterChange Server 内のインテグレーション・コンポーネントはそれぞれ、アーキテクチャー内で、厳密に定義されたロールを果たします。このモデルから大幅に逸脱すると、コンテンツを WebSphere Process Server 上の該当する成果物にマイグレーションすることが、より一層難しくなります。

将来のマイグレーション・プロジェクトの成功率を大幅に高めるもう 1 つの一般的なベスト・プラクティスは、システム設計を文書化することです。これには、サービス要件の機能設計と品質、プロジェクト全体で共用される成果物の相互依存性、およびデプロイ時に行われた設計上の意思決定も含めた、インテグレーション・アーキテクチャーと設計を必ず取り込むようにしてください。これにより、マイグレーション時のシステム分析が支援され、再作業の労力が最小化されます。

成果物定義の作成、構成、および変更の際には、提供されている開発ツールのみを使用することがきわめて重要です。成果物メタデータの手動操作 (例えば、XML ファイルの直接編集) は避けてください。これを行うと、マイグレーション対象の成果物が壊れる可能性があります。

Java コードを、コラボレーション・テンプレート、マップ、共通コード・ユーティリティ、およびその他のコンポーネント内で開発する場合、考慮すべき注意事項が以下のようにいくつかあります。

- 公開されている API のみを使用する
- Activity Editor を使用すること

- アダプターを使用して EIS にアクセスすること
- Java Snippet コードでの外部依存関係を避けること
- ポータビリティのために、J2EE の一般的な開発方法に準拠していること
- スレッドの作成またはスレッド同期プリミティブの使用を行わないこと。これらを作成または使用しなければならない場合、マイグレーション時に、これらを、非同期 Bean を使用するように変換する必要があります。
- java.io.* を使用したディスク I/O を行わないこと。 JDBC を使用して、データを保管してください。
- EJB コンテナ用に予約されている可能性のある、ソケット I/O、クラス・ロード、ネイティブ・ライブラリーのロードなどの機能を実行しないこと。これらを実行しなければならない場合、マイグレーション時に、これらの Snippet を EJB コンテナ機能を使用するように手動で変換する必要があります。

成果物用の製品ドキュメンテーションで公開されている API のみを使用してください。これらは、WebSphere InterChange Server の開発ガイドに概要が詳しく説明されています。多くの場合、互換性 API が WebSphere Process Server で提供されますが、公開済みの API のみが組み込まれます。WebSphere InterChange Server には、開発者が使用したくなる内部インターフェースがたくさん含まれていますが、将来に向かってサポートされるという保証はありません。

マップおよびコラボレーション・テンプレートでビジネス・ロジックおよび変換ルールを設計する場合は、可能な範囲で Activity Editor ツールを使用してください。これにより、ロジックは、より容易に新しい成果物に変換可能なメタデータを使用して記述されるようになります。ツールで再利用したい操作の場合は、Activity Editor の「My Collections」フィーチャーを可能な限り使用してください。WebSphere InterChange Server のクラスパスでは、Java アーカイブ (*.jar) ファイルとして組み込まれる、フィールド開発の共通コード・ユーティリティ・ライブラリーは避けるようにしてください。これらは、手動でマイグレーションする必要があります。

開発する必要がある Java コード Snippet では、コードは、可能な限り単純かつ最小単位になるようにすることが推奨されます。Java コードにおける複雑さのレベルは、スクリプト記述 (基本的な評価、演算、および計算を含む) の順序、データのフォーマット設定、型変換などによって決まるはずです。1より包括的または複雑なアプリケーション・ロジックが必要な場合は、WebSphere Application Server で稼働する EJB を使用してロジックをカプセル化することを検討し、Web サービス呼び出しを使用してそれを WebSphere InterChange Server から呼び出してください。別個にマイグレーションする必要がある、サード・パーティーまたは外部のライブラリーではなく、標準 JDK ライブラリーを使用してください。さらに、すべての関連ロジックを単一のコード Snippet 内に集め、接続およびトランザクション・コンテキストが複数のコード Snippet にまたがるロジックの使用は避けてください。例えば、データベース操作の場合、接続の取得、トランザクションの開始と終了、および接続の解放に関連するコードは、1つのコード Snippet にしてください。

一般に、エンタープライズ情報システム (EIS) とインターフェースするように設計されるコードは、アダプター内に配置し、マップまたはコラボレーション・テンプレート内には配置しないようにしてください。これは、一般にアーキテクチャー設計のためのベスト・プラクティスです。これはまた、接続管理や、考えられる Java Native Interface (JNI) 実装など、サード・パーティー・ライブラリーおよびコード内の関連考慮事項に対する前提条件の回避に役立ちます。

適切な例外処理を使用することにより、コードをできるだけ安全にしてください。また、たとえコードが現在 J2SE 環境内で稼働しているにしても、コードに互換性を持たせて、J2EE アプリケーション・サーバー環境内でも稼働するようにしてください。静的変数、スレッドの作成、ディスク I/O を回避するなど、J2EE の一般的な開発方法に従うようにしてください。これらは一般に従うべき優れたベスト・プラクティスですが、移植性にも関係します。

共通コード・ユーティリティ

先に述べたように、WebSphere InterChange Server 環境内で、インテグレーション成果物間で使用する共通コード・ユーティリティ・ライブラリーの開発を回避することが推奨されます。インテグレーション成果物間でコードの再利用が必要な場合は、Activity Editor ツールの「My Collections」フィーチャーの活用が推奨されます。また、WebSphere Application Server で稼働する EJB を使用してロジックをカプセル化することを検討し、Web サービス呼び出しを使用してそれらを WebSphere InterChange Server から呼び出してください。一部の共通コード・ユーティリティ・ライブラリーが WebSphere Process Server 上で適切に実行されることもあり得ますが、カスタム・ユーティリティのマイグレーションは、ユーザーの責任で行ってください。

データベース接続プール

ユーザー定義のデータベース接続プールは、単純なデータ検索の場合、およびプロセス・インスタンス間でのより複雑な状態管理の場合に、マップおよびコラボレーション・テンプレート内で非常に有用です。WebSphere InterChange Server のデータベース接続プールは、WebSphere Process Server の標準 JDBC リソースとして提供され、基本機能は同じです。しかし、接続およびトランザクションの管理方法は異なる場合があります。

将来の移植性を最大化するために、コラボレーション・テンプレートまたはマップ内の複数の Java Snippet ノードに渡ってデータベース・トランザクションをアクティブに保持することは避けてください。例えば、接続の取得、トランザクションの開始と終了、および接続の解放に関連するコードは、1 つのコード Snippet に入れてください。

ビジネス・オブジェクト

ビジネス・オブジェクトの開発での主要な考慮事項は、成果物を構成するために提供されているツールのみを使用すること、データ属性に対して明示的なデータ型と長さを使用すること、および文書化された API のみを使用することです。

WebSphere Process Server 内のビジネス・オブジェクトは、強く型付けされたデータ属性を使用する Service Data Object (SDO) を基にしています。WebSphereInterChange Server およびアダプター内のビジネス・オブジェクトの場合、データ属性は、強く型付けされておらず、ユーザーはときにはストリング・データ型を非ストリング・データ属性に対して指定することがあります。WebSphere Process Server 内の問題を避けるため、データ型の指定は明示的に行ってください。

WebSphere Process Server 内のビジネス・オブジェクトは、複数のコンポーネント間で受け渡されると、実行時にシリアル化される場合があるため、データ属性に必須の長さを明示して、システム・リソースの使用率を最小化することが重要です。この理由のため、例えば、ストリング属性には最大の 255 文字長を使用しないでください。同様に、現在のところデフォルトで 255 文字になっていますが、ゼロの長さ属性も指定しないでください。その代わりに、属性に必要な正確な長さを指定してください。

WebSphere Process Server 内のビジネス・オブジェクトの属性名には、XSD NCName ルールが適用されるため、ビジネス・オブジェクトの属性の名前の中でスペースまたは「:」を使用しないでください。スペースまたは「:」の付いたビジネス・オブジェクトの属性名は、WebSphere Process Server 内では無効です。マイグレーションする前に、ビジネス・オブジェクトの属性名を変更してください。

ビジネス・オブジェクト内で配列を使用する場合は、「マップ」または「関係」(あるいは、その両方) 内の配列に索引付けする際に、配列の順序に依存することはできません。WebSphere Process Server 内にマイグレーションされる構成では、特にエントリーが削除される場合、索引順序を保証しません。

重ねて、先に述べたように、Business Object Designer ツールのみを使用してビジネス・オブジェクト定義を編集すること、およびインテグレーション成果物内のビジネス・オブジェクトに対して、公開されている API のみを使用することが重要です。

コラボレーション・テンプレート

先に述べたガイドラインの多くは、コラボレーション・テンプレートの開発に適用されます。

プロセスがメタデータを使用して適切に記述されるようにするためには、コラボレーション・テンプレートの作成および変更には、常に Process Designer ツールを使用するようにし、メタデータ・ファイルを直接に編集することは避けてください。可能な限り Activity Editor ツールを使用して、メタデータの使用を最大化し、必要なロジックを記述してください。

マイグレーションで必要となることがある手動による再作業量を最小化するために、コラボレーション・テンプレート内では、文書化された API のみを使用してください。静的変数の使用は避けてください。その代わりに、非静的変数およびコラボレーション・プロパティを使用して、ビジネス・ロジックの要件に対応してください。Java Snippet 内では、Java 修飾子 `final`、`transient`、および `native` の使用は避けてください。これらは、コラボレーション・テンプレートをマイグレーションした結果である、BPEL Java Snippet 内で使用することができません。

将来の移植性を最大化するために、「ユーザー定義のデータベース接続プール」での明示的な接続解放呼び出し、および明示的なトランザクション・ブラケット操作（すなわち、明示コミットおよび明示ロールバック）の使用は避けてください。その代わりに、コンテナが管理する暗黙接続クリーンアップ、および暗黙トランザクション・ブラケット操作を使用してください。また、コラボレーション・テンプレート内の複数の Java Snippet ノードに渡って、システム接続およびトランザクションをアクティブに保つことは避けてください。これは、外部システムへのすべての接続のほか、ユーザー定義のデータベース接続プールにもあてはまります。先に述べたように、外部 EIS 内の操作はアダプター内で管理する必要があり、データベース操作に関連するコードは 1 つのコード Snippet 内に含まれている必要があります。これは、BPEL ビジネス・プロセス・コンポーネントとして提供される場合に、割り込み可能フローとして選択的にデプロイされることがあるコラボレーション内で必要となります。この場合、プロセスはいくつかの別々のトランザクションで構成され、状態およびグローバル変数の情報のみがアクティビティー間で受け渡されます。これらの複数のプロセス・トランザクションに渡るシステム接続または関連トランザクションのコンテキストは失われます。

コラボレーション・テンプレートのプロパティ名には、特殊文字を使用しないでください。これらの特殊文字は、マイグレーション先の BPEL プロパティ名では無効です。マイグレーションで生成される BPEL での構文エラーを避けるため、マイグレーション前に、プロパティを名前変更してこれらの特殊文字を除去してください。

「this。」を使用して変数を参照しないでください。例えば、「this.inputBusObj」ではなく、「inputBusObj」を使用してください。

変数には、シナリオのスコープを持つ変数の代わりに、クラス・レベルのスコープを使用してください。シナリオのスコープは、マイグレーション時に転送されません。

Java Snippet に宣言されたすべての変数を、デフォルト値で初期化してください。例えば、「Object myObject = null;」のようにします。マイグレーション前に、宣言で、必ずすべての変数を初期化してください。

コラボレーション・テンプレートのユーザー変更可能セクションに、`Java import` 文がないことを確認してください。コラボレーション・テンプレートの定義で、インポート・フィールドを使用して、インポートする `Java` パッケージを指定してください。

マップ

コラボレーション・テンプレートに関して説明したガイドラインの多くは、マップにも適用されます。

マップがメタデータを使用して適切に記述されるようにするために、マップの作成および変更には、常に `Map Designer` ツールを使用するようにし、メタデータ・ファイルを直接に編集することは避けてください。可能な限り `Activity Editor` ツールを使用して、メタデータの使用を最大化し、必要なロジックを記述してください。

マップ内の子ビジネス・オブジェクトを参照する場合は、子ビジネス・オブジェクト用のサブマップを使用してください。

`SET` 内で `Java` コードを「値」として使用することは避けてください。これは、`WebSphere Process Server` 内では無効であるためです。代わりに、定数を使用してください。例えば、`set` の値が `"xml version=" + "1.0" + " encoding=" + "UTF-8"` である場合、これは、`WebSphere Process Server` 内では妥当性検査されません。代わりに、マイグレーションする前に、`"xml version=1.0 encoding=UTF-8"` に変更してください。

マイグレーションで必要となることがある手動による再作業量を最小化するために、コラボレーション・テンプレート内では、文書化された `API` のみを使用してください。静的変数の使用は避けてください。その代わりに、非静的変数およびコラボレーション・プロパティを使用して、ビジネス・ロジックの要件に対応してください。`Java Snippet` 内では、`Java` 修飾子 `final`、`transient`、および `native` の使用は避けてください。

ビジネス・オブジェクト内で配列を使用する場合は、「マップ」内で配列に索引付けする際に、配列の順序に依存することはできません。`WebSphere Process Server` 内にマイグレーションされる構成では、特にエントリーが削除される場合、索引順序を保証しません。

将来の移植性を最大化するために、「ユーザー定義のデータベース接続プール」での明示的な接続解放呼び出し、および明示的なトランザクション・ブラケット操作（すなわち、明示コミットおよび明示ロールバック）の使用は避けてください。その代わりに、コンテナが管理する暗黙接続クリーンアップ、および暗黙トランザクション・ブラケット操作を使用してください。また、複数の変換ノード境界に渡って、`Java Snippet` 内でシステム接続およびトランザクションをアクティブに保つことは避けてください。これは、外部システムへのすべての接続のほか、ユーザー定義のデータベース接続プールにもあてはまります。先に述べたように、外部 `EIS` 内の操作はアダプター内で管理する必要があり、データベース操作に関連するコードは 1 つのコード `Snippet` 内に含まれている必要があります。

関係

関係については、関係定義は `WebSphere Process Server` で使用するためにマイグレーションが可能であると同時に、関係テーブル・スキーマおよびインスタンス・データは、`WebSphere Process Server` による再使用、および `WebSphere InterChange Server` と `WebSphere Process Server` 間での並行した共用も可能であることに注目してください。

関係についての主な考慮事項は、関連するコンポーネントを構成するために提供されているツールのみを使用すること、およびインテグレーション成果物内での関係に、公開されている `API` のみを使用することです。

関係定義の編集には、Relationship Designer ツールのみを使用することが重要です。さらに、関係定義のデプロイメント時に自動的に生成される関係スキーマの構成は、WebSphere InterChange Server にのみ許可してください。データベース・ツールまたは SQL スクリプトを使用して関係テーブル・スキーマを直接に変更することは、しないでください。

また、関係テーブル・スキーマ内の関係インスタンス・データを手動で変更する必要がある場合は、必ず、Relationship Designer によって提供されている機能を使用してください。

先に述べたように、インテグレーション成果物内の関係に対しては、公開されている API のみを使用することが重要です。

フレームワーク・クライアントへのアクセス

CORBA IDL インターフェース API を採用して新規クライアントを開発することは、しないでください。これは、WebSphere Process Server ではサポートされていません。

WebSphere MQ Workflow から WebSphere Integration Developer へのマイグレーション

WebSphere Integration Developer には、WebSphere MQ Workflow からマイグレーションするために必要なツールが用意されています。

マイグレーション・ウィザードを使用すると、WebSphere MQ Workflow の Buildtime コンポーネントからエクスポートしたビジネス・プロセスの FDL 定義を、Business Process Choreographer の対応する成果物に変換することができます。生成された Business Process Choreographer 成果物は、ビジネス・オブジェクトの XMLSchema 定義、WSDL 定義、BPEL、および TEL 定義を構成します。

変換ツールには、WebSphere MQ Workflow Buildtime からオプション **export deep** によってエクスポートするプロセス・モデルのセマンティックに完全な FDL 定義が必要です。このオプションを使用すると、必要なデータ、プログラム、およびサブプロセス仕様のすべてがインクルードされます。また、WebSphere MQ Workflow プロセス・モデルで参照されるすべてのユーザー定義プロセス実行サーバー定義 (UPES) が、WebSphere MQ Workflow Buildtime から FDL をエクスポートするときに選択されます。

注: マイグレーション・ウィザードでは、以下のマイグレーションは行いません。

- WebSphere MQ Workflow ランタイム・インスタンス
- WebSphere MQ Workflow プログラム実行エージェント (PEA)、または WebSphere MQ Workflow プロセス実行サーバー (PES for z/OS®) によって呼び出されるプログラム・アプリケーション

WebSphere MQ Workflow からのマイグレーションの準備

WebSphere MQ Workflow から WebSphere Integration Developer にマイグレーションする前に、環境が適切に準備されていることを最初に確認する必要があります。

マッピングの範囲と完成度は、以下のマイグレーションのガイドラインにどれだけ準拠しているかによります。

- FDL プログラム・アクティビティは、純粋な **スタッフ**・アクティビティではない場合、ユーザー定義プロセスの実行サーバー (UPES) に関連付けられます。
- WebSphere MQ Workflow プログラム・アクティビティのスタッフ割り当てが TEL デフォルトの **スタッフ verb** に準拠していることを確認してください。

- 短形式の単純名を使用して、マイグレーション済みプロセス・モデルを読みやすくします。FDL 名が正しくない BPEL 名の場合があることに注意してください。マイグレーション・ウィザードで、自動的に FDL 名を有効な BPEL 名に変換できます。

マイグレーション・ウィザードは、実行可能なビジネス・プロセス・エディター成果物に手動で適用する必要のある、マイグレーションされない WebSphere MQ Workflow 構成 (PEA または PES プログラム・アクティビティ、動的スタッフ割り当てなど) に対しても、構文的に正しいビジネス・プロセス・エディター構成を作成します。

下の表に、適用されるマッピング・ルールの概要を示します。

表 2. マッピング・ルール

WebSphere MQ Workflow	Business Process Choreographer
プロセス	実行モードのプロセス: <i>longRunning</i> ; プロセスのインバウンドおよびアウトバウンドのインターフェースのパートナー・リンク
ソースおよびシンク	プロセス入力およびプロセス出力の 変数; 受信 アクティビティおよび応答 アクティビティ
プログラム・アクティビティ	呼び出し アクティビティ
プロセス・アクティビティ	呼び出し アクティビティ
空のアクティビティ	空の アクティビティ
ブロック	組み込み BPEL アクティビティを持つスコープ
アクティビティの終了条件	<i>While</i> アクティビティ (実アクティビティを囲む)
アクティビティの開始条件	アクティビティの結合条件
アクティビティのスタッフ割り当て	ヒューマン・タスク・アクティビティ
アクティビティの入力コンテナおよび出力コンテナ	呼び出し アクティビティの入出力の指定に使用する変数
コントロール・コネクタ; 遷移状態	リンク; 遷移状態
データ・コネクタ	割り当て アクティビティ
グローバル・データ・コンテナ	変数

可能であれば、最初は小さなプロジェクトのマイグレーション・プロセスを試してください。マイグレーション・ウィザードを使用すると、WebSphere MQ Workflow プロセス・モデルをビジネス・プロセス・エディターのプロセス・モデルに簡単に変換できますが、新しいプログラミング・モデルを作成するときと同じようにプロセスを 1 対 1 でマッピングすることはできません。基礎をなすプロセス仕様言語 (FDL および BPEL) のセマンティック・スコープは、交差するエリアを共用しますが、完全に重なり合うことはありません。そうでないと、ビジネス・プロセス・エディターの新しい利点が期待できなくなります。Web サービスは、使用すべきではないソリューションを新しいソリューションに置き換えることを要求する、新しいテクノロジーを表します。

一般に、生成された成果物は常時検討し、必要に応じて変更する必要があります。マイグレーションを正常に終了するため、またはマイグレーション・タスクを完了するために、さらに作業が必要になることもあります。


マイグレーション・ウィザードを使用した WebSphere MQ Workflow のマイグレーション

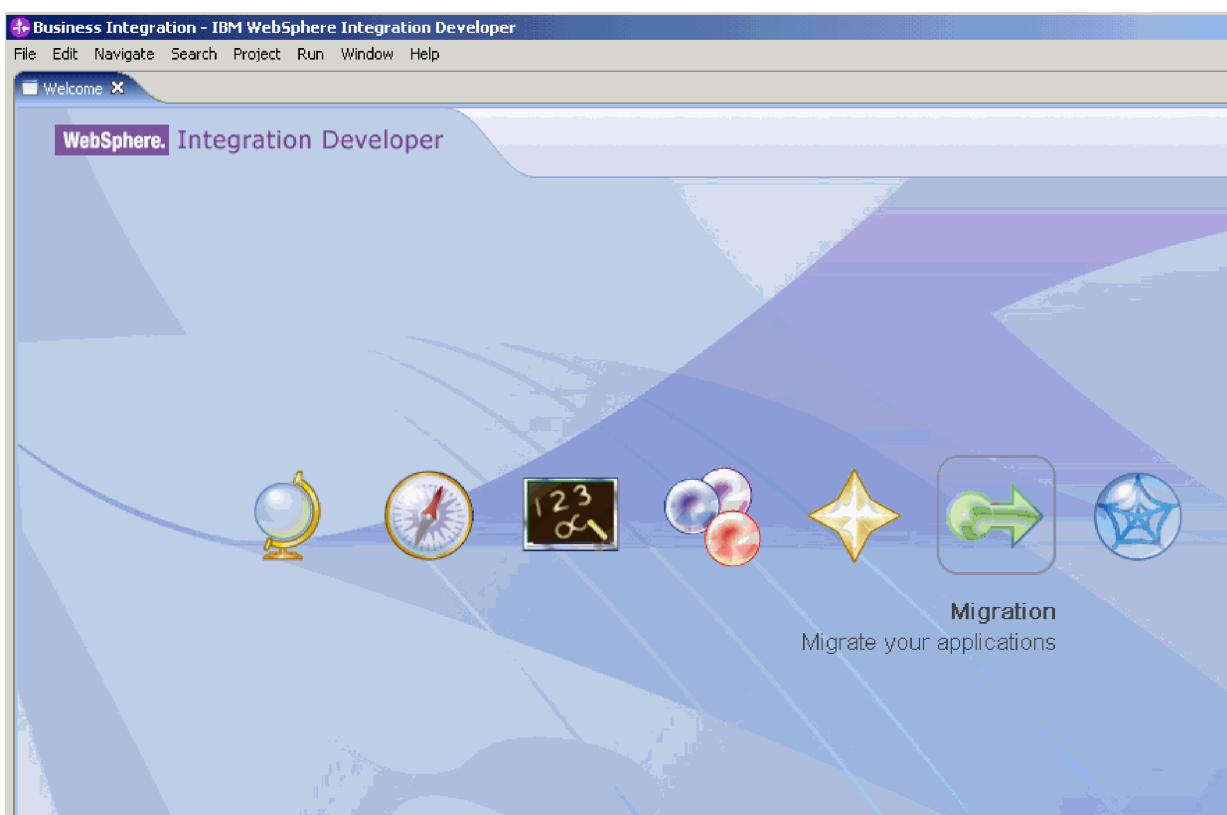
マイグレーション・ウィザードを使用すると、WebSphere MQ Workflow の Buildtime コンポーネントからエクスポートしたビジネス・プロセスの FDL 定義を、Business Process Choreographer の対応する成果物に変換することができます。生成された Business Process Choreographer 成果物は、ビジネス・オブジェクトの XMLSchema 定義、WSDL 定義、BPEL、および TEL 定義を構成します。

注: マイグレーション・ウィザードでは、以下のマイグレーションは行いません。

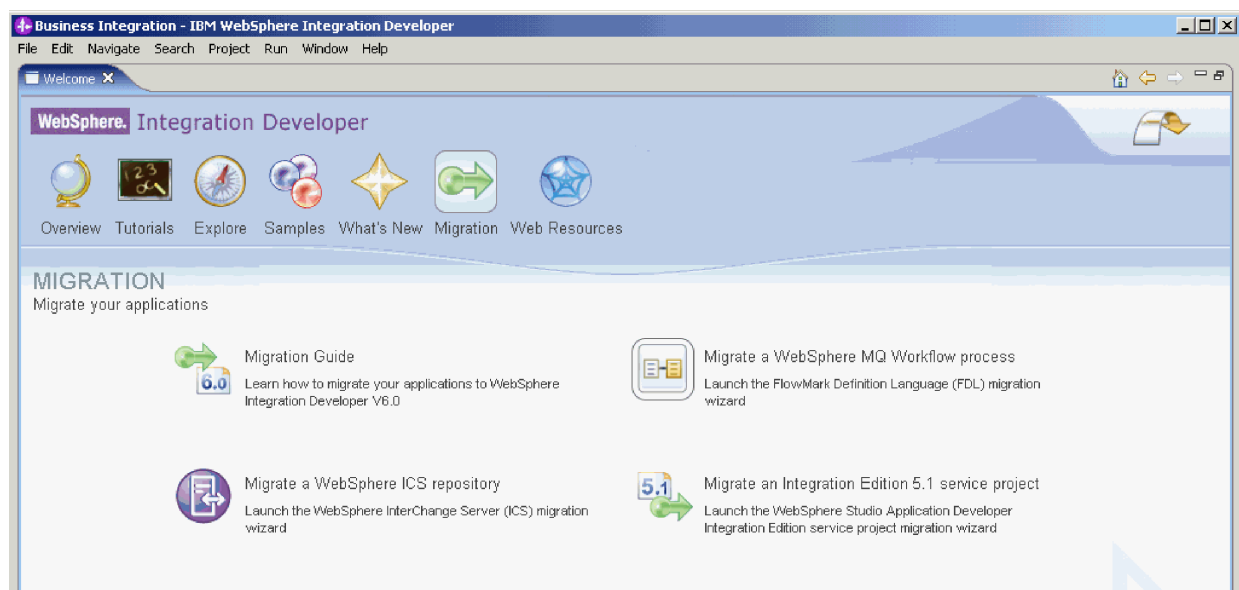
- WebSphere MQ Workflow ランタイム・インスタンス
- WebSphere MQ Workflow プログラム実行エージェント (PEA)、または WebSphere MQ Workflow プロセス実行サーバー (PES for z/OS) によって呼び出されるプログラム・アプリケーション

マイグレーション・ウィザードを使用して WebSphere MQ Workflow 成果物をマイグレーションするには、以下のステップに従います。

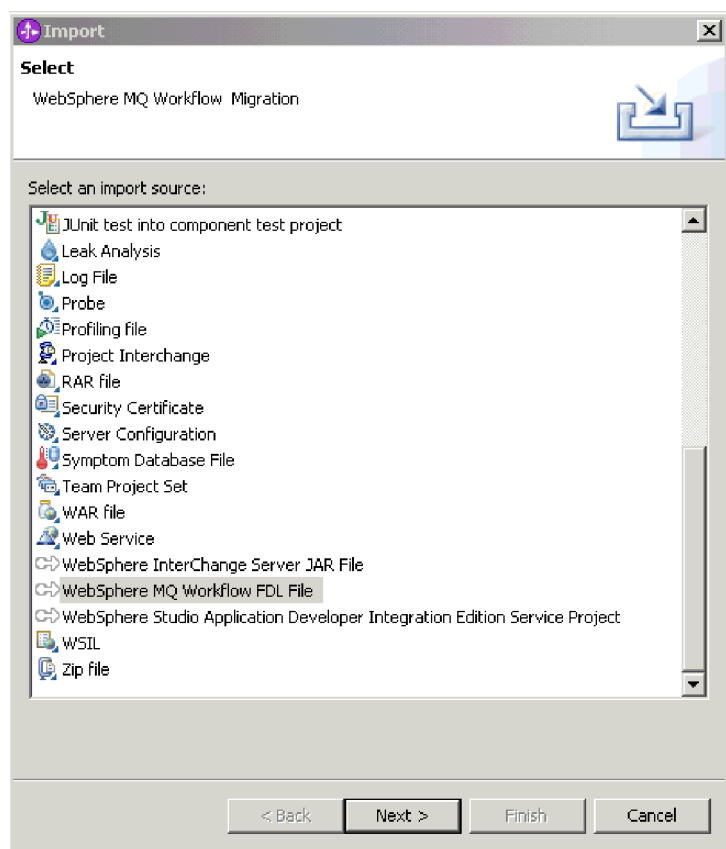
1. ウェルカム・ページから、 をクリックしてマイグレーション・ページを開きます。



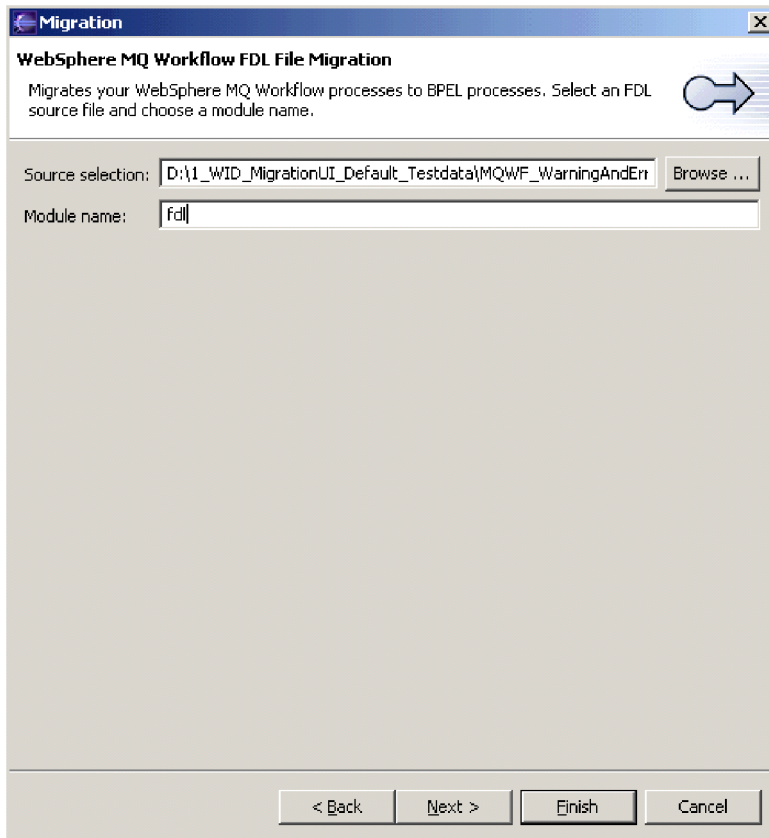
2. マイグレーション・ページから、「WebSphere MQ Workflow プロセスをマイグレーションする」オプションを選択します。



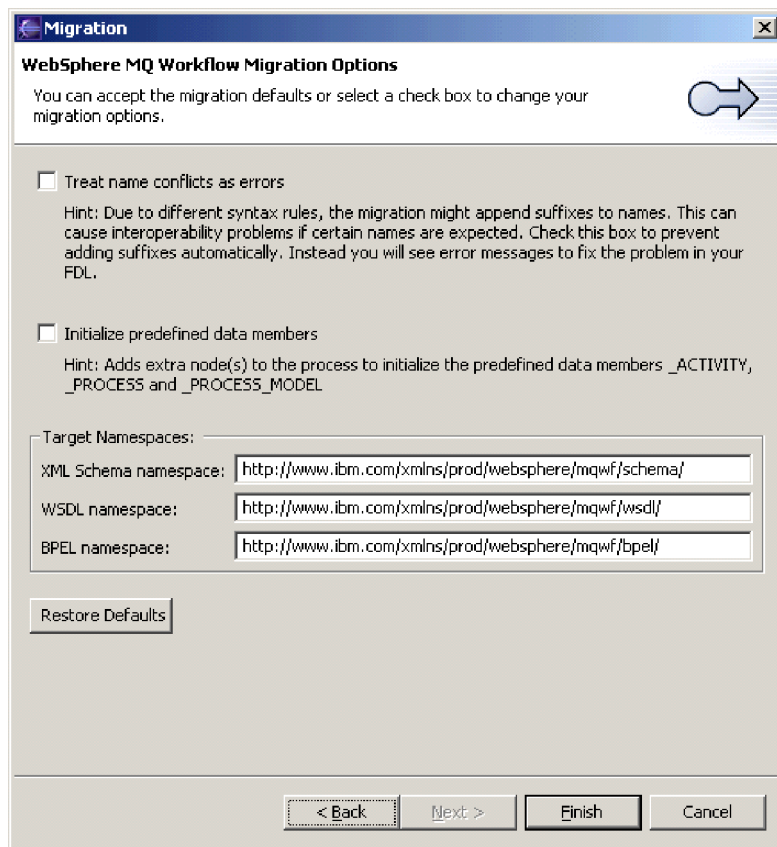
(注: 「ファイル」 → 「インポート」 → 「WebSphere MQ Workflow FDL ファイル」をクリックしてマイグレーション・ウィザードを開くこともできます。



3. マイグレーション・ウィザードがオープンします。「参照」ボタンをクリックしてファイルに移動し、.fdl ファイルの名前を「ソース選択 (Source selection)」フィールドに入力します。関連するフィールドにモジュール名を入力します。「次へ」をクリックします。



4. 「マイグレーション・オプション」ページが開きます。ここでマイグレーションのデフォルトを受け入れるか、チェック・ボックスを選択してオプションを変更することができます。「**名前の競合をエラーとして扱う**」チェック・ボックスを選択すると、インターオペラビリティ・エラーとなる可能性のある接尾部が自動的に追加されなくなります。「**事前定義データ・メンバーを初期する**」チェック・ボックスを選択すると、定義済みデータ・メンバーを初期化するために、追加のノードがプロセスに追加されます。



The image shows a Windows-style dialog box titled "Migration" with a close button (X) in the top right corner. The main title is "WebSphere MQ Workflow Migration Options". Below the title, there is a text block: "You can accept the migration defaults or select a check box to change your migration options." To the right of this text is a blue arrow icon pointing to the right. There are two checkboxes with associated hints:

- ☐ Treat name conflicts as errors
Hint: Due to different syntax rules, the migration might append suffixes to names. This can cause interoperability problems if certain names are expected. Check this box to prevent adding suffixes automatically. Instead you will see error messages to fix the problem in your FDL.
- ☐ Initialize predefined data members
Hint: Adds extra node(s) to the process to initialize the predefined data members _ACTIVITY, _PROCESS and _PROCESS_MODEL

Below the checkboxes is a section titled "Target Namespaces:" containing three text input fields:

- XML Schema namespace:
- WSDL namespace:
- BPEL namespace:

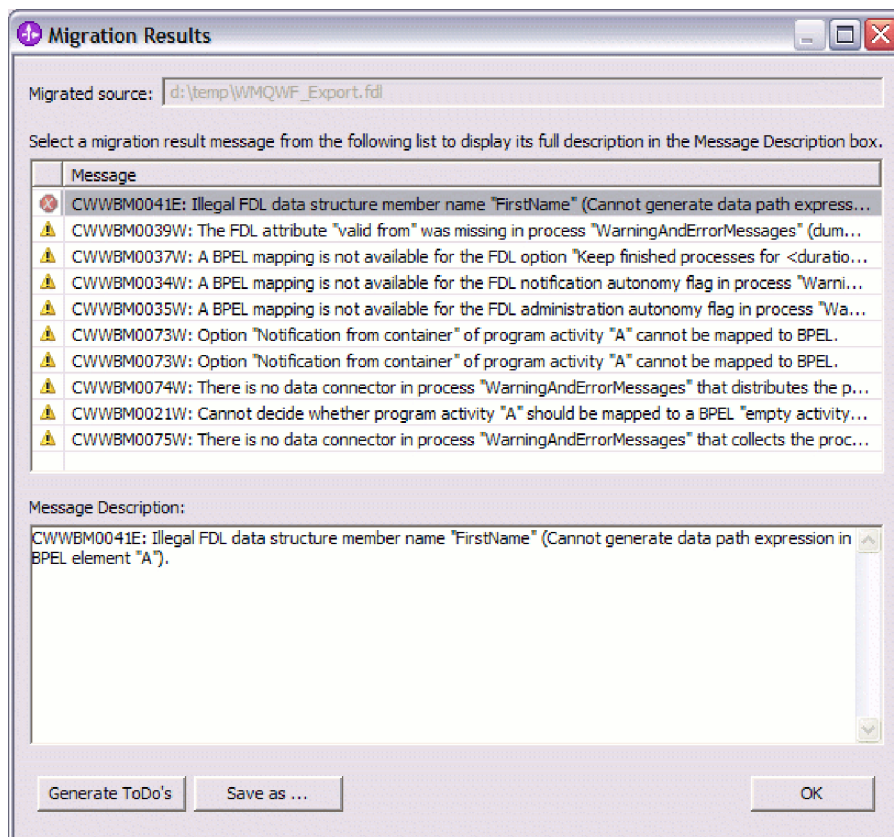
Below the input fields is a button labeled "Restore Defaults". At the bottom of the dialog box are four buttons: "< Back", "Next >", "Finish", and "Cancel".

「終了」をクリックします。

WebSphere MQ Workflow マイグレーションの検査

マイグレーション・ウィザードが正常に完了すると、エラー・メッセージ、警告メッセージ、情報メッセージのリストが表示されます。これらのメッセージを使用して、WebSphere MQ Workflow マイグレーションを検査することができます。

マイグレーション・ウィザードが完了すると、次のページが表示されます。



「マイグレーション結果」ウィンドウにマイグレーション・メッセージのリストが表示されます。メッセージをクリックして、詳細説明を確認します。必要な場合は、「タスクの生成」ボタンをクリックして、このメッセージ・リストから修正の必要な項目のリストを作成できます。

マイグレーション・プロセスの制限 (WebSphere MQ Workflow から)

WebSphere MQ Workflow マイグレーション・プロセスに関連する特定の制限があります。

FDL をマイグレーションすると、UPES アクティビティおよび対応する WSDL の呼び出しアクティビティが生成されます。ただし、呼び出しメッセージとその応答を関連付けるために使用する手法に関して、IBM® WebSphere MQ Workflow と IBM WebSphere Process Server との間ではランタイム環境に大きな違いがあります。そのため、UPES 互換レイヤーを使用できない場合は、生成される BPEL を正常に実行することができません。

WebSphere Studio Application Developer Integration Edition から WebSphere Integration Developer へのソース成果物のマイグレーション

ソース成果物は、WebSphere Studio Application Developer Integration Edition から WebSphere Integration Developer にマイグレーションできます。アプリケーションでソース成果物をマイグレーションする場合は、新しい WebSphere Integration Developer プログラミング・モデルにそれらをマイグレーションして、新しい機能やフィーチャーが使用できるようにする処理も行われます。その後、アプリケーションは、WebSphere Integration Developer Version 6.0 サーバーに再デプロイおよびインストールすることができます。

WebSphere Business Integration Server Foundation 5.1 で使用可能な多くのフィーチャーは、ベースの WebSphere Application Server 6.0 に移動しました。これらのフィーチャーのマイグレーションについてのヒントは、サイト http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.websphere.base.doc/info/aes/ae/rins_migratepme.html を参照してください。

WebSphere Studio Application Developer Integration Edition サービス・プロジェクトを完全にマイグレーションするには、完了すべき基本作業が 2 つあります。

1. マイグレーション・ウィザードを使用して、成果物をビジネス・インテグレーション・モジュール・プロジェクトに自動的にマイグレーションする。
2. WebSphere Integration Developer を使用して手動でマイグレーションを完了する。これには、自動的にマイグレーションできなかった Java コードの修正と、マイグレーションされた成果物の再ワイヤリングが含まれます。

注: ランタイム・マイグレーション (アップグレード・パス) は WebSphere Process Server 6.0.0 では提供されないため、このソース成果物のマイグレーション・パスは、6.0.0 で WebSphere Studio Integration Edition サービス・プロジェクトをマイグレーションする場合のみのオプションになります。

ソース成果物をマイグレーションする場合のサポート対象マイグレーション・パス

WebSphere Studio Application Developer Integration Edition からのソース成果物のマイグレーションを開始する前に、WebSphere Integration Developer によってサポートされるマイグレーション・パスを検討する必要があります。

マイグレーション・ウィザードでは、WebSphere Studio Application Developer Integration Edition バージョン 5.1 (またはそれ以上) のサービス・プロジェクトを一度に 1 つマイグレーションすることができます。ワークスペース全体のマイグレーションは行われません。

マイグレーション・ウィザードでは、アプリケーション・バイナリーはマイグレーションされません。WebSphere Studio Application Developer Integration Edition サービス・プロジェクトで見つかったソース成果物のみがマイグレーションされます。

マイグレーションのためのソース成果物の準備

ソース成果物を WebSphere Studio Application Developer Integration Edition から WebSphere Integration Developer にマイグレーションする前に、環境が適切に準備されていることを最初に確認する必要があります。

以下のステップでは、ソース成果物を WebSphere Studio Application Developer Integration Edition から WebSphere Integration Developer にマイグレーションする前に環境を準備する方法について説明します。

1. マイグレーションを試行する前に、5.1 のワークスペース全体のバックアップ・コピーを取っていることを確認します。
2. Rational® Application Developer Information Center (<http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.rad.migration.doc/topics/tmigratefrom51x.html>) のマイグレーションのセクションを確認して、ワークスペース内の非 WBI 固有のプロジェクトをマイグレーションする最適な方法を決定します。
3. Rational Application Developer によって提供される Web サービス機能の背景情報について、Rational Application Developer Information Center (<http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.webservices.rad.nav.doc/developingweb.html>) の Web サービスのセクションを確認します。

4. 該当するすべての WebSphere Integration Developer フィーチャーが使用可能になっていることを確認します。これらのフィーチャーが使用可能になっていないと、以下で説明するメニュー・オプションが表示されないことがあります。重要なフィーチャーを使用可能にするには、以下を行います。
 - WebSphere Integration Developer で、「ウィンドウ」メニュー項目に進み、「設定 (Preferences)」を選択します。
 - 「ワークベンチ」に進み、「ケイパビリティ (Capabilities)」カテゴリを選択します。
 - 以下のカテゴリの下のすべてのフィーチャーを選択します。
 - 拡張 J2EE
 - エンタープライズ Java
 - Integration Developer
 - Java デベロッパー
 - Web デベロッパー (標準)
 - Web サービス・デベロッパー
 - XML デベロッパー
 - 「OK」をクリックします。
5. WebSphere Integration Developer 用に新しいワークスペース・ディレクトリーを使用します。サービス・プロジェクトが含まれている古い WebSphere Studio Application Developer Integration Edition ワークスペースで WebSphere Integration Developer を開くことはお勧めしません。このプロジェクトは、WebSphere Integration Developer が読み取ることができるフォーマットに最初にマイグレーションする必要があるためです。これを行うための推奨ステップは、以下のとおりです。
 - a. すべての非サービス・プロジェクトを古いワークスペースから新しいワークスペースにコピーします。5.1 のサービス・プロジェクト用にデプロイメント・コードが生成されたときに作成された 5.1 の EJB、Web、および EAR プロジェクトはコピーしないでください。新しい 6.0 のデプロイメント・コードは、BI モジュールがビルドされるときに自動的に生成されます。
 - b. ブランクのワークスペースで WebSphere Integration Developer を開き、「ファイル」→「インポート」→「既存プロジェクトをワークスペースへ」をクリックしてすべての非サービス・プロジェクトをインポートし、新しいワークスペースにコピーしたプロジェクトを選択します。
 - プロジェクトが J2EE プロジェクトである場合、Rational Application Developer マイグレーション・ウィザードを使用して、以下のように、そのプロジェクトを 1.4 レベルにマイグレーションする必要があります。
 - 1) プロジェクトを右クリックして、「マイグレーション」→「J2EE マイグレーション・ウィザード... (J2EE Migration Wizard...)」を選択します。
 - 2) 最初のページの警告文を確認して、「次へ」をクリックします。
 - 3) プロジェクト・リスト内の「J2EE プロジェクト (J2EE project)」にチェック・マークが付いていることを確認します。「プロジェクト構造をマイグレーションする」および「J2EE 仕様レベルをマイグレーションする (Migrate J2EE specification level)」は、チェック・マークが付いたままにしてください。J2EE バージョン 1.4 および「ターゲット・サーバー WebSphere Process Server v6.0 (Target Server WebSphere Process Server v6.0)」を選択します。
 - 4) J2EE プロジェクトに適切なその他のオプションがあれば選択して、「終了」をクリックします。このステップが正常に完了すると、「マイグレーションが正常に終了しました (Migration finished successfully)」というメッセージが表示されます。
 - 5) マイグレーション後に J2EE プロジェクトにエラーがある場合は、v5 の .jar ファイルまたはライブラリーを参照しているすべてのクラスパス・エントリーを除去して、代わりにクラス

パスに **JRE システム・ライブラリー**および **WPS サーバー・ターゲット・ライブラリー**を追加する必要があります (下記に説明があります)。これで、多くのエラーが解決されるはずですが、一部のエラーが解決されない場合もあります。

- 拡張メッセージング (CMM) または Container Managed Persistence over Anything (CMP/A) を使用する WebSphere Business Integration EJB プロジェクトの場合、5.1 のプロジェクトが 6.0 のワークスペースにインポートされた後で、IBM EJB JAR 拡張記述子ファイルをマイグレーションする必要があります。詳しくは、『WebSphere Business Integration EJB プロジェクトのマイグレーション』を参照してください。
 - ワークスペースにインポートした非サービス・プロジェクトごとにクラスパスを修正します。クラスパスに JRE および WebSphere Process Server ライブラリーを追加するには、インポートしたプロジェクトを右クリックして、「**プロパティ**」を選択します。「**Java のビルド・パス**」エントリに移動して、「**ライブラリー**」タブを選択します。次に、以下を行います。
 - 1) 「**ライブラリーの追加 (Add library)**」 → 「**JRE システム・ライブラリー**」 → 「**代替 JRE - WPS Server v6.0 JRE (Alternate JRE - WPS Server v6.0 JR)**」 → 「**終了**」を選択します。
 - 2) 次に、「**ライブラリーの追加 (Add library)**」 → 「**WPS サーバー・ターゲット**」 → 「**WPS サーバー・クラスパスの構成**」 → 「**終了**」を選択します。
6. 最適の結果を得るために、マイグレーション・ウィザードを実行する前に、「**プロジェクト**」メニュー項目に移動して、「**自動的にビルド (Build Automatically)**」にチェック・マークが付いていないことを確認します。WebSphere Integration Developer は、サービス・デプロイメント・オプションが設計時に指定されるという点で、WebSphere Studio Application Developer Integration Edition と異なります。プロジェクトがビルドされると、デプロイメント・コードは、生成済み EJB と Web プロジェクトで自動的に更新されるため、手動で「**デプロイメント・コードの生成**」を行うためのオプションはなくなりました。
7. サービス・プロジェクト内の .bpel ファイルを完全にマイグレーションするために、.bpel ファイルによって参照されるすべての .wsdl および .xsd ファイルが新しいワークスペースのビジネス・インテグレーション・プロジェクトで解決可能であることを確認する必要があります。
- .wsdl および/または .xsd ファイルが .bpel ファイルと同じサービス・プロジェクトにある場合は、これ以上のアクションは不要です。
 - .wsdl ファイルまたは .xsd ファイル (あるいは、その両方) が、マイグレーションしているサービス・プロジェクトとは異なるサービス・プロジェクト内にある場合、マイグレーションの前に WebSphere Studio Application Developer Integration Edition を使用して、5.1 の成果物を再編成する必要があります。これを行う理由は、ビジネス・インテグレーション・モジュールのプロジェクトが成果物を共用できないためです。以下は、5.1 の成果物を再編成するための 2 つのオプションです。
 - WebSphere Studio Application Developer Integration Edition で、すべての共通成果物を保持する新規 Java プロジェクトを作成します。複数のサービス・プロジェクトによって共用されるすべての .wsdl ファイルおよび .xsd ファイルを、この新規 Java プロジェクト内に置きます。この新規 Java プロジェクトの依存関係を、これらの共通成果物を使用するすべてのサービス・プロジェクトに追加します。WebSphere Integration Developer で、サービス・プロジェクトをマイグレーションする前に、5.1 の共用 Java プロジェクトと同じ名前を持つ、新規ビジネス・インテグレーション・ライブラリー・プロジェクトを作成します。古い .wsdl ファイルと .xsd ファイルを、5.1 の共用 Java プロジェクトからこの新規 BI ライブラリー・プロジェクト・フォルダーに手動でコピーします。これは、BPEL サービス・プロジェクトのマイグレーションを行う前に行う必要があります。
 - もう 1 つのオプションは、サービス・プロジェクト間での依存関係がない各サービス・プロジェクトで、これらの共用の .wsdl 成果物と .xsd 成果物のローカル・コピーを保持することです。

- .wsdl および/または .xsd ファイルが他のタイプのプロジェクト (通常、他の Java プロジェクト) にある場合は、5.1 のプロジェクトと同じ名前を持つビジネス・インテグレーション・ライブラリー・プロジェクトを作成する必要があります。また、新しいライブラリー・プロジェクトのクラスパスもセットアップする必要があります。このとき、5.1 の Java プロジェクトからのエントリーがあればそれを追加します。このタイプのプロジェクトは、共用成果物の保管に役立ちます。

これでマイグレーション・プロセスを開始する準備ができました。

WebSphere Integration Developer マイグレーション・ウィザードを使用したサービス・プロジェクトのマイグレーション

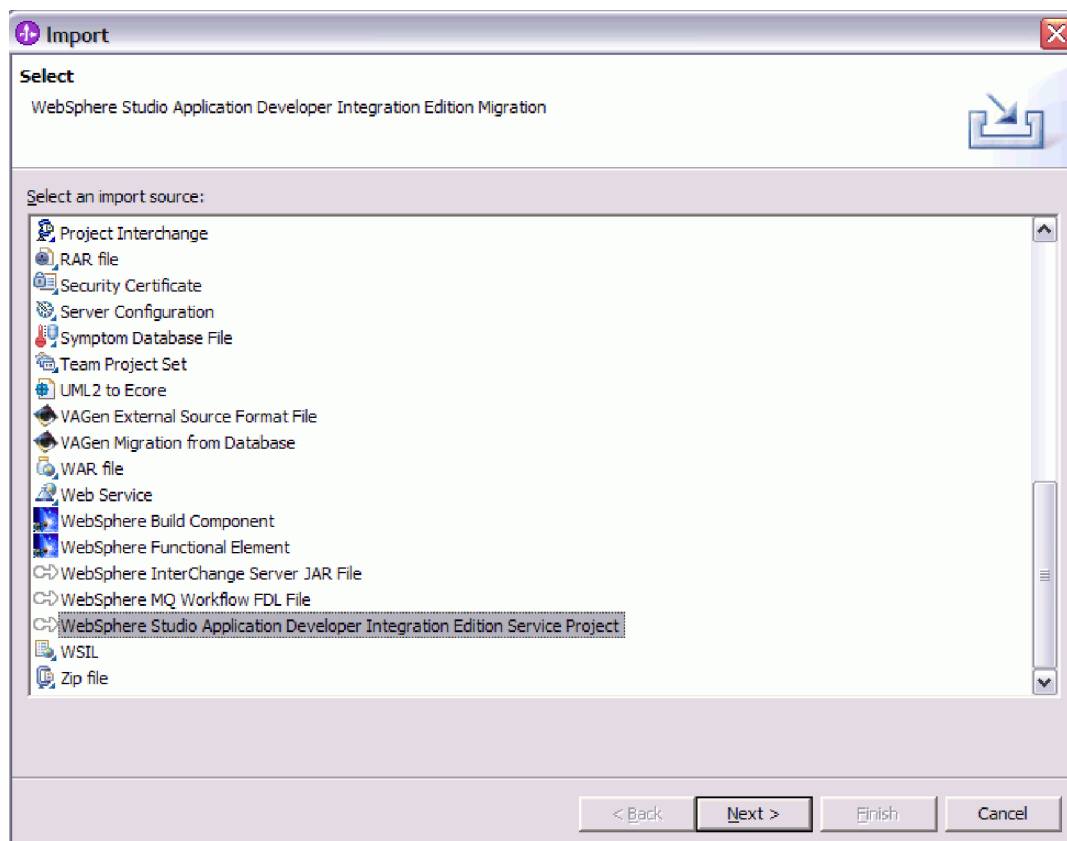
WebSphere Integration Developer マイグレーション・ウィザードを使用して、サービス・プロジェクトをマイグレーションできます。

マイグレーション・ウィザードは、以下を行います。

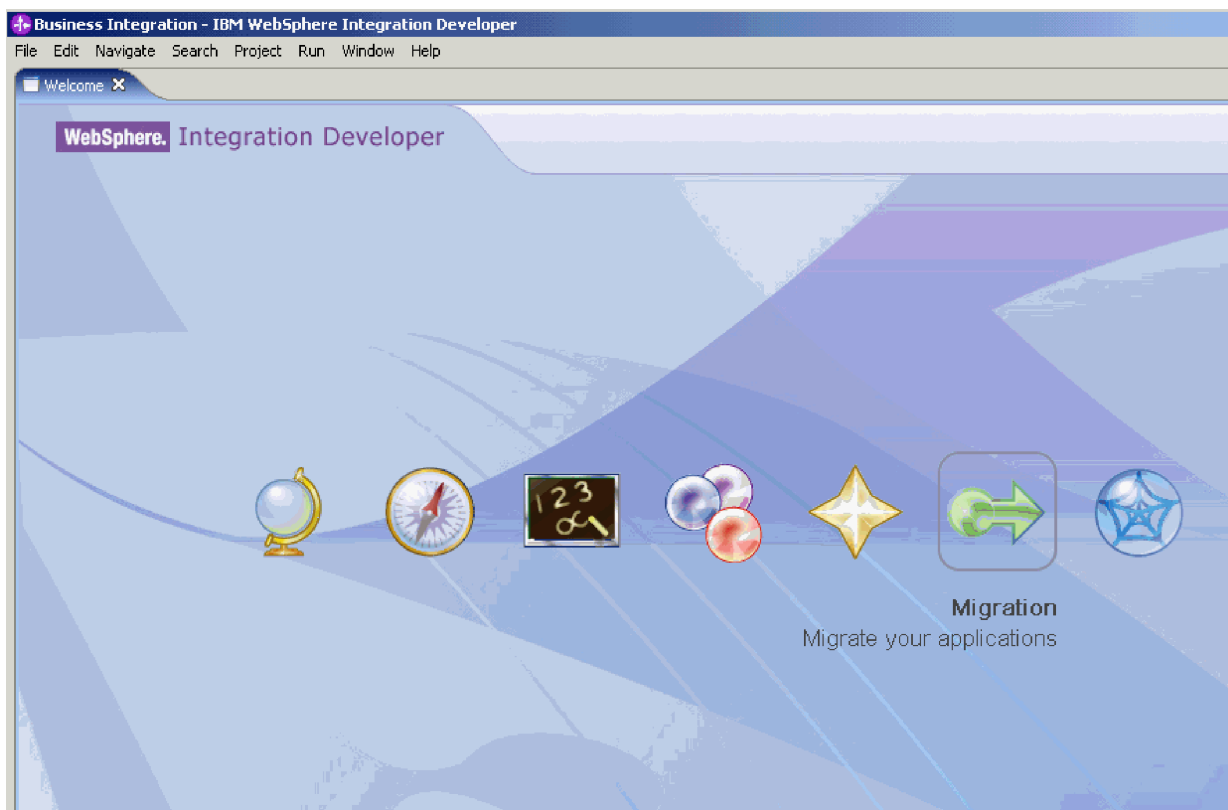
1. 新規ビジネス・インテグレーション・モジュールを作成します (モジュール名は定義済みです)。
2. サービス・プロジェクトのクラスパス・エントリーを新しいモジュールにマイグレーションします。
3. すべての WebSphere Business Integration Server Foundation ソース成果物を、選択されているソース・プロジェクトからこのモジュールにコピーします。
4. WSDL ファイル内の BPEL 拡張機能をマイグレーションします。
5. ビジネス・プロセス (.bpel ファイル) を BPEL4WS バージョン 1.1 から WebSphere Process Server でサポートされている新しいレベルにマイグレーションします。これは、今度の WS-BPEL バージョン 2.0 仕様の主要な機能を備えた BPEL4WS バージョン 1.1 に組み込まれています。
6. .bpel プロセスごとに SCA コンポーネントを作成します。
7. WebSphere Studio Application Developer Integration Edition からデフォルト・モニター動作を保存するために BPEL プロセスごとにモニター .mon ファイルを生成します (必要な場合)。

WebSphere Integration Developer マイグレーション・ウィザードを使用してサービス・プロジェクトをマイグレーションするには、以下のステップに従います。

1. 「ファイル」 → 「インポート」 → 「WebSphere Studio Application Developer Integration Edition サービス・プロジェクト」を選択してウィザードを起動します。



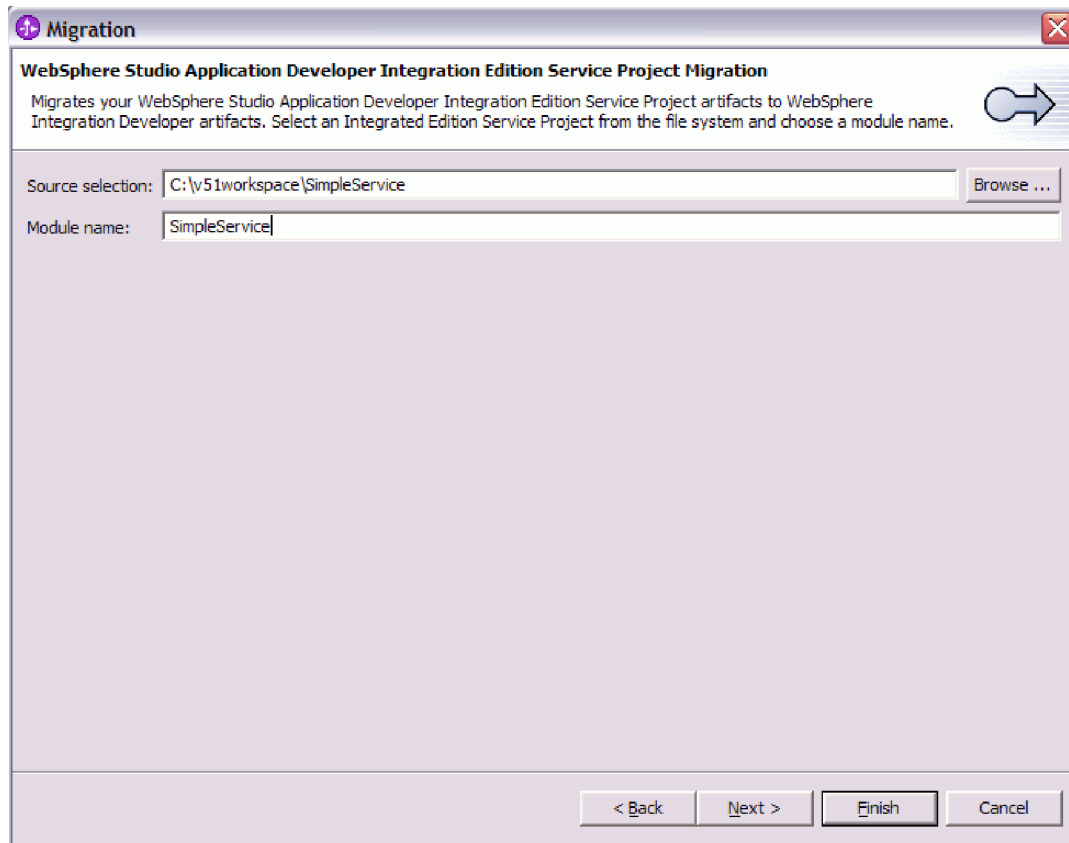
をクリックして、「ようこそ」ページからマイグレーション・ウィザードを開くこともできます。



「マイグレーション」ページから、「Integration Edition 5.1 サービス・プロジェクトをマイグレーションする」オプションを選択します。

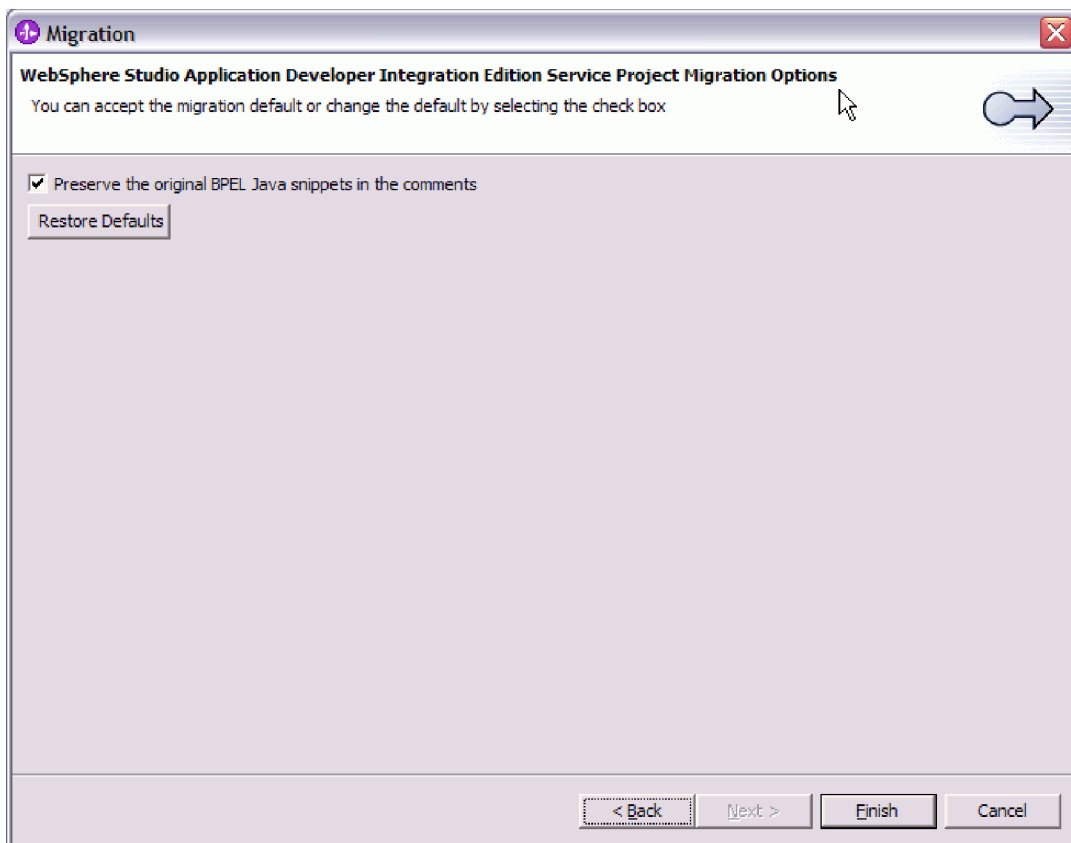


2. マイグレーション・ウィザードがオープンします。「ソース選択 (Source Selection)」にパスを入力するか、または「参照」ボタンをクリックしてパスを見つけます。また、マイグレーションする WebSphere Studio Application Developer Integration Edition サービス・プロジェクトのロケーションのモジュール名を入力します。



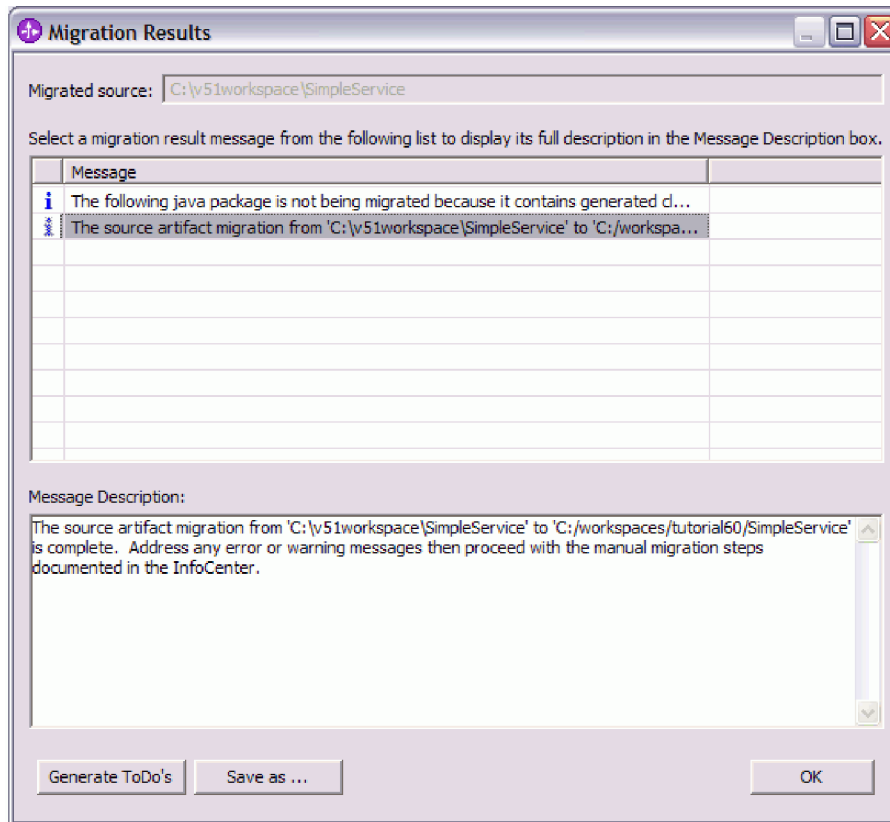
注: サービス・プロジェクトの名前にはモジュール名を選択することをお勧めします。これは、WebSphere Studio Application Developer Integration Edition ワークスペース内にこのプロジェクトに従属する他のプロジェクトがある場合、これらの従属プロジェクトを WebSphere Integration Developer にインポートした後に、従属プロジェクトのクラスパスを更新する必要がなくなるためです。

3. 「マイグレーション」オプションから、「オリジナルの BPEL Java Snippet をコメントに保存する (Preserve original BPEL Java snippets in the comments)」チェック・ボックスを選択します。



「終了」をクリックします。

4. マイグレーション・プロセスが完了すると、「マイグレーション結果」ウィンドウが開きます。



これらのマイグレーション・メッセージを含むログ・ファイルが、6.0 のワークスペースの .metadata フォルダに自動的に生成されます。ログ・ファイルの名前は「.log」になります。

マイグレーション・ウィザードの完了後、作成されたビジネス・インテグレーション・モジュールをビルドし、ビルド・エラーがあれば解決を試みます。すべてのマイグレーション済み .bpel ファイルを検査します。ファイルが完全にマイグレーションされていて、WebSphere Integration Developer BPEL エディターで開くことができることを確認します。BPEL Java Snippet には、自動的にマイグレーションできないものもあります。BPEL Java Snippet にエラーを見つけた場合は、エラーの修正に必要なステップについて、『SCA プログラミング・モデルへのマイグレーション』を参照してください。また、マイグレーション・ウィザードを使用してサービス・プロジェクトを BI モジュールにマイグレーションする場合、モジュール依存関係エディターを開き、依存関係が正しく設定されていることを確認します。これを行うには、ビジネス・インテグレーション・パースペクティブに切り替えて、「BI モジュール・プロジェクト (BI module project)」をダブルクリックします。ここでビジネス・インテグレーション・ライブラリー・プロジェクト、Java プロジェクト、および J2EE プロジェクトへの依存関係を追加できます。

アプリケーションの手動マイグレーション

マイグレーション・ウィザードで成果物を新しいビジネス・インテグレーション・モジュールに正常にマイグレーションした後で、成果物をワイヤリングして SCA モデルに準拠するアプリケーションを作成する必要があります。

1. WebSphere Integration Developer を開き、ビジネス・インテグレーション・パースペクティブに切り替えます。マイグレーション・ウィザードによって作成されたモジュール (マイグレーションされたサービス・プロジェクトごとに 1 つのモジュール) が表示されます。モジュール・プロジェクトの下にリストされた最初の成果物は、モジュールのアセンブリー・ファイルです (これはモジュールと同じ名前を持ちます)。

2. アセンブリ・ファイルをダブルクリックしてアセンブリ・エディターで開きます。このエディターで SCA コンポーネントを作成およびワイヤリングすれば、バージョン 5.1 アプリケーションと同様の機能を得ることができます。WebSphere Studio Application Developer Integration Edition サービス・プロジェクトに BPEL プロセスがあれば、マイグレーション・ウィザードはこのプロセスごとにデフォルト SCA コンポーネントを作成しているので、そのコンポーネントがアセンブリ・エディターに表示されます。
3. コンポーネントを選択し、「プロパティ」ビューに移動します。このビューでは、「説明」、「詳細」、および「実装」プロパティが表示され、それらを編集することができます。

次の情報では、WebSphere Integration Developer に用意されたツールを使用してアプリケーションを手動で再ワイヤリングする方法について詳しく説明します。

再ワイヤリングのためにアプリケーション内のサービス用 SCA コンポーネントおよび SCA インポートを作成:

すべてのプロジェクトで、マイグレーションの後、5.1 で接続されていたとおりにサービスを再接続するために、ある程度の再ワイヤリングが必要です。例えば、マイグレーションされたすべてのビジネス・プロセスを、そのビジネス・パートナーに再ワイヤリングする必要があります。その他のすべてのサービス・タイプ用に、SCA コンポーネントまたはインポートを作成する必要があります。プロジェクトの外部にあるシステムまたはエンティティと対話する WebSphere Studio Application Developer Integration Edition サービス・プロジェクトでは、マイグレーションされたプロジェクトが、SCA モデルに従ってこれらのエンティティにサービスとしてアクセスするために、SCA インポートを作成することができます。

プロジェクト内のエンティティ (例えば、ビジネス・プロセス、変換プログラム・サービス、または Java クラス) と対話する WebSphere Studio Application Developer Integration Edition サービス・プロジェクトでは、マイグレーションされたプロジェクトが、SCA モデルに従ってこれらのエンティティにサービスとしてアクセスするために、SCA インポートを作成することができます。

以下のセクションでは、マイグレーションする必要があるサービスのタイプに基づいて作成する SCA インポートまたは SCA コンポーネントの詳細を説明します。

Java サービスのマイグレーション:

Java サービスを SCA Java コンポーネントにマイグレーションすることができます。

WebSphere Studio Application Developer Integration Edition サービス・プロジェクトが他の Java プロジェクトに付属していた場合、既存のプロジェクトを新しいワークスペース・ディレクトリーにコピーして、ウィザード (「ファイル」 → 「インポート」 → 「既存プロジェクトをワークスペースへ」) を使用してそれらを WebSphere Integration Developer にインポートします。

WebSphere Studio Application Developer Integration Edition では、既存の Java クラスから新規の Java サービスを生成するときに、以下のオプションが提供されます。

- 複素数データ型の XSD スキーマを作成する
 - インターフェース WSDL ファイル内に
 - データ型ごとに新規ファイルとして
- エラー処理機能をサポートする
 - フォールトを生成する
 - フォールトを生成しない
- バインディングおよびサービス名などを生成するサービスに関する他の詳細

6.0 には、データ・マッピング、インターフェース・メディエーション、ビジネス・ステート・マシン、セクター、ビジネス・ルールなど、新しい機能を提供する多くの新規コンポーネントがあります。まず最初に、これらの新規コンポーネントのいずれかでカスタム Java コンポーネントを置き換えることができるかどうかを判断します。それが不可能な場合は、下記のマイグレーション・パスに従ってください。

マイグレーション・ウィザードを使用してサービス・プロジェクトをインポートします。この結果、ビジネス・インテグレーション・モジュールが作成され、WSDL メッセージ、ポート・タイプ、バインディング、およびサービスが WebSphere Studio Application Developer Integration Edition に生成されます。

ビジネス・インテグレーション・パースペクティブでモジュールを展開し、内容を参照します。モジュール・プロジェクトの下にある最初の項目をダブルクリックしてアセンブリ・エディターを開きます (これはプロジェクトと同じ名前になります)。

以下のオプションがあります。

各 Java サービス再ワイヤリング・オプションの利点と欠点:

Java サービス再ワイヤリング・オプションにはそれぞれ、利点と欠点があります。

以下のリストで、両方のオプションと、それぞれの利点および欠点について説明します。

- 最初のオプションでは、Web サービスが Java コンポーネントより遅い速度で呼び出されるため、実行時のパフォーマンスが向上する可能性があります。
- 最初のオプションはコンテキストを伝搬することができますが、Web サービスの呼び出しでは同じようにコンテキストを伝搬することはできません。
- 2 番目のオプションには、カスタム・コードの作成が含まれません。
- Java サービスの生成には制限があるため、2 番目のオプションは一部の Java インターフェース定義には使用可能でない場合があります。Rational Application Developer のドキュメンテーション (<http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.doc/ref/rlimit.html>) を参照してください。
- 2 番目のオプションでは、結果としてインターフェースが変更され、そのために SCA コンシューマーにも変更が生じる可能性があります。
- 2 番目のオプションでは、WebSphere Process Server 6.0 サーバーがインストールされ、WebSphere Integration Developer と連動するように構成されている必要があります。WebSphere Integration Developer と連動するように構成されているインストール済みランタイムを調べるには、「ウィンドウ (Window)」 → 「設定 (Preferences)」 → 「サーバー」 → 「インストール済みランタイム」に進み、「WebSphere Process Server v6.0」エントリーがあればこれを選択し、製品がインストールされているロケーションを指していることを確認します。サーバーが存在する場合にはこのエントリーにチェックマークが付けられ、このサーバーが実際にはインストールされていない場合にはチェックマークが外されていることを確認してください。「追加...」ボタンをクリックして、別のサーバーを追加することもできます。
- Java コンポーネントが、WSDL から Java スケルトンを生成するトップダウンのアプローチを使用して WebSphere Studio Application Developer Integration Edition に組み込まれている場合は、この Java クラス内外のパラメーターはおそらく WSIFFormatPartImpl をサブクラス化します。このような場合は、オプション 1 を選択してオリジナルの WSDL/XSD から新しい SCA スタイルの Java スケルトンを生成するか、またはオプション 2 を選択してオリジナルの WSDL インターフェースから (WSIF または DataObject API に依存しない) 新しい汎用 Java スケルトンを生成してください。

カスタム Java コンポーネントの作成: オプション 1:

推奨されるマイグレーション手法は、Java サービスを 1 つの SCA コンポーネントとして表せるようにする、WebSphere Integration Developer Java コンポーネント・タイプを使用するものです。マイグレーション時に、SCA Java インターフェース・スタイルと既存の Java コンポーネントのインターフェース・スタイルの間で変換されるように、カスタム Java コードが作成されている必要があります。

カスタム Java コンポーネントを作成するには、以下のようにします。

1. モジュール・プロジェクトで、「**インターフェース**」を展開して、WebSphere Studio Application Developer Integration 内のこの Java クラス用に生成された WSDL インターフェースを選択します。
2. このインターフェースをアセンブリ・エディターにドラッグ・アンド・ドロップします。作成するコンポーネントのタイプを選択するように求めるダイアログがポップアップ表示されます。「**実装タイプのないコンポーネント**」を選択して、「**OK**」をクリックします。
3. アセンブリ図に、汎用コンポーネントが表示されます。そのコンポーネントを選択して、「**プロパティ**」ビューに進みます。
4. 「**記述**」タブで、コンポーネントの名前を変更して、より分かりやすい名前で表示することができます。
5. 「**詳細**」タブで、このコンポーネントに 1 つのインターフェース (アセンブリ・エディターにドラッグ・アンド・ドロップしたインターフェース) があることが分かります。
6. アクセスを試みている Java クラスがサービス・プロジェクト自体の中に含まれていない場合は、そのクラスがサービス・プロジェクトのクラスパス上にあることを確認します。
7. モジュール・プロジェクトを右クリックして、「**依存関係エディターを開く...**」を選択します。「**Java**」セクションで、古い Java クラスが含まれているプロジェクトがリストされていることを確認します。リストされていない場合は、「**追加...**」ボタンをクリックして追加してください。
8. アセンブリ・エディターに戻り、作成したばかりのコンポーネントを右クリックして、「**実装の生成...**」→「**Java**」を選択します。Java の実装が生成されるパッケージを選択します。これにより、SCA プログラミング・モデル (複合タイプは `commonj.sdo.DataObject` であるオブジェクトによって表され、単純タイプはそれと同等の Java オブジェクトによって表される) に準拠した WSDL インターフェースに従ったスケルトン Java サービスが作成されます。

下記のコードの例は、以下を示しています。

1. 5.1 の WSDL インターフェースからの関連する定義
2. WSDL に対応する WebSphere Studio Application Developer Integration Edition 5.1 Java メソッド
3. 同じ WSDL 用の WebSphere Integration Developer 6.0 Java メソッド

以下のコードは、5.1 の WSDL インターフェースからの関連する定義を示しています。

```
<types>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
attributeFormDefault="qualified"
elementFormDefault="unqualified"
targetNamespace="http://migr.practice.ibm.com/"
xmlns:xsd="http://migr.practice.ibm.com/">
<complexType name="StockInfo">
<all>
<element name="index" type="int"/>
<element name="price" type="double"/>
<element name="symbol" nillable="true"
type="string"/>
</all>
</complexType>
</schema>
</types>
<message name="getStockInfoRequest">
```

```

<part name="symbol" type="xsd:string"/>
</message>
<message name="getStockInfoResponse">
<part name="result" type="xsd:StockInfo"/>
</message>
<operation name="getStockInfo" parameterOrder="symbol">
<input message="tns:getStockInfoRequest"
name="getStockInfoRequest"/>
<output message="tns:getStockInfoResponse"
name="getStockInfoResponse"/>
</operation>

```

以下のコードは、WSDL に対応する WebSphere Studio Application Developer Integration Edition 5.1 Java メソッドを示しています。

```

public StockInfo getStockInfo(String symbol)
{
    return new StockInfo();
}
public void setStockPrice(String symbol, float newPrice)
{
    // set some things
}

```

以下のコードは、同じ WSDL 用の WebSphere Integration Developer 6.0 Java メソッドを示しています。

```

public DataObject getStockInfo(String aString) {
//TODO Needs to be implemented.
return null;
}
public void setStockPrice(String symbol, Float newPrice) {
//TODO Needs to be implemented.
}

```

ここで、生成された Java 実装クラスの中の「//TODO」タグで示されている部分に、コードを入力する必要があります。次の 2 つのオプションがあります。

1. ロジックをオリジナルの Java クラスからこのクラスに移動し、DataObject を使用するように適合させます。
 - WebSphere Studio Application Developer Integration Edition でトップダウン・アプローチを選択しており、Java コンポーネントに DataObject パラメーターを扱わせたい場合には、これが推奨されるオプションです。WebSphere Studio Application Developer Integration Edition の WSDL 定義から生成された Java クラスが、除去する必要がある WSIF 依存関係を持っているため、この再作業が必要になります。
2. この生成済み Java クラスの内側に、古い Java クラスの専用インスタンスを作成し、以下のことを行うためのコードを作成します。
 - a. 生成された Java 実装クラスのすべてのパラメーターを、古い Java クラスが期待するパラメーターに変換する。
 - b. 変換されたパラメーターを使用して、古い Java クラスの専用インスタンスを呼び出す。
 - c. 古い Java クラスの戻り値を、生成された Java 実装メソッドで宣言されている戻り値の型に変換する。
 - d. このオプションは、新しい 6.0 スタイルの Java コンポーネントが WSIF サービス・プロキシを利用する必要がある使用量シナリオで、推奨されます。

上記のオプションの 1 つが完了したら、Java サービスを再ワイヤリングする必要があります。参照は存在しないはずであるため、必要なのは Java コンポーネントのインターフェースの再ワイヤリングのみです。

- このサービスが同じモジュール内のビジネス・プロセスによって呼び出される場合は、該当するビジネス・プロセス参照からこの Java コンポーネントのインターフェースへのワイヤーを作成します。
- このサービスが別のモジュール内のビジネス・プロセスによって呼び出される場合は、**SCA バインディング付きエクスポート**を作成し、このエクスポートを他のモジュールからそのモジュールのアセンブリ・エディターにドラッグ・アンド・ドロップして、対応する **SCA バインディング付きインポート**を作成します。該当するビジネス・プロセス参照をそのインポートにワイヤリングします。
- このサービスが、外部での公開のために WebSphere Studio Application Developer Integration Edition で公開されていた場合、サービスの再公開の方法については、セクション『マイグレーション済みサービスにアクセスするために SCA エクスポートを作成』を参照してください。

Java Web サービスの作成: オプション 2:

考慮すべき代替オプションは、Java クラスに Web サービスを作成できる、Rational Application Developer Web サービス・ツールです。

注: この方法を使用してマイグレーションを試みる前に、サイト <http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.was.creation.ui.doc/tasks/twsbeanw.html> の情報を参照してください。

注: このオプションでは、Web サービス・ウィザードを呼び出す前に、WebSphere Integration Developer を介して Web サービス・ランタイムが構成されていることが必要です。

WebSphere Studio Application Developer Integration Edition でボトムアップのアプローチを使用して Java クラスに WSDL を生成する場合は、以下のステップを実行してください。

1. 新しい Web プロジェクトを作成し、この Web プロジェクトの Java ソース・フォルダーに、サービスをビルドする Java クラスをコピーします。
2. サービスを作成している Java クラスのコンテナであるエンタープライズ・アプリケーション・プロジェクトを右クリックします。
3. 「プロパティ」を選択し、「サーバー」プロパティで、「ターゲット・ランタイム (Target runtime)」が、「**WebSphere Process Server v6.0**」に設定され、「デフォルト・サーバー」が、インストール済みの「**WebSphere Process Server v6.0**」に設定されていることを確認します。
4. テスト・サーバーを開始し、そのサーバーにこのアプリケーションをデプロイして、正常に開始されるようにします。
5. 次に、サービスを作成したい Java クラスを右クリックして、「**Web サービス**」→「**Web サービスの作成**」を選択します。
6. 「**Web サービス・タイプ (Web Service Type)**」について、「**Java Bean Web サービス (Java bean Web Service)**」を選択し、Web サービスを今すぐにデプロイしたい場合を除き、「**Web プロジェクトの Web サービスを開始**」オプションのチェック・マークを外します。オプションで、クライアント・プロキシの生成も選択できます。「**次へ**」をクリックします。
7. 右クリックした Java クラスが表示されるので、「**次へ**」をクリックします。
8. ここで、サービス・デプロイメント・オプションを構成する必要があります。「**編集...**」ボタンをクリックします。サーバー・タイプには、「**WPS Server v6.0**」を選択し、Web サービス・ランタイムには、「**IBM WebSphere**」および「**J2EE バージョン 1.4 (J2EE version 1.4)**」を選択します。これを行うことで有効な組み合わせを選択できない場合は、v1.4 レベルへの J2EE プロジェクトのマイグレーションについて、セクション『マイグレーションの準備』を参照してください。「**OK**」をクリックします。

9. サービス・プロジェクトについて、Web プロジェクトの名前を入力します。また、該当する EAR プロジェクトも選択します。「次へ」をクリックします。このとき、数分間待たなければならない場合があります。
10. 「Web サービス Java Bean の識別」パネルで、WSDL 定義を入れる WSDL ファイルを選択します。Web サービスで公開したいメソッドを選択し、該当するスタイル/エンコード (文書/リテラル、RPC/リテラル、または RPC/エンコード) を選択します。「パッケージから名前空間へのカスタム・マッピングを定義する」オプションを選択し、この Java クラスのインターフェースで使用するすべての Java パッケージについて、マイグレーションされる Java クラス内で固有な名前空間を選択します (デフォルトの名前空間は、パッケージ名が固有ですが、同じ Java クラスを使用する別の Web サービスを作成する場合に、競合が発生する可能性があります)。適宜、その他のパラメーターを入力します。
11. 「次へ」をクリックし、「Web サービスのパッケージから名前空間へのマッピング」パネルで、「追加」ボタンをクリックし、作成される行に Java Bean のパッケージ名を入力してから、この Java クラスを一意的に識別するカスタム名前空間を追加します。Java Bean インターフェースで使用するすべての Java パッケージに対して、マッピングの追加を続行します。
12. 「次へ」をクリックします。このとき、数分間待たなければならない場合があります。
13. 「終了」をクリックします。サービス・プロジェクトが Java サービスのコンシューマーであった場合には、ウィザードが完了した後、Java サービスを記述する生成済み WSDL ファイルをビジネス・インテグレーション・モジュール・プロジェクトにコピーする必要があります。このプロジェクトは、フォルダー WebContent/WEB-INF/wsdl の下の生成済みルーター Web プロジェクトの中にあります。ビジネス・インテグレーション・モジュール・プロジェクトの更新/再ビルドを行ってください。
14. ビジネス・インテグレーション・パースペクティブに切り替え、モジュールを展開して、次に「Web サービス・ポート」論理カテゴリーを展開します。
15. 前のステップで作成されたポートを選択し、そのポートをアセンブリ・エディターにドラッグ・アンド・ドロップして、**Web サービス・バインディング付きインポート**の作成を選択します。プロンプトが出されたら、Java クラスの WSDL インターフェースを選択します。これで、5.1 で Java コンポーネントを利用する SCA コンポーネントをこのインポートにワイヤリングして、手動による再ワイヤリングのマイグレーション・ステップを完了することができます。

このインターフェースは 5.1 のインターフェースと多少の違いがあり、5.1 コンシューマーと新しいインポートの間にインターフェース・メディエーション・コンポーネントを挿入することが必要な場合があります。これを行うためには、アセンブリ・エディターでワイヤー・ツールをクリックして、SCA ソース・コンポーネントを、この新しい **Web サービス・バインディング付きインポート**にワイヤリングします。インターフェースが異なるため、「ソース・ノードとターゲット・ノードにマッチング・インターフェースがありません。」というプロンプトが出されます。ソース・ノードとターゲット・ノードの間に**インターフェース・マッピングを作成すること**を選択します。アセンブリ・エディターで作成されたマッピング・コンポーネントをダブルクリックします。これにより、マッピング・エディターが開きます。インターフェース・マッピングの作成についての説明は、インフォメーション・センターを参照してください。

WebSphere Studio Application Developer Integration Edition でトップダウンのアプローチを使用していた場合は、WSDL 定義から Java クラスを生成した後、以下のステップを実行してください。

1. 新しい Web プロジェクトを作成し、Java スケルトンにしたい WSDL ファイルを、この Web プロジェクトのソース・フォルダーにコピーします。
2. Java スケルトンの生成元にしたいポート・タイプが含まれている WSDL ファイルを右クリックして、「**Web サービス**」→「**Java bean スケルトンを生成 (Generate Java bean skeleton)**」を選択します。
3. Web サービスのタイプ「**スケルトン Java bean Web サービス (Skeleton Java bean Web Service)**」を選択して、ウィザードを完了します。

ウィザードを完了した後は、サービス・インターフェースを実装する Java クラスが作成され、それは WSIF API に依存したものではないはずです。

EJB サービスのマイグレーション:

EJB サービスをステートレス・セッション Bean バインディング付き SCA インポートにマイグレーションできます。

WebSphere Studio Application Developer Integration Edition サービス・プロジェクトが別の EJB、EJB クライアント、または Java プロジェクトに依存する場合、「ファイル」→「インポート」→「既存プロジェクトをワークスペースへ」ウィザードを使用して、これら既存のプロジェクトをインポートします。これは、EJB がサービス・プロジェクトから参照される場合の通常のケースでした。サービス・プロジェクトから参照される WSDL または XSD ファイルが別のタイプのプロジェクトに存在する場合、古い非サービス・プロジェクトと同じ名前を使用して新しいビジネス・インテグレーション・ライブラリーを作成し、成果物をすべてこのライブラリーにコピーします。

マイグレーション・ウィザードを使用してサービス・プロジェクトをインポートします。この結果、ビジネス・インテグレーション・モジュールが作成され、WSDL メッセージ、ポート・タイプ、バインディング、およびサービスが WebSphere Studio Application Developer Integration Edition に生成されます。

ビジネス・インテグレーション・パースペクティブでモジュールを展開し、内容を参照します。モジュール・プロジェクトの下にある最初の項目をダブルクリックしてアセンブリ・エディターを開きます (これはプロジェクトと同じ名前になります)。

以下のオプションがあります。

各 EJB サービス再ワイヤリング・オプションの利点と欠点:

EJB サービス再ワイヤリング・オプションにはそれぞれ、利点と欠点があります。

以下のリストで、両方のオプションと、それぞれの利点および欠点について説明します。

- 最初のオプションでは、Web サービスが EJB より遅い速度で呼び出されるため、実行時のパフォーマンスが向上する可能性があります。
- 最初のオプションはコンテキストを伝搬することができますが、Web サービスの呼び出しでは同じようにコンテキストを伝搬することはできません。
- 2 番目のオプションには、カスタム・コードの作成が含まれません。
- EJB サービスの生成には制限があるため、2 番目のオプションは一部の EJB インターフェース定義には使用可能でない場合があります。Rational Application Developer のドキュメンテーション (<http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.doc/ref/rlimit.html>) を参照してください。
- 2 番目のオプションでは、結果としてインターフェースが変更され、そのために SCA コンシューマーにも変更が生じる可能性があります。
- 2 番目のオプションでは、WebSphere Process Server 6.0 サーバーがインストールされ、WebSphere Integration Developer と連動するように構成されている必要があります。WebSphere Integration Developer と連動するように構成されているインストール済みランタイムを調べるには、「ウィンドウ (Window)」→「設定 (Preferences)」→「サーバー」→「インストール済みランタイム」に進み、「WebSphere Process Server v6.0」エントリーがあればこれを選択し、製品がインストールされているロケーションを指していることを確認します。サーバーが存在する場合にはこのエントリーにチェック

マークが付けられ、このサーバーが実際にはインストールされていない場合にはチェックマークが外されていることを確認してください。「追加...」ボタンをクリックして、別のサーバーを追加することもできます。

- Java コンポーネントが、WSDL から EJB スケルトンを生成するトップダウンのアプローチを使用して WebSphere Studio Application Developer Integration Edition に組み込まれている場合は、この Java クラス内外のパラメーターはおそらく WSIFFormatPartImpl をサブクラス化します。このような場合は、オプション 2 を選択してオリジナルの WSDL インターフェースから (WSIF または DataObject API に依存しない) 新しい汎用 EJB スケルトンを生成してください。

カスタム EJB コンポーネントの作成: オプション 1:

推奨されるマイグレーション手法は、SCA コンポーネントとしてステートレス・セッション EJB を呼び出すことができるようにする、ステートレス・セッション・バインディング・タイプの WebSphere Integration Developer Import を使用することです。マイグレーション時に、SCA Java インターフェースのスタイルと既存の EJB インターフェースのスタイルの間で変換されるように、カスタムの Java コードが作成されている必要があります。

カスタム EJB コンポーネントを作成するには、以下を行います。

1. モジュール・プロジェクトの下で、「インターフェース」を展開して、WebSphere Studio Application Developer Integration 内でこの EJB 用に生成された WSDL インターフェースを選択します。
2. このインターフェースをアセンブリ・エディターにドラッグ・アンド・ドロップします。作成するコンポーネントのタイプを選択するように求めるダイアログがポップアップ表示されます。「実装タイプのないコンポーネント」を選択して、「OK」をクリックします。
3. アセンブリ図に、汎用コンポーネントが表示されます。そのコンポーネントを選択して、「プロパティ」ビューに進みます。
4. 「記述」タブで、コンポーネントの名前を変更して、より分かりやすい名前が表示することができます。EJB の名前に類似した名前を選択してください。ただし、このコンポーネントは WebSphere Studio Application Developer Integration の EJB 用に生成された WSDL インターフェースと、その EJB の Java インターフェースとの間を仲介する Java コンポーネントになるため、その名前に「JavaMed」などの接尾部を付加します。
5. 「詳細」タブで、このコンポーネントに 1 つのインターフェース (アセンブリ・エディターにドラッグ・アンド・ドロップしたインターフェース) があることが分かります。
6. アセンブリ・エディターに戻り、作成したばかりのコンポーネントを右クリックして、「実装の生成...」→「Java」を選択します。Java の実装が生成されるパッケージを選択します。これにより、SCA プログラミング・モデル (複合タイプは commonj.sdo.DataObject であるオブジェクトによって表され、単純タイプはそれと同等の Java オブジェクトによって表される) に準拠した WSDL インターフェースに従ったスケルトン Java サービスが作成されます。

下記のコードの例は、以下を示しています。

1. 5.1 の WSDL インターフェースからの関連する定義
2. WSDL に対応する WebSphere Studio Application Developer Integration Edition 5.1 Java メソッド
3. 同じ WSDL 用の WebSphere Integration Developer 6.0 Java メソッド

以下のコードは、5.1 の WSDL インターフェースからの関連する定義を示しています。

```
<types>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
attributeFormDefault="qualified"
elementFormDefault="unqualified"
```

```

targetNamespace="http://migr.practice.ibm.com/"
xmlns:xsd="http://migr.practice.ibm.com/"
<complexType name="StockInfo">
<all>
<element name="index" type="int"/>
<element name="price" type="double"/>
<element name="symbol" nillable="true"
type="string"/>
</all>
</complexType>
</schema>
</types>
<message name="getStockInfoRequest">
<part name="symbol" type="xsd:string"/>
</message>
<message name="getStockInfoResponse">
<part name="result" type="xsd:StockInfo"/>
</message>
<operation name="getStockInfo" parameterOrder="symbol">
<input message="tns:getStockInfoRequest"
name="getStockInfoRequest"/>
<output message="tns:getStockInfoResponse"
name="getStockInfoResponse"/>
</operation>

```

以下のコードは、WSDL に対応する WebSphere Studio Application Developer Integration Edition 5.1 Java メソッドを示しています。

```

public StockInfo getStockInfo(String symbol)
{
return new StockInfo();
}
public void setStockPrice(String symbol, float newPrice)
{
// set some things
}

```

以下のコードは、同じ WSDL 用の WebSphere Integration Developer 6.0 Java メソッドを示しています。

```

public DataObject getStockInfo(String aString) {
//TODO Needs to be implemented.
return null;
}
public void setStockPrice(String symbol, Float newPrice) {
//TODO Needs to be implemented.
}

```

生成された Java 実装クラスの中の「//TODO」タグで示されている部分には、最終的に実際のコードを入力する必要があります。まず最初に、この Java コンポーネントから実際の EJB への参照を作成して、これが SCA プログラミング・モデルに従って EJB にアクセスできるようにする必要があります。

1. アセンブリ・エディターを開いたままにして、J2EE パースペクティブに切り替えます。サービスの作成対象の EJB が含まれている EJB プロジェクトを見つけます。
2. その「デプロイメント記述子: <project-name> (Deployment Descriptor: <project-name>)」項目を展開して、EJB を見つけます。これをアセンブリ・エディターにドラッグ・アンド・ドロップします。更新を必要としているプロジェクト依存関係に関する警告が出された場合は、「モジュール依存関係エディターを開き、... (Open the module dependency editor...)」チェック・ボックスを選択して、「OK」をクリックします。
3. J2EE セクションで、その EJB プロジェクトがリストされていることを確認し、リストされていない場合は、「追加...」ボタンをクリックして追加します。

- モジュール依存関係を保管し、そのエディターを閉じます。新しいインポートがアセンブリ・エディターに作成されていることが分かります。そのインポートを選択し、「記述」タブの「プロパティ」ビューに進み、そのインポートの名前を変更して、より分かりやすい名前で表示できます。「バインディング」タブでは、インポート・タイプが自動的に「ステートレス・セッション Bean バインディング」に設定され、すでに EJB の JNDI 名が適切に設定されていることが分かります。
- アセンブリ・エディターのパレットから、ワイヤー・ツールを選択します。
- Java コンポーネントをクリックして、マウスを放します。
- 次に、「EJB インポート」をクリックして、マウスを放します。
- 「ソース・ノード上にマッチング参照が作成されます。継続しますか?」が表示されたら、「OK」をクリックします。これにより、2 つのコンポーネント間にワイヤーが作成されます。
- アセンブリ・エディター内および「詳細」タブの「プロパティ」ビューで Java コンポーネントを選択し、「参照」を展開して、作成したばかりの EJB への参照を選択します。生成された名前があまり分かりやすくないか、適切でない場合は、参照の名前を更新できます。この参照の名前は、将来の利用のために覚えておいてください。
- アセンブリ図を保管します。

生成された Java クラスから EJB を呼び出すためには、SCA プログラミング・モデルを使用する必要があります。生成された Java クラスを開き、以下のステップに従って、EJB サービスを呼び出すコードを作成します。生成された Java 実装クラスについて、以下のようにします。

- 次の `private` 変数 (タイプがリモート EJB インターフェースのもの) を作成します。

```
private YourEJBInterface ejbService = null;
```

- EJB インターフェースに複合タイプがある場合は、BOFactory 用の `private` 変数も作成します。

```
private BOFactory boFactory = (BOFactory)
ServiceManager.INSTANCE.locateService("com/ibm/websphere/bo
/BOFactory");
```

- Java 実装クラスのコンストラクターで、SCA API を使用して、EJB 参照を解決し (前のステップで作成した EJB 参照の名前を入力してください)、この参照と等しい `private` 変数を設定します。

```
// Locate the EJB service
this.ejbService = (YourEJBInterface)
ServiceManager.INSTANCE.locateService("name-of-your-ejb-reference");
```

生成された Java 実装クラスの中にあるそれぞれの「//TODO」について、以下のようにします。

- すべてのパラメーターを、EJB が期待するパラメーター・タイプに変換します。
- SCA プログラミング・モデルを使用して、EJB 参照上の該当メソッドを呼び出し、変換されたパラメーターを送信します。
- EJB の戻り値を、生成された Java 実装メソッドによって宣言された戻り値の型に変換します。

```
/**
 * Method generated to support the implementing WSDL port type named
 * "interface.MyBean".
 */
public BusObjImpl getStockInfo(String aString) {
    BusObjImpl boImpl = null;
    try {
        // invoke the EJB method
        StockInfo stockInfo = this.ejbService.getStockInfo(aString);
        // formulate the SCA data object to return.
        boImpl = (BusObjImpl)
        this.boFactory.createClass(StockInfo.class);
        // manually convert all data from the EJB return type into the
        // SCA data object to return
        boImpl.setInt("index", stockInfo.getIndex());
    }
}
```



```

boImpl.setString("symbol", stockInfo.getSymbol());
boImpl.setDouble("price", stockInfo.getPrice());
} catch (RemoteException e) {
e.printStackTrace();
}
return boImpl;
}
/**
 * Method generated to support the implementing WSDL port type named
 * "interface.MyBean".
 */
public void setStockPrice(String symbol, Float newPrice) {
try {
this.ejbService.setStockPrice(symbol, newPrice.floatValue());
} catch (RemoteException e) {
e.printStackTrace();
}
}
}

```

EJB Web サービスの作成: オプション 2:

考慮すべき代替オプションは、EJB に Web サービスを作成できる、Rational Application Developer Web サービス・ツールです。

注: この方法を使用してマイグレーションを試みる前に、サイト <http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.was.creation.ejb.ui.doc/tasks/twsejbw.html> の情報を参照してください。

注: このオプションでは、Web サービス・ウィザードを呼び出す前に、WebSphere Integration Developer を介して Web サービス・ランタイムが構成されていることが必要です。

EJB に Web サービスを作成するには、以下のステップに従ってください。

1. サービスを作成している EJB のコンテナであるエンタープライズ・アプリケーション・プロジェクトを右クリックします。
2. 「プロパティ」を選択し、「サーバー」プロパティで、「ターゲット・ランタイム (Target runtime)」が、「**WebSphere Process Server v6.0**」に設定され、「デフォルト・サーバー」が、インストール済みの「**WebSphere Process Server v6.0**」に設定されていることを確認します。
3. テスト・サーバーを開始し、そのサーバーにこのアプリケーションをデプロイして、正常に開始されるようにします。
4. J2EE パースペクティブで、「プロジェクト・エクスプローラー」ビューの「**EJB プロジェクト**」を展開します。「**デプロイメント記述子 (Deployment Descriptor)**」を展開し、次に「**セッション Bean**」カテゴリを展開します。Web サービスを生成する Bean を生成します。
5. 右クリックして、「**Web サービス**」→「**Web サービスの作成**」を選択します。
6. **Start Web service in Web project** 「**Web サービス・タイプ (Web Service Type)**」について、「**EJB Web サービス**」を選択し、Web サービスを今すぐにデプロイしたい場合を除き、「**Web プロジェクトの Web サービスを開始 (Start Web service in Web project)**」オプションのチェック・マークを外します。「次へ」をクリックします。
7. 右クリックした EJB がここで選択されていることを確認して、「次へ」をクリックします。
8. ここで、サービス・デプロイメント・オプションを構成する必要があります。「編集...」ボタンをクリックします。サーバー・タイプには、「**WPS Server v6.0**」を選択し、Web サービス・ランタイムには、「**IBM WebSphere**」および「**J2EE バージョン 1.4 (J2EE version 1.4)**」を選択します。これを

行うことで有効な組み合わせを選択できない場合は、v1.4 レベルへの J2EE プロジェクトのマイグレーションについて、セクション『マイグレーションの準備』を参照してください。「OK」をクリックします。

9. サービス・プロジェクトについて、EJB が含まれている EJB プロジェクトの名前を入力します。また、該当する EAR プロジェクトも選択します。「次へ」をクリックします。このとき、数分間待たなければならない場合があります。
10. 「Web サービス EJB の構成」パネルで、使用する該当ルーター・プロジェクトを選択します (作成したいルーター Web プロジェクトの名前を選択すると、このプロジェクトは、オリジナルの EJB と同じエンタープライズ・アプリケーションに追加されます)。希望するトランスポート (「HTTP 上の SOAP」または「JMS 上の SOAP」) を選択します。「次へ」をクリックします。
11. WSDL 定義を入れる WSDL ファイルを選択します。Web サービスで公開したいメソッドを選択し、該当するスタイル/エンコード (文書/リテラル、RPC/リテラル、または RPC/エンコード) を選択します。「パッケージから名前空間へのカスタム・マッピングを定義する」オプションを選択し、EJB で使用するすべての Java パッケージについて、マイグレーションされる EJB 内で固有な名前空間を選択します (デフォルトの名前空間は、パッケージ名が固有ですが、同じ Java クラスを使用する別の Web サービスを作成する場合に、競合が発生する可能性があります)。適宜、その他のパラメーターを入力します。それぞれのスタイル/エンコードの組み合わせには、制限があります。制限についての詳細は、<http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.doc/ref/rlimit.html> を参照してください。
12. 「次へ」をクリックし、「Web サービスのパッケージから名前空間へのマッピング」パネルで、「追加」ボタンをクリックし、作成される行に EJB のパッケージ名を入力してから、この EJB を一意的に識別するカスタム名前空間を入力します。EJB インターフェースで使用するすべての Java パッケージに対して、マッピングの追加を続行します。
13. 「次へ」をクリックします。このとき、数分間待たなければならない場合があります。
14. 「終了」をクリックします。サービス・プロジェクトが EJB サービスのコンシューマーであった場合には、ウィザードが完了した後、EJB サービスを記述する生成済み WSDL ファイルをビジネス・インテグレーション・モジュール・プロジェクトにコピーする必要があります。このプロジェクトは、フォルダー WebContent/WEB-INF/wsdl の下の生成済みルーター Web プロジェクトの中にあります。ビジネス・インテグレーション・モジュール・プロジェクトの更新/再ビルドを行ってください。
15. ビジネス・インテグレーション・パースペクティブに切り替え、マイグレーションされたモジュールを展開し、次に「Web サービス・ポート」論理カテゴリを展開します。
16. 前のステップで生成されたポートを選択し、そのポートをアセンブリ・エディターにドラッグ・アンド・ドロップして、**Web サービス・バインディング付きインポート**の作成を選択します。プロンプトが出されたら、EJB の WSDL インターフェースを選択します。これで、5.1 で EJB を利用する SCA コンポーネントをこのインポートにワイヤリングして、手動による再ワイヤリングのマイグレーション・ステップを完了することができます。

WebSphere Studio Application Developer Integration Edition でトップダウンのアプローチを使用していた場合は、WSDL 定義から EJB スケルトンを生成した後、以下のステップを実行してください。

1. 新しい Web プロジェクトを作成し、EJB スケルトンの生成元にしたい WSDL ファイルを、この Web プロジェクトのソース・フォルダーにコピーします。
2. EJB スケルトンの生成元にしたいポート・タイプが含まれている WSDL ファイルを右クリックして、「Web サービス」→「Java bean スケルトンを生成」を選択します。
3. Web サービスのタイプ「スケルトン EJB Web サービス」を選択して、ウィザードを完了します。

ウィザードを完了した後は、サービス・インターフェースを実装する EJB が作成され、それは WSIF API に依存したものではないはずです。

このインターフェースは 5.1 のインターフェースと多少の違いがあり、5.1 コンシューマーと新しいインポートの間にインターフェース・メディエーション・コンポーネントを挿入することが必要な場合があります。これを行うためには、アセンブリ・エディターでワイヤー・ツールをクリックして、SCA ソース・コンポーネントを、この新しい **Web サービス・バインディング付きインポート** にワイヤリングします。インターフェースが異なるため、「ソース・ノードとターゲット・ノードにマッチング・インターフェースがありません。」というプロンプトが出されます。ソース・ノードとターゲット・ノードの間にインターフェース・マッピングを作成することを選択します。アセンブリ・エディターで作成されたマッピング・コンポーネントをダブルクリックします。これにより、マッピング・エディターが開きます。インターフェース・マッピングの作成についての説明は、インフォメーション・センターを参照してください。

これが完了したら、EJB サービスを再ワイヤリングする必要があります。参照は存在しないはずであるため、必要なのは Java コンポーネントのインターフェースの再ワイヤリングのみです。

- このサービスが同じモジュール内のビジネス・プロセスによって呼び出される場合は、該当するビジネス・プロセス参照からのこの EJB コンポーネントへのワイヤーを作成します。
- このサービスが別のモジュール内のビジネス・プロセスによって呼び出される場合は、**SCA バインディング付きエクスポート**を作成し、このエクスポートを他のモジュールからそのモジュールのアセンブリ・エディターにドラッグ・アンド・ドロップして、対応する **SCA バインディング付きインポート**を作成します。該当するビジネス・プロセス参照をそのインポートにワイヤリングします。
- このサービスが、外部での公開のために、WebSphere Studio Application Developer Integration Edition で公開されていた場合は、サービスの再公開の方法について、セクション『インバウンド非 BPEL サービス・マイグレーション』を参照してください。

ビジネス・プロセス・サービス呼び出しへのビジネス・プロセスのマイグレーション:

このシナリオは、別のビジネス・プロセスを呼び出すビジネス・プロセスに適用されます。この場合、2 番目のビジネス・プロセスは、WSIF プロセス・バインディングを使用して呼び出されます。本セクションでは、ワイヤーまたは **SCA バインディング付きインポート/エクスポート**を使用して、BPEL を BPEL サービス呼び出しにマイグレーションする方法を説明します。

アウトバウンド・サービス・マイグレーション用のプロセス (BPEL) バインディング・サービス・プロジェクトをマイグレーションするには、以下のステップに従います。

1. ビジネス・インテグレーション・パースペクティブでモジュールを展開し、内容を参照します。モジュール・プロジェクトの下にある最初の項目をダブルクリックしてアセンブリ・エディターを開きます (これはプロジェクトと同じ名前になります)。
2. BPEL プロセスが別の BPEL プロセスを呼び出すことができるシナリオは複数あります。以下のシナリオから、ご使用のアプリケーションに適用されるものを見つけてください。
 - 呼び出される BPEL が同じモジュール内にある場合、最初の BPEL コンポーネント上の該当する参照から、ターゲット BPEL コンポーネント上の該当するインターフェースへのワイヤーを作成します。
 - 呼び出される BPEL が別のモジュールにある場合 (もう一方のモジュールがマイグレーション済みサービス・プロジェクトの場合)、以下のようにします。
 - a. モジュールのアセンブリ図に、2 番目のビジネス・プロセスの **SCA バインディング付きエクスポート**を作成します。
 - b. 「ビジネス・インテグレーション」ビューのナビゲーターで、2 番目のモジュールのアセンブリ・アイコンを展開します。作成したばかりのエクスポートが表示されます。
 - c. 2 番目のモジュールの「ビジネス・インテグレーション」ビューから、最初のモジュールの開いているアセンブリ・エディターにエクスポートをドラッグ・アンド・ドロップします。これに

よって、最初のモジュール内に SCA バインディング付きインポートが作成されます。このサービスが、外部での公開のために WebSphere Studio Application Developer Integration Edition で公開されていた場合、セクション『マイグレーション済みサービスにアクセスするために SCA エクスポートを作成』を参照してください。

- d. 最初のビジネス・プロセスの該当する参照から、モジュール内に作成したばかりのインポートにワイヤリングします。
 - e. アセンブリー図を保管します。
- 2 番目のビジネス・プロセスを呼び出す際に遅延バインディングを行うには、以下のようにします。
 - a. 1 番目のビジネス・プロセス・コンポーネントの参照をワイヤリングされていないままにします。BPEL エディターで 1 番目のプロセスを開き、「参照パートナー」セクションの下で、遅延バインディングを使用して呼び出す 2 番目の BPEL プロセスに対応するパートナーを選択します。
 - b. 「記述」タブの「プロパティ」ビューで、2 番目のビジネス・プロセスの名前を「プロセス・テンプレート (Process Template)」フィールドに入力します。
 - c. ビジネス・プロセスを保管します。これで、遅延バインド呼び出しのセットアップを終了しました。

Web サービスのマイグレーション (SOAP/JMS):

Web サービス (SOAP/JMS) を、Web サービス・バインディング付き SCA インポートにマイグレーションできます。

アウトバウンド・サービス・マイグレーション用の SOAP/JMS サービス・プロジェクトをマイグレーションするには、以下のステップに従います。

1. 最初に、マイグレーション・ウィザードを使用してサービス・プロジェクトをインポートする必要があります。この結果、ビジネス・インテグレーション・モジュールが作成され、WSDL メッセージ、ポート・タイプ、バインディング、およびサービスが WebSphere Studio Application Developer Integration Edition に生成されます。このアプリケーションが起動する IBM Web サービス (SOAP/JMS) も、マイグレーションされる WebSphere Studio Application Developer Integration Edition Web サービスである場合には、マイグレーション時に Web サービスへの更新が行われた可能性があります。その場合、ここでは Web サービスのマイグレーション済み WSDL ファイルを使用してください。
2. ビジネス・インテグレーション・パースペクティブでモジュールを展開し、内容を参照できるようにします。モジュール・プロジェクトの下にある最初の項目をダブルクリックしてアセンブリー・エディターを開きます (これはプロジェクトと同じ名前になります)。
3. 次に、インポートを追加して、アプリケーションが SCA プログラミング・モデルに従って IBM Web サービスと (SOAP/JMS を介して) 対話できるようにします。WSDL インターフェース、バインディング、およびサービス定義が、マイグレーションされたモジュール内、またはマイグレーションされたモジュールが従属するライブラリー内に存在することを確認してください。
4. ビジネス・インテグレーション・パースペクティブで、マイグレーションされたモジュールを展開し、そのアセンブリー図をアセンブリー・エディター内で開きます。
5. 「Web サービス・ポート」論理カテゴリーを展開して、呼び出したいサービスに対応するポートをアセンブリー・エディターにドラッグ・アンド・ドロップします。
6. Web サービス・バインディング付きインポートの作成を選択します。
7. インポートを作成した後、そのインポートをアセンブリー・エディター内で選択して、「プロパティ」ビューに進みます。「バインディング」タブの下に、そのインポートがバインドされるポートとサービスが表示されます。

8. アセンブリー図を保管します。

これが完了したら、以下のようにして、サービスを再ワイヤリングする必要があります。

- このサービスが同じモジュール内のビジネス・プロセスによって呼び出される場合は、該当するビジネス・プロセス参照からこのインポートへのワイヤーを作成します。
- このサービスが別のモジュール内のビジネス・プロセスによって呼び出される場合は、**SCA バインディング付きエクスポート**を作成し、このエクスポートを他のモジュールからそのモジュールのアセンブリー・エディターにドラッグ・アンド・ドロップして、対応する **SCA バインディング付きインポート**を作成します。該当するビジネス・プロセス参照をそのインポートにワイヤリングします。
- アセンブリー図を保管します。

Web サービスのマイグレーション (SOAP/HTTP):

Web サービス (SOAP/HTTP) を、Web サービス・バインディング付き SCA インポートにマイグレーションすることができます。

アウトバウンド・サービス・マイグレーション用の SOAP/HTTP サービス・プロジェクトをマイグレーションするには、以下のステップに従います。

1. 最初に、マイグレーション・ウィザードを使用してサービス・プロジェクトをインポートする必要があります。この結果、ビジネス・インテグレーション・モジュールが作成され、WSDL メッセージ、ポート・タイプ、バインディング、およびサービスが WebSphere Studio Application Developer Integration Edition に生成されます。このアプリケーションが起動する IBM Web サービス (SOAP/HTTP) も、マイグレーションされる WebSphere Studio Application Developer Integration Edition Web サービスである場合には、マイグレーション時に Web サービスへの更新が行われた可能性があります。その場合、ここでは Web サービスのマイグレーション済み WSDL ファイルを使用してください。
2. ビジネス・インテグレーション・パースペクティブでモジュールを展開し、内容を参照できるようにします。モジュール・プロジェクトの下にある最初の項目をダブルクリックしてアセンブリー・エディターを開きます (これはプロジェクトと同じ名前になります)。
3. 次に、インポートを追加して、アプリケーションが SCA プログラミング・モデルに従って IBM Web サービスと (SOAP/HTTP を介して) 対話できるようにします。WSDL インターフェース、バインディング、およびサービス定義が、マイグレーションされたモジュール内、またはマイグレーションされたモジュールが従属するライブラリー内に存在することを確認してください。
4. ビジネス・インテグレーション・パースペクティブで、マイグレーションされたモジュールを展開し、そのアセンブリー図をアセンブリー・エディター内で開きます。
5. 「Web サービス・ポート」論理カテゴリーを展開して、呼び出したいサービスに対応するポートをアセンブリー・エディターにドラッグ・アンド・ドロップします。
6. **Web サービス・バインディング付きインポート**の作成を選択します。
7. インポートを作成した後、そのインポートをアセンブリー・エディター内で選択して、「プロパティ」ビューに進みます。「バインディング」タブの下に、そのインポートがバインドされるポートとサービスが表示されます。
8. アセンブリー図を保管します。

これが完了したら、以下のようにして、サービスを再ワイヤリングする必要があります。

- このサービスが同じモジュール内のビジネス・プロセスによって呼び出される場合は、該当するビジネス・プロセス参照からこのインポートへのワイヤーを作成します。
- このサービスが別のモジュール内のビジネス・プロセスによって呼び出される場合は、**SCA バインディング付きエクスポート**を作成し、このエクスポートを他のモジュールからそのモジュールのアセンブ

リー・エディターにドラッグ・アンド・ドロップして、対応する **SCA バインディング付きインポート** を作成します。該当するビジネス・プロセス参照をそのインポートにワイヤリングします。

- アセンブリー図を保管します。

JMS サービスのマイグレーション:

JMS サービスを JMS バインディング付き SCA インポートにマイグレーションすることができます。

注: JMS メッセージが WebSphere Business Integration Adapter に送信される場合は、セクション『WebSphere Business Integration Adapter との相互作用のマイグレーション』を参照してください。

アウトバウンド・サービス・マイグレーション用の JMS サービス・プロジェクトをマイグレーションするには、以下のステップに従います。

1. 最初に、マイグレーション・ウィザードを使用してサービス・プロジェクトをインポートする必要があります。この結果、ビジネス・インテグレーション・モジュールが作成され、WSDL メッセージ、ポート・タイプ、バインディング、およびサービスが WebSphere Studio Application Developer Integration Edition に生成されます。
2. ビジネス・インテグレーション・パースペクティブでモジュールを展開し、内容を参照できるようにします。モジュール・プロジェクトの下にある最初の項目をダブルクリックしてアセンブリー・エディターを開きます (これはプロジェクトと同じ名前になります)。
3. 次に、インポートを追加して、アプリケーションが SCA プログラミング・モデルに従って JMS キューと対話できるようにします。
4. アセンブリー・エディターで、マイグレーションされたモジュール・プロジェクトを展開し、「インターフェース」カテゴリを展開して、アプリケーションが呼び出す Web サービスを記述する WSDL ポート・タイプを見つけます。これをアセンブリー・エディターにドラッグ・アンド・ドロップします。
5. 「コンポーネントの作成」ダイアログでは、作成するコンポーネントのタイプを選択できます。「バインディングのないインポート」を選択します。
6. アセンブリー・エディターで作成された新しいインポートが表示され、そのインポートを選択して、「記述」タブの「プロパティ」ビューに進めば、インポートの名前を変更して、より分かりやすい名前が表示できます。
7. 5.1 の WSDL バインディングおよびサービス・ファイルを参照して、マイグレーションする JMS サービスについての詳細を見つけ、それらを使用して 6.0 の「JMS バインディング付きインポート」の詳細を入力することができます。5.1 のサービス・プロジェクト内で、5.1 の JMS バインディングおよびサービスの WSDL ファイル (通常、*JMSBinding.wsdl および *JMSService.wsdl という名前です) を見つけてください。そこに収集されているバインディングとサービスの情報を検査してください。バインディングからは、テキスト・メッセージとオブジェクト・メッセージのどちらが使用されたか、およびカスタム・データ・フォーマット・バインディングが使用されたかどうかを判別できます。これらが使用されていた場合には、6.0 の「JMS バインディング付きインポート」にカスタム・データ・バインディングを作成することを検討する必要があります。サービスからは、初期コンテキスト・ファクトリー、JNDI 接続ファクトリー名、JNDI 宛先名、および宛先スタイル (キュー) を見つけることができます。
8. そのインポートを右クリックして、「バインディングの生成 (Generate Binding)」を選択し、次に「JMS バインディング」を選択します。以下のパラメーターの入力を求めるプロンプトが出されます。

JMS メッセージング・ドメインの選択:

- Point-to-Point
- Publish-Subscribe

- Domain-Independent

ビジネス・オブジェクトおよび JMS メッセージ間でのデータのシリアル化方法の選択:

- テキスト
- オブジェクト
- ユーザー提供

「ユーザー提供」を選択した場合、以下のようになります。

`com.ibm.websphere.sca.jms.data.JMSDataBinding` 実装クラスの完全修飾名を指定します。ご使用のアプリケーションが、「JMS インポート・バインディング」で通常は使用できない JMS ヘッダー・プロパティの設定を必要とする場合は、ユーザー定義のデータ・バインディングを指定する必要があります。この場合は、標準 JMS データ・バインディング

`com.ibm.websphere.sca.jms.data.JMSDataBinding` を拡張するカスタム・データ・バインディング・クラスを作成し、カスタム・コードを追加して `JMSMessage` に直接アクセスすることができます。下のリンク先にある『インポートおよびエクスポート・コンポーネントのバインディングの作成および変更』で JMS の例を参照してください。

インバウンド接続がデフォルト JMS 関数セクター・クラスを使用している:

<selected> または <deselected>

- いま作成したインポートを選択します。「プロパティ」ビューの「バインディング」タブに移動します。そこにリストされたすべてのバインディング情報に、WebSphere Studio Application Developer Integration Edition で以前に指定したものと同じ値を手動で入力できます。指定可能なバインディング情報には以下があります。

- JMS インポート・バインディング (これが最重要)
- 接続
- リソース・アダプター
- JMS 宛先
- メソッド・バインディング

これが完了したら、以下のようにして、サービスを再ワイヤリングする必要があります。

- このサービスが同じモジュール内のビジネス・プロセスによって呼び出される場合は、該当するビジネス・プロセス参照からこのインポートへのワイヤーを作成します。
- このサービスが別のモジュール内のビジネス・プロセスによって呼び出される場合は、**SCA バインディング付きエクスポート**を作成し、このエクスポートを他のモジュールからそのモジュールのアセンブリ・エディターにドラッグ・アンド・ドロップして、対応する **SCA バインディング付きインポート**を作成します。該当するビジネス・プロセス参照をそのインポートにワイヤリングします。
- アセンブリ図を保管します。

J2C-IMS サービスのマイグレーション:

J2C-IMS サービスを、EIS バインディング付き SCA インポートまたは Web サービス・バインディング付き SCA インポートにマイグレーションできます。

この IMS™ サービス用に生成された WebSphere Studio Application Developer Integration Edition の成果物は使用しないでください。WebSphere Integration Developer で使用可能なウィザードを使用してサービスを再作成し、手動でアプリケーションの再ワイヤリングを行う必要があります。

注: 自動ビルドをオンにするか、またはモジュールを手動でビルドしてください。

以下のオプションがあります。

注: どちらのオプションでも、BPEL サービスがこの IMS サービスを呼び出す場合は、BPEL を多少変更する必要があります。これは、EIS サービスによって公開されるインターフェースは旧 5.1 インターフェースと多少異なるためです。これを行うには、BPEL エディターを開き、EIS サービスに対応するパートナー・リンクを調整して、上記のステップを実行したときに生成された新しいインターフェース (WSDL ファイル) を使用します。EIS サービスの新しい WSDL インターフェースに関して、BPEL アクティビティーに変更が必要であれば、それを行います。

各 J2C-IMS サービス再ワイヤリング・オプションの利点と欠点:

J2C-IMS サービス再ワイヤリング・オプションにはそれぞれ、利点と欠点があります。

以下のリストで、両方のオプションと、それぞれの利点および欠点について説明します。

- 最初のオプションは、標準 SCA コンポーネントを使用して IMS サービスを呼び出します。
- 最初のオプションには、以下のようにいくつかの制限事項があります。
 - SDO バージョン 1 仕様の API は、COBOL または C バイト配列へのアクセスを提供しません。これは、IMS 複数セグメントを操作する顧客に影響することになります。
 - シリアライゼーション用 SDO バージョン 1 仕様は、COBOL 再定義または C 共用体をサポートしません。
- 2 番目のオプションでは標準 JSR 109 の方法を使用して IMS サービスに接続します。この機能は、Rational Application Developer の一部として使用できます。

IMS サービスを呼び出す SCA インポートの作成: オプション 1:

IMS システムと通信するためにメッセージ/データを保管する DataObject を使用する、EIS バインディング付き SCA インポートを作成できます。

IMS サービスを呼び出す SCA インポートを作成するには、以下のステップに従ってください。

1. この新しい IMS サービスを収容するために、新しいビジネス・インテグレーション・モジュール・プロジェクトを作成します。
2. EIS サービスを再作成するために、「ファイル」→「新規」→「その他」→「ビジネス・インテグレーション」→「エンタープライズ・サービス・ディスカバリー」と進みます。
3. このウィザードを使用して、EIS システムからサービスをインポートできます。これは、5.1 で WSIF ベースの EIS サービスを作成した、WebSphere Studio Application Developer Integration Edition ウィザードに非常によく似ています。このウィザードで、新しい J2C IMS リソース・アダプターをインポートできます。WebSphere Integration Developer がインストールされているディレクトリを参照して、「リソース・アダプター (Resource Adapters)」→「ims15」→「imsico9102.rar」とドリルダウンする必要があります。

注: プロパティおよび操作パネルの保管の実行について詳しくは、インフォメーション・センターを参照してください。エンタープライズ・サービス・ディスカバリー・ウィザードで操作を追加するときに、その操作の入力データ型または出力データ型用のビジネス・オブジェクトを作成することもできます。このためには、WebSphere Studio Application Developer Integration Edition ウィザードで使用した、C または COBOL ソース・ファイルが必要です。古いサービス・プロジェクトにあるソース・ファイルをポイントできるように、これらのファイルをそのプロジェクトにコピーしておく必要があります。別のウィザード（「ファイル」→「新規」→「その他」→「ビジネス・インテグレーション」→「エンタープライズ・データ・ディスカバリー」）を使用して、ビジネス・オブジェクトをインポートすることもできます。

4. ウィザードを完了したら、ビジネス・インテグレーション・パースペクティブを開き、モジュールを展開して、その内容を確認できるようにします。モジュールの「データ型」に新しいビジネス・オブジェクトがリストされ、「インターフェース」に新しいインターフェースがリストされます。
5. モジュール・プロジェクトの下にある最初の項目をダブルクリックしてアセンブリ・エディターを開きます (これはプロジェクトと同じ名前になります)。キャンバス上にインポートが存在し、このインポートに EIS バインディングがあり、作成したばかりのサービスを表していることが分かります。

このサービスを利用者に公開する方法については、『マイグレーション済みサービスにアクセスするために SCA エクスポートを作成』というセクションを参照してください。

J2C サービスへの Web サービスの作成: オプション 2:

J2C Web サービスを作成することができ、そのサービスのコンシューマーが SCA コンポーネントである場合は、そのサービスを IBM Web サービス (SOAP/HTTP または SOAP/JMS) として利用することができます。

J2C サービスに Web サービスを作成するには、以下のステップに従ってください。

1. 「ファイル」 → 「新規」 → 「J2C」 → 「J2C Java Bean」とクリックすることにより、J2C Java Bean を作成します。
2. **IMS Connector for Java** の 1.5 バージョンを選択して、「次へ」をクリックします。
3. 「管理接続 (Managed Connection)」にチェック・マークを付け、JNDI ルックアップ名を入力します。「次へ」をクリックします。
4. 新しい Java Bean のプロジェクト、パッケージ、および名前を指定します。この Bean は、1 つのインターフェースと 1 つの実装クラスから構成されます。「次へ」をクリックします。
5. EIS からアクセスしたいそれぞれの関数またはサービスごとに、Java メソッドを追加します。追加のメソッドは、Java ソース・エディターで「Snippet」ビューを使用して後で追加できます。「追加...」ボタンをクリックし、そのメソッドの名前を選択して、「次へ」をクリックします。
6. これで、「参照...」を選択して、既存の型を再利用するか、または「新規...」を選択して、入出力データ型について CICS/IMS Java データ・バインディング・ウィザード (ここで、COBOL または C のソース・ファイルを参照できます) を起動できます。
7. Java メソッドの作成を終了したら、「次へ」をクリックします。
8. このウィザードの残りのステップを完了して、J2C Java Bean を作成します。
9. 「ファイル」 → 「新規」 → 「J2C」 → 「Web ページ、Web サービス、または J2C Java Bean からの EJB (Web Page, Web Service, or EJB from J2C Java Bean)」とクリックすることで Web サービスを作成して、J2C Java Bean に Web サービスを作成します。
10. ウィザードを完了します。

これで、このサービスのコンシューマーは、このウィザードによって生成された WSDL サービスを使用して IMS サービスを呼び出すことができます。

J2C-CICS ECI サービスのマイグレーション:

J2C-CICS ECI サービスを EIS バインディング付き SCA インポートまたは Web サービス・バインディング付き SCA インポートにマイグレーションできます。

トピック『J2C-IMS サービス・プロジェクトのマイグレーション』に記載された手順に従いますが、必ず IMS RAR ファイルの代わりに、次の RAR ファイルをインポートしてください。

- WebSphere Integration Developer がインストールされているディレクトリーを参照し、「リソース・アダプター (Resource Adapters)」 → 「cics15」 → 「cicseci.rar」までドリルダウンします。

2 番目のオプションに従って J2C Web サービスを作成する場合は、J2C Java Bean 作成ウィザードの 2 番目のパネルで v 1.5 の **ECIResourceAdapter** を選択します。

トピック『J2C-IMS サービスのマイグレーション』も参照してください。

J2C-CICS EPI サービスのマイグレーション:

WebSphere Integration Developer には、J2C-CICS EPI サービスに対する直接サポートはありません。SCA モジュールからこのサービスにアクセスするには、**使用量シナリオ** を使用してマイグレーションを行う必要があります。

このサービス・タイプを WebSphere Integration Developer にマイグレーションするための説明については、トピック『サービス・マイグレーションの使用量シナリオ』を参照してください。

J2C-HOD サービスのマイグレーション:

WebSphere Integration Developer には、J2C-HOD サービスに対する直接サポートはありません。SCA モジュールからこのサービスにアクセスするには、**使用量シナリオ** を使用してマイグレーションを行う必要があります。

このサービス・タイプを WebSphere Integration Developer にマイグレーションするための説明については、トピック『サービス・マイグレーションの使用量シナリオ』を参照してください。

変換プログラム・サービスのマイグレーション:

可能な部分について、変換プログラム・サービスを SCA データ・マップおよびインターフェース・マップにマイグレーションできます。**使用量シナリオ** を使用して、SCA モジュールからこのサービスにアクセスすることもできます。

データ・マップとインターフェース・マップのコンポーネントは、バージョン 6.0 の新機能です。これらのコンポーネントは、5.1 からの変換プログラム・サービスと類似した機能を提供しますが、完全な XSL 変換の機能は持っていません。ご使用の変換プログラム・サービスをこれらのコンポーネントのいずれかで置き換えることができない場合は、WebSphere Integration Developer には変換プログラム・サービスに対する直接のサポートがないため、**使用量シナリオ** を使用してマイグレーションを行う必要があります。『サービス・マイグレーションの使用量シナリオ』セクションで説明されているステップに従って、SCA モジュールからこのサービスにアクセスしてください。

サービス・マイグレーションの使用量シナリオ:

WebSphere Studio Application Developer Integration Edition サービス・タイプに直接対応するサービス・タイプがない場合、WebSphere Integration Developer でアプリケーションを再設計するときに古い WebSphere Studio Application Developer Integration Edition サービスをそのまま利用するには、**使用量シナリオ**が必要になります。

マイグレーション・ウィザードを呼び出す前 に、WebSphere Studio Application Developer Integration Edition で以下のステップを実行してください。

1. このクライアント・プロキシー・コードを保持するための新しい Java プロジェクトを作成します。このクライアント・プロキシー・コードをサービス・プロジェクトに書き込まないでください。バージョ

ン 5.1 スタイルで生成されたメッセージおよび Java Bean クラスは、サービス・プロジェクトをマイグレーションする自動マイグレーション・ウィザードでスキップされるためです。

2. WebSphere Studio Application Developer Integration Edition を開き、変換プログラム・バインディングおよびサービスが含まれる WSDL ファイルを右クリックして、「**エンタープライズ・サービス (Enterprise Services)**」 → 「**サービス・プロキシの生成 (Generate Service Proxy)**」を選択します。作成するプロキシのタイプを確認されますが、「**Web Services Invocation Framework (WSIF)**」のみが選択可能です。「**次へ**」をクリックします。
3. ここで、作成するサービス・プロキシ Java クラスのパッケージおよび名前を指定できます (現行サービス・プロジェクト内にプロキシを作成することになります)。「**次へ**」をクリックします。
4. ここで、プロキシ・スタイルを指定できます。「**クライアント・スタブ (Client Stub)**」を選択して、プロキシに組み込む目的の操作を選択し、「**終了**」をクリックします。これによって、WebSphere Studio Application Developer Integration Edition サービスと同じメソッドを公開する Java クラスが作成されます。このクラスでは、Java メソッドへの引数はソース WSDL メッセージのパーツとなります。

ここで、WebSphere Integration Developer へのマイグレーションを行うことができます。

1. クライアント・プロキシ Java プロジェクトを新しいワークスペースにコピーし、「**ファイル**」 → 「**インポート**」 → 「**既存プロジェクトをワークスペースへ**」に進んで、このプロジェクトをインポートします。
2. マイグレーション・ウィザードを使用してサービス・プロジェクトをインポートします。この結果、ビジネス・インテグレーション・モジュールが作成され、WSDL メッセージ、ポート・タイプ、バインディング、およびサービスが WebSphere Studio Application Developer Integration Edition に生成されます。
3. ビジネス・インテグレーション・パースペクティブでモジュールを展開し、内容を参照できるようにします。モジュール・プロジェクトの下にある最初の項目をダブルクリックしてアセンブリ・エディターを開きます (これはプロジェクトと同じ名前になります)。
4. カスタム Java コンポーネントを作成するには、モジュール・プロジェクトの下にある「**インターフェース**」を展開し、WebSphere Studio Application Developer Integration Edition でこの変換プログラム・サービスについて生成された WSDL インターフェースを選択します。
5. このインターフェースをアセンブリ・エディターにドラッグ・アンド・ドロップします。作成するコンポーネントのタイプを選択するように求めるダイアログがポップアップ表示されます。「**実装タイプのないコンポーネント**」を選択して、「**OK**」をクリックします。
6. アセンブリ図に、汎用コンポーネントが表示されます。そのコンポーネントを選択して、「**プロパティ**」ビューに進みます。
7. 「**説明**」タブで、コンポーネントの名前を変更して、より分かりやすい名前を表示できます (この場合、ご使用の EJB の名前に類似した名前を付けますが、このコンポーネントは、WebSphere Studio Application Developer Integration Edition で変換プログラム・サービスについて生成された WSDL インターフェースと変換プログラム・クライアント・プロキシの Java インターフェースとの間を仲介する Java コンポーネントになるため、「**JavaMed**」などの接尾部を付加します)。
8. 「**詳細**」タブで、このコンポーネントに 1 つのインターフェース (アセンブリ・エディターにドラッグ・アンド・ドロップしたインターフェース) があることが分かります。
9. アセンブリ・エディターに戻り、作成したばかりのコンポーネントを右クリックして、「**実装の生成...**」 → 「**Java**」を選択します。Java の実装が生成されるパッケージを選択します。これにより、SCA プログラミング・モデル (複合タイプは `commonj.sdo.DataObject` であるオブジェクトによって表され、単純タイプはそれと同等の Java オブジェクトによって表される) に準拠した WSDL インターフェースに従ったスケルトン Java サービスが作成されます。

ここで、生成された Java 実装クラスの中の「//TODO」タグで示されている部分に、コードを入力する必要があります。次の 2 つのオプションがあります。

1. ロジックをオリジナルの Java クラスからこのクラスに移動し、新しいデータ構造に適合させます。
2. この生成済み Java クラスの内側に、古い Java クラスの専用インスタンスを作成し、以下のことを行うためのコードを作成します。
 - a. 生成された Java 実装クラスのすべてのパラメーターを、古い Java クラスが期待するパラメーターに変換する。
 - b. 変換されたパラメーターを使用して、古い Java クラスの専用インスタンスを呼び出す。
 - c. 古い Java クラスの戻り値を、生成された Java 実装メソッドで宣言されている戻り値の型に変換する。

上記のオプションを完了した後、クライアント・プロキシを再ワイヤリングする必要があります。「参照」は存在しないはずであるため、必要なのは Java コンポーネントのインターフェースの再ワイヤリングのみです。

- このサービスが同じモジュール内のビジネス・プロセスによって呼び出される場合は、該当するビジネス・プロセス参照からこの Java コンポーネントのインターフェースへのワイヤーを作成します。
- このサービスが別のモジュール内のビジネス・プロセスによって呼び出される場合は、**SCA バインディング付きエクスポート**を作成し、このエクスポートを他のモジュールからそのモジュールのアセンブリー・エディターにドラッグ・アンド・ドロップして、対応する **SCA バインディング付きインポート**を作成します。該当するビジネス・プロセス参照をそのインポートにワイヤリングします。
- このサービスが、外部での公開のために WebSphere Studio Application Developer Integration Edition で公開されていた場合、サービスの再公開の方法については、セクション『マイグレーション済みサービスにアクセスするために SCA エクスポートを作成』を参照してください。

マイグレーション済みサービスにアクセスするために SCA エクスポートを作成:

WebSphere Studio Application Developer Integration Edition サービス・プロジェクトにデプロイメント・コードを生成したすべてのサービスの SCA モデルに従って、外部コンシューマーがマイグレーション済みサービスを使用できるようにするためには、SCA エクスポートを作成する必要があります。これには、アプリケーション外部のクライアントが呼び出すすべてのサービスが含まれます。

WebSphere Studio Application Developer Integration Edition から、BPEL プロセスまたは他の Service WSDL を右クリックして、「**エンタープライズ・サービス (Enterprise Services)**」→「**デプロイメント・コードの生成**」を選択した場合は、次の手動マイグレーション・ステップを実行する必要があります。

WebSphere Integration Developer は、すべてのデプロイメント・オプションを保管するという点で、WebSphere Studio Application Developer Integration Edition と異なります。プロジェクトがビルドされると、デプロイメント・コードは、生成済み EJB と Web プロジェクトで自動的に更新されるため、手動で「**デプロイメント・コードの生成**」を行うためのオプションはなくなりました。

「BPEL デプロイメント・コードの生成 (Generate BPEL Deploy Code)」ウィザードの「パートナー用インターフェース (Interfaces for Partners)」セクションには 5 つのバインディング・オプションがありました。以下のインバウンド BPEL サービス・マイグレーション情報には、WebSphere Studio Application Developer Integration Edition で選択されたデプロイメント・バインディング・タイプに基づいて作成するエクスポートのタイプおよびプロパティに関する詳細があります。

- EJB
- IBM Web サービス (SOAP/JMS)
- IBM Web サービス (SOAP/HTTP)

- Apache Web サービス (SOAP/HTTP)
- JMS

EJB および EJB プロセス・バインディングのマイグレーション:

EJB および EJB プロセス・バインディングは、推奨 SCA 構成にマイグレーションすることができます。

WebSphere Studio Application Developer Integration Edition で、このバインディング・タイプは、EJB を呼び出すことによって、クライアントによる BPEL プロセスや他のサービスとの通信を可能にしました。マイクロプロセスの場合、このバインディング・タイプはオプションではなかったことに注意してください。生成済み EJB が他のバインディング・タイプによって内部的に使用されるために、必ず選択されていました。

生成済み EJB の JNDI 名は、BPEL の名前、ターゲット名前空間、および開始日付タイム・スタンプの組み合わせとして自動的に生成されました。例えば、これらの属性は、「説明」および「サーバー」のコンテンツ・タブにおいて BPEL エディターで BPEL プロセスのプロパティを調べることによって見つけることができます。

表 3. 生成済み名前空間

プロセス名	MyService
ターゲット名前空間	http://www.example.com/process87787141/
開始日付	Jan 01 2003 02:03:04

この例の生成済み名前空間は com/example/www/process87787141/MyService20030101T020304 です。

WebSphere Studio Application Developer Integration Edition では、EJB バインディングがデプロイメント・タイプとして選択された場合、指定されるオプションはありませんでした。

WebSphere Studio Application Developer Integration Edition プロセス・バインディングをマイグレーションするには、4 つのオプションがあります。サービスにアクセスするクライアントのタイプによって、次のどのマイグレーション・オプションが実行されるかが決まります。

注: 手動マイグレーション・ステップの完了後、クライアントも新しいプログラミング・モデルにマイグレーションする必要があります。次のクライアント・タイプについては、該当するトピックを参照してください。

表 4. クライアントのマイグレーションに関する詳細情報

クライアント・タイプ	詳細については以下を参照
生成済みセッション Bean を呼び出す EJB クライアント。このようなクライアントは、呼び出す BPEL 操作に対応する EJB メソッドを呼び出します。	EJB クライアントのマイグレーション
EJB プロセス・バインディングを使用する WSIF クライアント	EJB プロセス・バインディング・クライアントのマイグレーション
汎用 business process choreographer EJB API	Business Process Choreographer 汎用 EJB API クライアントのマイグレーション
汎用 business process choreographer Messaging API	Business Process Choreographer 汎用 Messaging API クライアントのマイグレーション
同じモジュール内の他の BPEL プロセス	N/A: コンポーネント・アセンブリ・エディターをともに使用するワイヤー BPEL コンポーネント

表 4. クライアントのマイグレーションに関する詳細情報 (続き)

クライアント・タイプ	詳細については以下を参照
異なるモジュール内の別の BPEL プロセス	N/A: 参照元モジュールに SCA バインディング付きインポート を作成して、次のオプション 1 で作成する SCA バインディング付きインポート をポイントするようにそのバインディングを構成します。

重要な点として、ビジネス・プロセスが、そのモジュールの外部に (サービス参照を介して) ビジネス・プロセス自体への参照を渡す場合は、常に下記のオプション 1 に従って (常にこれらのオプションのうち複数を実行できます)、そのビジネス・プロセスの SCA バインディング付きエクスポートを作成する必要があることに注意してください。ビジネス・プロセスのエクスポートにはモジュールのデフォルト・エクスポートとしてマークを付ける必要があるため、モジュールごとに 1 つのビジネス・プロセスのみが、そのサービス参照をモジュールの外部に渡すことができます。これを行うには、次などの、エクスポートの「default」という名前の属性に「true」を指定します。

Default endpoint reference

「ビジネス・インテグレーション」ビューで「エクスポート」を右クリックし、「**アプリケーションから開く**」を選択してから、「**テキスト・エディター**」を選択して、このビジネス・プロセスのエクスポートに手動でデフォルトのマークを付ける必要があります。

EJB および EJB プロセス・バインディングのマイグレーション・オプション 1:

WebSphere Studio Application Developer Integration Edition EJB プロセス・バインディングの最初のマイグレーション・オプションは、ビジネス・プロセスによる同じモジュール内の別コンポーネントのアクセスを可能にします。

アセンブリー・エディターで、この別コンポーネントを BPEL コンポーネントにワイヤリングします。

1. ツールバーから「**ワイヤー**」項目を選択します。
2. 他のコンポーネントをクリックして、ワイヤーのソースとして選択します。
3. 「**BPEL SCA**」コンポーネントをクリックして、ワイヤーのターゲットとして選択します。
4. アセンブリー図を保管します。

EJB および EJB プロセス・バインディングのマイグレーション・オプション 2:

WebSphere Studio Application Developer Integration Edition EJB プロセス・バインディングの 2 番目のマイグレーション・オプションは、ビジネス・プロセスによる他の SCA モジュールおよびクライアントのアクセスを可能にします。

注: 以下の手順は、汎用 business process choreographer API がビジネス・プロセスの起動に使用される場合に必須です。

SCA バインディング付きエクスポートは、他の SCA モジュールによる SCA コンポーネントのアクセスを可能にします。SCA バインディング付きエクスポートを作成するには、以下を行います。

1. マイグレーション・ウィザードによって作成されたモジュールをアセンブリー・エディターで開きます。
2. EJB バインディングが WebSphere Studio Application Developer Integration Edition に生成された BPEL プロセス・インターフェースごとに、SCA バインディング付きエクスポートを作成します。
 - a. アセンブリー・エディターで BPEL コンポーネントを右クリックします。

- b. 「エクスポート...」を選択します。
- c. 「SCA バインディング」を選択します。
- d. プロセスに複数のインターフェースがある場合は、このバインディング・タイプでエクスポートするインターフェースを選択します。
- e. SCA エクスポートが作成されたら、アセンブリ・エディターと「プロパティ」ビューでエクスポートを選択し、「説明」コンテンツ・ペインを選択します。エクスポートの名前と説明がリストされ、必要であれば変更できます。
- f. アセンブリ図を保管します。

EJB および EJB プロセス・バインディングのマイグレーション・オプション 3:

WebSphere Studio Application Developer Integration Edition EJB プロセス・バインディングの 3 番目のマイグレーション・オプションは、非 SCA エンティティ (例えば JSP や Java クライアント) によるモジュールのアクセスを可能にします。

スタンドアロン参照は、外部クライアントによる SCA コンポーネントのアクセスを可能にします。スタンドアロン参照を作成するには、以下を行います。

1. マイグレーション・ウィザードによって作成されたモジュールをアセンブリ・エディターで開きます。
2. EJB バインディングが WebSphere Studio Application Developer Integration Edition に生成された BPEL プロセス・インターフェースごとに、スタンドアロン参照を作成します。
 - a. ツールバーから「スタンドアロン参照」項目を選択します。
 - b. アセンブリ・エディターのキャンバスをクリックして、スタンドアロン参照 SCA エンティティを作成します。
 - c. ツールバーから「ワイヤー」項目を選択します。
 - d. 「スタンドアロン参照」エンティティをクリックして、ワイヤーのソースとして選択します。
 - e. 「BPEL SCA」コンポーネントをクリックして、ワイヤーのターゲットとして選択します。
 - f. アラート「ソース・ノード上にマッチング参照が作成されます。続きますか? (Matching reference will be created on the source node. Would you like to continue?)」が表示されたら、「OK」をクリックします。
 - g. 作成したばかりの「スタンドアロン参照」エンティティを選択して、「プロパティ」ビューで「説明」コンテンツ・ペインを選択します。
 - h. 「参照」リンクを展開して、いま作成したリファレンスを選択します。リファレンスの名前と説明がリストされ、必要であれば変更できます。
 - i. プロセスに複数のインターフェースがある場合は、このバインディング・タイプでエクスポートするインターフェースを選択します。
 - j. アセンブリ図を保管します。

EJB および EJB プロセス・バインディングのマイグレーション・オプション 4:

WebSphere Studio Application Developer Integration Edition EJB プロセス・バインディングの 4 番目のマイグレーション・オプションは、Web サービス・クライアントによるビジネス・プロセスのアクセスを可能にします。

Web サービス・バインディング付きエクスポートは、外部 Web サービス・クライアントによる SCA コンポーネントのアクセスを可能にします。Web サービス・バインディング付きエクスポートを作成するには、以下を行います。

1. マイグレーション・ウィザードによって作成されたモジュールをアセンブリー・エディターで開きます。
2. EJB バインディングが WebSphere Studio Application Developer Integration Edition に生成された BPEL プロセス・インターフェースごとに、SCA バインディング付きエクスポートを作成します。
 - a. アセンブリー・エディターで BPEL コンポーネントを右クリックします。
 - b. 「エクスポート...」を選択します。
 - c. 「Web サービス・バインディング」を選択します。
 - d. プロセスに複数のインターフェースがある場合は、このバインディング・タイプでエクスポートするインターフェースを選択します。
 - e. トランスポート、**soap/http** または **soap/jms** を選択します。
 - f. Web サービス付きエクスポートが作成されたら、アセンブリー・エディターと「プロパティ」ビューでエクスポートを選択し、「説明」コンテンツ・ペインを選択します。エクスポートの名前と説明がリストされ、必要であれば変更できます。
 - g. アセンブリー図を保管します。

JMS および JMS プロセス・バインディングのマイグレーション:

JMS および JMS プロセス・バインディングは、推奨 SCA 構成にマイグレーションすることができます。

WebSphere Studio Application Developer Integration Edition では、このバインディング・タイプは、クライアントに対して、メッセージを MDB に送信することによって BPEL プロセスや他のサービスと通信する機能を与えます。このバインディング・タイプは、長時間実行プロセスのオプションではなく、必ず選択されていたことに注意してください。事実、このバインディング・タイプは、長時間実行プロセスの要求/応答インターフェース用に許可された、**唯一のバインディング・タイプ**でした。その他のサービス・タイプについては、MDB が生成され、これが該当するサービスを呼び出します。

JMS バインディングによって使用される JNDI 名は、BPEL の名前、ターゲット名前空間、および開始日付タイム・スタンプの組み合わせでした。

WebSphere Studio Application Developer Integration Edition では、JMS バインディングが BPEL プロセスのデプロイメント・タイプとして選択された場合、以下のオプションが与えられました。

- **JNDI 接続ファクトリー** - デフォルトは `jms/BPECF` です (これは、ターゲット・ビジネス・プロセス・コンテナのキュー接続ファクトリーの JNDI 名です)。
- **JNDI 宛先キュー** - デフォルトは `jms/BPEIntQueue` です (これはターゲット・ビジネス・プロセス・コンテナの内部キューの JNDI 名です)。
- **JNDI プロバイダー URL: サーバー提供またはカスタム** - アドレスを入力する必要があります。デフォルトは `iiop://localhost:2809` です。

WebSphere Studio Application Developer Integration Edition JMS プロセス・バインディングのマイグレーションには、5 つのオプションがあります。サービスにアクセスするクライアントのタイプによって、次のどのマイグレーション・オプションが実行されるかが決まります。

注: 手動マイグレーション・ステップの完了後、クライアントも新しいプログラミング・モデルにマイグレーションする必要があります。次のクライアント・タイプについては、該当するトピックを参照してください。

表 5. クライアントのマイグレーションに関する詳細情報

クライアント・タイプ	詳細については以下を参照
JMS プロセス・バインディングを使用する WSIF クライアント	Business Process Choreographer 汎用 Messaging API クライアントおよび JMS プロセス・バインディング・クライアントのマイグレーション
汎用 business process choreographer EJB API	Business Process Choreographer 汎用 EJB API クライアントのマイグレーション
ビジネスをマイグレーションする汎用 business process choreographer Messaging API	Business Process Choreographer 汎用 Messaging API クライアントのマイグレーション
同じモジュール内の他の BPEL プロセス	N/A: コンポーネント・アセンブリ・エディターをともに使用するワイヤー BPEL コンポーネント
異なるモジュール内の別の BPEL プロセス	N/A: 参照元モジュールに『SCA バインディング付きインポート』を作成して、次のオプション 1 で作成する『SCA バインディング付きエクスポート』をポイントするようにそのバインディングを構成します。

重要な点として、ビジネス・プロセスが、そのモジュールの外部に (サービス参照を介して) ビジネス・プロセス自体への参照を渡す場合は、常に下記のオプション 1 に従って (常にこれらのオプションのうち複数を実行できます)、そのビジネス・プロセスの SCA バインディング付きエクスポートを作成する必要があることに注意してください。ビジネス・プロセスのエクスポートにはモジュールのデフォルト・エクスポートとしてマークを付ける必要があるため、モジュールごとに 1 つのビジネス・プロセスのみが、そのサービス参照をモジュールの外部に渡すことができます。これを行うには、次などの、エクスポートの「default」という名前の属性に「true」を指定します。

Default endpoint reference

「ビジネス・インテグレーション」ビューで「エクスポート」を右クリックし、「アプリケーションから開く」を選択してから、「テキスト・エディター」を選択して、このビジネス・プロセスのエクスポートに手動でデフォルトのマークを付ける必要があります。

JMS および JMS プロセス・バインディングのマイグレーション・オプション 1:

WebSphere Studio Application Developer Integration Edition JMS プロセス・バインディングの最初のマイグレーション・オプションは、ビジネス・プロセスによる同じモジュール内の別コンポーネントのアクセスを可能にします。

アセンブリ・エディターで、この別コンポーネントを BPEL コンポーネントにワイヤリングします。

1. ツールバーから「ワイヤー」項目を選択します。
2. 他のコンポーネントをクリックして、ワイヤーのソースとして選択します。
3. 「BPEL SCA」コンポーネントをクリックして、ワイヤーのターゲットとして選択します。
4. アセンブリ図を保管します。

JMS および JMS プロセス・バインディングのマイグレーション・オプション 2:

WebSphere Studio Application Developer Integration Edition JMS プロセス・バインディングの 2 番目のマイグレーション・オプションは、ビジネス・プロセスによる他の SCA モジュールおよびクライアントのアクセスを可能にします。

SCA バインディング付きエクスポートは、他の SCA モジュールによる SCA コンポーネントのアクセスを可能にします。SCA バインディング付きエクスポートを作成するには、以下を行います。

1. マイグレーション・ウィザードによって作成されたモジュールをアセンブリ・エディターで開きます。
2. JMS バインディングが WebSphere Studio Application Developer Integration Edition に生成された BPEL プロセス・インターフェースごとに、SCA バインディング付きエクスポートを作成します。
 - a. アセンブリ・エディターで BPEL コンポーネントを右クリックします。
 - b. 「エクスポート...」を選択します。
 - c. 「SCA バインディング」を選択します。
 - d. プロセスに複数のインターフェースがある場合は、このバインディング・タイプでエクスポートするインターフェースを選択します。
 - e. SCA エクスポートが作成されたら、アセンブリ・エディターと「プロパティ」ビューでエクスポートを選択し、「説明」コンテンツ・ペインを選択します。エクスポートの名前と説明がリストされ、必要であれば変更できます。
 - f. アセンブリ図を保管します。

JMS および JMS プロセス・バインディングのマイグレーション・オプション 3:

WebSphere Studio Application Developer Integration Edition JMS プロセス・バインディングの 3 番目のマイグレーション・オプションは、非 SCA エンティティ (例えば JSP や Java クライアント) によるビジネス・プロセスのアクセスを可能にします。

スタンドアロン参照は、外部クライアントによる SCA コンポーネントのアクセスを可能にします。スタンドアロン参照を作成するには、以下を行います。

1. マイグレーション・ウィザードによって作成されたモジュールをアセンブリ・エディターで開きます。
2. JMS バインディングが WebSphere Studio Application Developer Integration Edition に生成された BPEL プロセス・インターフェースごとに、スタンドアロン参照を作成します。
 - a. ツールバーから「スタンドアロン参照」項目を選択します。
 - b. アセンブリ・エディターのキャンバスをクリックして、スタンドアロン参照 SCA エンティティを作成します。
 - c. ツールバーから「ワイヤー」項目を選択します。
 - d. 「スタンドアロン参照」エンティティをクリックして、ワイヤーのソースとして選択します。
 - e. 「BPEL SCA」コンポーネントをクリックして、ワイヤーのターゲットとして選択します。
 - f. アラート「ソース・ノード上にマッチング参照が作成されます。続きますか? (Matching reference will be created on the source node. Would you like to continue?)」が表示されたら、「OK」をクリックします。
 - g. 作成したばかりの「スタンドアロン参照」エンティティを選択して、「プロパティ」ビューで「説明」コンテンツ・ペインを選択します。
 - h. 「参照」リンクを展開して、いま作成したリファレンスを選択します。リファレンスの名前と説明がリストされ、必要であれば変更できます。
 - i. プロセスに複数のインターフェースがある場合は、このバインディング・タイプでエクスポートするインターフェースを選択します。
 - j. アセンブリ図を保管します。

JMS および JMS プロセス・バインディングのマイグレーション・オプション 4:

WebSphere Studio Application Developer Integration Edition JMS プロセス・バインディングの 4 番目のマイグレーション・オプションは、Web サービス・クライアントによるビジネス・プロセスのアクセスを可能にします。

Web サービス・バインディング付きエクスポートは、外部 Web サービス・クライアントによる SCA コンポーネントのアクセスを可能にします。Web サービス・バインディング付きエクスポートを作成するには、以下を行います。

1. マイグレーション・ウィザードによって作成されたモジュールをアセンブリ・エディターで開きます。
2. JMS バインディングが WebSphere Studio Application Developer Integration Edition に生成された BPEL プロセス・インターフェースごとに、SCA バインディング付きエクスポートを作成します。
 - a. アセンブリ・エディターで BPEL コンポーネントを右クリックします。
 - b. 「エクスポート...」を選択します。
 - c. 「Web サービス・バインディング」を選択します。
 - d. プロセスに複数のインターフェースがある場合は、このバインディング・タイプでエクスポートするインターフェースを選択します。
 - e. トランSPORT、**soap/http** または **soap/jms** を選択します。
 - f. Web サービス付きエクスポートが作成されたら、アセンブリ・エディターと「プロパティ」ビューでエクスポートを選択し、「説明」コンテンツ・ペインを選択します。エクスポートの名前と説明がリストされ、必要であれば変更できます。
 - g. アセンブリ図を保管します。

JMS および JMS プロセス・バインディングのマイグレーション・オプション 5:

WebSphere Studio Application Developer Integration Edition JMS プロセス・バインディングの 5 番目のマイグレーション・オプションは、JMS クライアントによるビジネス・プロセスのアクセスを可能にします。

JMS バインディング付きエクスポートは、外部 JMS クライアントによる SCA コンポーネントのアクセスを可能にします。JMS バインディング付きエクスポートを作成するには、以下を行います。

1. BPEL サービスの場合、5.1 JMS プロセス・バインディングが標準 5.1 JMS バインディングとはまったく異なっていたため、新しいキュー・リソースを作成および参照する必要があります。非 BPEL サービスの場合、**JMSBinding.wsdl** および **JMSService.wsdl** という名前の WSDL ファイルを、生成された EJB プロジェクトの **ejbModule/META-INF** フォルダの下に該当するパッケージ内で見つけ、バインディングおよびサービス情報がそこに収集されていることを検査することによって、WebSphere Studio Application Developer Integration Edition 5.1 の JMS デプロイメント・コード用に選択した値を見つけることができます。バインディングからは、テキスト・メッセージとオブジェクト・メッセージのどちらが使用されたか、およびカスタム・データ・フォーマット・バインディングが使用されたかどうかを判別できます。これらが使用されていた場合には、6.0 の「**JMS バインディング付きエクスポート**」用のカスタム・データ・バインディングを作成することを検討する必要もあります。サービスからは、初期コンテキスト・ファクトリー、JNDI 接続ファクトリー名、JNDI 宛先名、および宛先スタイル (キュー) をつけることができます。
2. マイグレーション・ウィザードによって作成されたモジュールをアセンブリ・エディターで開きます。

3. アセンブリー・エディターで BPEL コンポーネントを右クリックして、JMS バインディングが WebSphere Studio Application Developer Integration Edition に生成された BPEL プロセス・インターフェースごとに、JMS バインディング付きエクスポートを作成します。
4. 「エクスポート...」を選択します。
5. 「JMS バインディング」を選択します。
6. プロセスに複数のインターフェースがある場合は、このバインディング・タイプでエクスポートするインターフェースを選択します。
7. 次のパネル (JMS エクスポート・バインディング属性) で、「JMS メッセージング・ドメイン」を選択します。この属性を「Point-to-Point」として定義してください。
8. 「ビジネス・オブジェクトおよび JMS メッセージ間でのデータのシリアル化方法 (how data is serialized between Business Object and JMS Message)」を選択し、以下の値を入力します (「オブジェクト」ではなく、「テキスト」を選択することが推奨されます。テキストは、通常 XML であり、ランタイムから独立しており、異種システム間のサービス統合を可能にするためです)。
 - a. 「テキスト」に「デフォルト JMS 関数セクター (Default JMS function selector)」を選択するか、FunctionSelector 実装クラスの完全修飾名を入力します。
 - b. 「オブジェクト」に「デフォルト JMS 関数セクター (Default JMS function selector)」を選択するか、FunctionSelector 実装クラスの完全修飾名を入力します。
 - c. 「ユーザー提供」に JMSDataBinding 実装クラスの完全修飾名を入力します。ご使用のアプリケーションが、「JMS インポート・バインディング」ですぐには使用できない JMS ヘッダー・プロパティへのアクセスを必要とする場合は、「ユーザー提供」を選択する必要があります。この場合には、標準 JMS データ・バインディング `com.ibm.websphere.sca.jms.data.JMSDataBinding` を拡張するカスタム・データ・バインディング・クラスを作成し、カスタム・コードを追加して JMSMessage に直接アクセスする必要があります。その後で、カスタム・クラスの名前をこのフィールドに入力します。下のリンク先にある『インポートおよびエクスポート・コンポーネントのバインディングの作成および変更』で JMS の例を参照してください。
 - d. 「ユーザー提供」に「デフォルト JMS 関数セクター (Default JMS function selector)」を選択するか、FunctionSelector 実装クラスの完全修飾名を入力します。
9. JMS バインディング付きエクスポートが作成されたら、アセンブリー・エディターと「プロパティ」ビューでエクスポートを選択し、「説明」コンテンツ・ペインを選択します。エクスポートの名前と説明がリストされ、必要であれば変更できます。
10. 「バインディング」コンテンツ・ペインを選択すると、さらに多くのオプションが表示されます。
11. アセンブリー図を保管します。

IBM Web サービス・バインディング (SOAP/JMS) のマイグレーション:

BPEL プロセスまたは他のサービス・タイプ用 IBM Web サービス・バインディング (SOAP/JMS) は、推奨される SCA 構成にマイグレーションできます。

WebSphere Studio Application Developer Integration Edition では、このバインディング・タイプは、クライアントに対して、IBM Web サービスを呼び出すことによって BPEL プロセスまたは他のサービス・タイプと通信する機能を与えました。この場合、通信プロトコルは JMS でメッセージは SOAP エンコード・ルールに準拠していました。

以下は、5.1 BPEL サービス用 IBM Web サービス (SOAP/JMS) を生成する際に使用される規則の例です。生成済み IBM Web サービスの JNDI 名は、BPEL の名前、ターゲット名前空間、および開始日付タイム・スタンプの組み合わせであり、インターフェース (デプロイメント・コードが生成された対象の WSDL ポート・タイプ) の名前でもありました。例えば、これらの属性は、「説明」および「サーバー」

のコンテンツ・タブで、BPEL エディター内で BPEL プロセスのプロパティを調べることによって見つけることができます。

表 6. 生成済み名前空間

プロセス名	MyService
ターゲット名前空間	http://www.example.com/process87787141/
開始日付	Jan 01 2003 02:03:04
インターフェース	ProcessPortType

この例の生成済み名前空間は com/example/www/process87787141/MyService20030101T020304_ProcessPortTypePT です。

WebSphere Studio Application Developer Integration Edition では、IBM Web サービス・バインディング (SOAP/JMS) が BPEL プロセスまたは他のサービス・タイプのデプロイメント・タイプとして選択されると、以下のオプションが与えられました。

- 文書スタイルのデフォルトは、「**DOCUMENT / その他のオプション: RPC (DOCUMENT / other option: RPC)**」です。
- 文書使用のデフォルトは、「**LITERAL / その他のオプション: ENCODED (LITERAL / other option: ENCODED)**」です。
- JNDI プロバイダー URL の場合、これは「**サーバー提供 (Server supplied)**」または「**カスタム**」のどちらかになります (アドレスは入力する必要があり、デフォルトは iiop://localhost:2809 です)。
- 宛先スタイルのデフォルトは、「**キュー / その他のオプションはトピック (queue / other option was topic)**」です。
- JNDI 接続ファクトリーの場合、デフォルトは「**jms/qcf**」です (これは、生成済み MDB キューのキュー接続ファクトリーの JNDI 名です)。
- JNDI 宛先キューの場合、デフォルトは「**jms/queue**」です (これは、生成済み MDB キューの JNDI 名です)。
- MDB リスナー・ポートの場合、デフォルトは **<サービス・プロジェクト名>MdbListenerPort** でした。

IBM Web サービス SOAP/JMS バインディングとサービスを指定する WSDL ファイルは、サービス・プロジェクト自体ではなく、生成済み EJB プロジェクトに作成されます。これはつまり、IBM Web サービス・クライアント・コードを変更する必要がある場合は、このファイルを手動で配置して、ビジネス統合モジュール・プロジェクトにコピーしなければならないということです。デフォルトでは、この WSDL ファイルは、ejbModule/META-INF/wsd1/<ビジネス・プロセス名>_<ビジネス・プロセス・インターフェース・ポート・タイプ名>_JMS.wsd1 にある EJB プロジェクトの中に作成されました。

ビジネス・プロセス・インターフェースの WSDL ポート・タイプとメッセージは、サービス・プロジェクトに定義された既存の WSDL ポート・タイプとメッセージを参照するのではなく、実際にはこの WSDL ファイルにもコピーされます。

IBM Web サービス・クライアント・コードをマイグレーション後も変更しないでおく必要がある場合は、このファイルの情報が次の手動マイグレーション・ステップに必要です。

WebSphere Studio Application Developer Integration Edition SOAP/JMS プロセス・バインディングのマイグレーションには、2 つのオプションがあります。クライアントを SCA プログラミング・モデルにマイグレーションするか、web サービス・クライアントとして残すかを選択する必要があります。

注: 手動マイグレーション・ステップの完了後、クライアントも新しいプログラミング・モデルにマイグレーションする必要があります。次のクライアント・タイプについては、該当するトピックを参照してください。

表 7. クライアントのマイグレーションに関する詳細情報

クライアント・タイプ	詳細については以下を参照
IBM Web サービス・クライアント	IBM Web サービス (SOAP/JMS) クライアントのマイグレーション

IBM Web サービス・バインディング (SOAP/JMS) のマイグレーション・オプション 1:

WebSphere Studio Application Developer Integration Edition SOAP/JMS バインディングの最初のマイグレーション・オプションは、Web サービス・クライアントによるサービスのアクセスを可能にすることです。

Web サービス・バインディング付きエクスポートは、外部 Web サービス・クライアントによる SCA コンポーネントのアクセスを可能にします。Web サービス・バインディング付きエクスポートを作成するには、以下を行います。

1. マイグレーション・ウィザードによって作成されたモジュールをアセンブリー・エディターで開きます。
2. WebSphere Studio Application Developer Integration Edition で、サービス・インターフェース用に生成された IBM Web サービス (SOAP/JMS) バインディングを持つそれぞれのサービス・インターフェースごとに、SCA バインディング付きエクスポートを作成します。
 - a. アセンブリー・エディターで SCA コンポーネントを右クリックします。
 - b. 「エクスポート...」を選択します。
 - c. 「Web サービス・バインディング」を選択します。
 - d. コンポーネントに複数のインターフェースがある場合は、このバインディング・タイプでエクスポートするインターフェースを選択します。
 - e. トランスポート **soap/jms** を選択します。
3. Web サービス・エクスポートが作成されたら、アセンブリー・エディターと「プロパティ」ビューでエクスポートを選択し、「説明」コンテンツ・ペインを選択します。エクスポートの名前と説明がリストされ、必要であれば変更できます。
4. アセンブリー図を保管します。
5. 「バインディング」コンテンツ・ペインを選択すると、IBM Web サービス WSDL バインディングおよびサービスが、モジュールのプロジェクト・フォルダーに直接生成されているのが分かります。これは、エクスポートされたコンポーネント **Export WSDL ポート・タイプ名 Jms_Service.wsdl** と命名されます。このファイルを調べると、文書/リテラル・ラップ済みバインディングがデフォルトで使用されていることが分かります。これは、6.0 の優先スタイルです。これは、IBM Web サービス・クライアントがサービスの呼び出しに使用する WSDL です。
6. クライアント・コードの保存を希望する場合、以下のステップに従って、新規の Web サービス・バインディングとサービスを生成します。
 - a. **ejbModule/META-INF/wsdl/ビジネス・プロセス名/ビジネス・プロセス・インターフェース・ポート・タイプ名 JMS.wsdl** にある 5.1 で生成された EJB プロジェクトから、5.1 WSDL ファイルをビジネス・インテグレーション・モジュール・プロジェクトにコピーします。
 - b. ファイルをコピーして、モジュールを再ビルドした後、Web サービスによって使用された XML スキーマ・タイプ、WSDL メッセージ、および WSDL ポート・タイプが、5.1 の IBM Web サービス WSDL ファイルで重複するために、エラー・メッセージが表示されることがあります。このエラ

ーを修正するには、これらの重複する定義を IBM Web サービスのバインディング/サービス WSDL から削除して、その場所に、実際のインターフェース WSDL 用の WSDL インポートを追加します。注: WebSphere Studio Application Developer Integration Edition が IBM Web サービスのデプロイメント・コードを生成したときに、一部のケースでスキーマ定義を変更することに注意することが重要です。これにより、IBM Web サービス WSDL を使用する既存のクライアントで、不整合が発生する可能性があります。例えば、「elementFormDefault」スキーマ属性は、オリジナルのスキーマ定義が qualified ではなくても、IBM Web サービス WSDL で生成されたインライン・スキーマでは「qualified」に設定されます。これによって、実行時に「WSWS3047E: エラー: エlementをデシリアライズできません (WSWS3047E: Error: Cannot deserialize element)」というエラーが生成されます。

- c. ビジネス・インテグレーション・モジュールにコピーしたばかりのこの WSDL ファイルを右クリックして、「アプリケーションから開く」を選択し、次に「WSDL エディター」を選択します。
- d. 「ソース」タブに進みます。このファイルの WSDL ポート・タイプおよび定義されたメッセージをすべて削除します。
- e. これで、「<binding>」バインディングに指定した「<portType>」ポート・タイプが未定義です (The「<portType>」port type specified for the「<binding>」binding is undefined)」というエラーが表示されます。エラーを修正するには、「グラフ」タブの WSDL エディターで「インポート」セクションを右クリックし、「インポートの追加」を選択します。
- f. 「一般」タブの「プロパティー」ビューで、「ロケーション」フィールドの右側にある「...」ボタンをクリックします。WSDL メッセージとポート・タイプ定義があるインターフェース WSDL を参照し、「OK」をクリックして、インターフェース WSDL をサービス/バインディング WSDL にインポートします。
- g. WSDL ファイルを保管します。
- h. プロジェクトの更新/再ビルドを行います。ビジネス・インテグレーション・パースペクティブに切り替えます。モジュールのアセンブリー図をアセンブリー・エディターで開きます。
- i. プロジェクト・エクスプローラー・ビューで、マイグレーションするモジュールを展開して、「Web サービス・ポート」論理カテゴリを展開します。これで、バインディング/サービス WSDL に存在するポートがリスト表示されます。これをアセンブリー・エディターにドラッグ・アンド・ドロップします。
- j. **Web サービス・バインディング付きエクスポート**を作成することを選択して、該当するポート名を選択します。これにより、古いバインディング/サービスを使用するエクスポートが作成され、既存の Web サービス・クライアントを変更する必要はありません。アセンブリー・エディターで作成したばかりのエクスポートを選択して「プロパティー」ビューに進んだ場合、「バインディング」タブには、5.1 のポートとサービスの名前がすでに入力されていることが分かります。
- k. 変更をすべて保管します。
- l. アプリケーションをデプロイする直前に、生成された Web プロジェクトの構成を、5.1 のサービス・アドレスに一致するように変更できます (これらの変更は、このファイルが再生成される原因となった SCA モジュールの変更を行うたびに実行する必要があります)。5.1 から再使用している IBM Web サービス WSDL サービス 定義を調べている場合は、5.1 のクライアントで
`<wsdl:soap:address location="http://localhost:9080/MyServiceWeb/services/MyServicePort"/>`
とコーディングされたサービス・アドレスが表示されます。
- m. 6.0 で生成された Web プロジェクト成果物がこの古いサービス・アドレスと一致するようにするには、生成された Web プロジェクトのデプロイメント記述子を変更する必要があります。WebSphere Integration Developer でデプロイメント記述子を開き、「サーブレット (Servlets)」タブで、そのエクスポート用の既存の URL マッピングと非常に良く似た追加の URL マッピングを、同じサーブレット名で異なる URL パターンを指定して追加します。

- n. また、オリジナルのサービス・アドレスのコンテキスト・ルート（この例では、コンテキスト・ルートは「MyServiceWeb」です）と一致するように、この Web プロジェクトのコンテキスト・ルートを変更する必要がある場合は、この Web プロジェクトが入っている J2EE エンタープライズ・アプリケーションのデプロイメント記述子を開いて、古いサービス・アドレスのコンテキスト・ルートと一致するように、その Web モジュールのコンテキスト・ルートを変更します。「CHKJ3017E: Web プロジェクト: <WEB PROJ NAME> が、EAR プロジェクト: <APP NAME> 内の無効なコンテキスト・ルート: <NEW CONTEXT ROOT> にマップされています。(CHKJ3017E: Web Project: <WEB PROJ NAME> is mapped to an invalid Context root: <NEW CONTEXT ROOT> in EAR Project: <APP NAME>.)」というエラーが表示されますが、無視できます。

IBM Web サービス・バインディング (SOAP/JMS) のマイグレーション・オプション 2:

WebSphere Studio Application Developer Integration Edition SOAP/JMS プロセス・バインディングの 2 番目のマイグレーション・オプションは、ビジネス・プロセスによる SCA 以外のエンティティ（例えば JSP または Java クライアント）のアクセスを可能にします。

スタンドアロン参照は、外部クライアントによる SCA コンポーネントのアクセスを可能にします。スタンドアロン参照を作成するには、以下を行います。

1. マイグレーション・ウィザードによって作成されたモジュールをアセンブリ・エディターで開きます。
2. IBM Web サービス (SOAP/JMS) バインディングが WebSphere Studio Application Developer Integration Edition に生成された BPEL プロセス・インターフェースごとに、スタンドアロン参照を作成します。
 - a. ツールバーから「**スタンドアロン参照**」項目を選択します。
 - b. アセンブリ・エディターのキャンバスをクリックして、スタンドアロン参照 SCA エンティティを作成します。
 - c. ツールバーから「**ワイヤー**」項目を選択します。
 - d. 「**スタンドアロン参照**」エンティティをクリックして、ワイヤーのソースとして選択します。
 - e. 「**BPEL SCA**」コンポーネントをクリックして、ワイヤーのターゲットとして選択します。
 - f. アラート「ソース・ノード上にマッチング参照が作成されます。続きますか? (Matching reference will be created on the source node. Would you like to continue?)」が表示されたら、「**OK**」をクリックします。
 - g. 作成したばかりの「**スタンドアロン参照**」エンティティを選択して、「プロパティ」ビューで「**説明**」コンテンツ・ペインを選択します。
 - h. 「**参照**」リンクを展開して、いま作成したリファレンスを選択します。リファレンスの名前と説明がリストされ、必要であれば変更できます。
 - i. プロセスに複数のインターフェースがある場合は、このバインディング・タイプでエクスポートするインターフェースを選択します。
 - j. アセンブリ図を保管します。

IBM Web サービス・バインディング (SOAP/HTTP) のマイグレーション:

BPEL プロセスまたは他のサービス・タイプ用 IBM Web サービス・バインディング (SOAP/HTTP) は、推奨される SCA 構成にマイグレーションできます。

WebSphere Studio Application Developer Integration Edition では、このバインディング・タイプは、クライアントに対して、IBM Web サービスを呼び出すことによって BPEL プロセスまたは他のサービス・タイプと通信する機能を与えました。この場合、通信プロトコルは HTTP で、メッセージは SOAP エンコード・ルールに準拠していました。

以下は、5.1 BPEL サービス用 IBM Web サービス (SOAP/HTTP) を生成する際に使用される規則の例です。生成済み IBM Web サービスの JNDI 名は、BPEL の名前、ターゲット名前空間、および開始日付タイム・スタンプの組み合わせであり、インターフェース (デプロイメント・コードが生成された対象の WSDL ポート・タイプ) の名前でもありました。例えば、これらの属性は、「説明」および「サーバー」のコンテンツ・タブで、BPEL エディター内で BPEL プロセスのプロパティを調べることによって見つけることができます。

表 8. 生成済み名前空間

プロセス名	MyService
ターゲット名前空間	http://www.example.com/process87787141/
開始日付	Jan 01 2003 02:03:04
インターフェース	ProcessPortType

この例の生成済み名前空間は com/example/www/process87787141/MyService20030101T020304_ProcessPortTypePT です。

WebSphere Studio Application Developer Integration Edition では、IBM Web サービス・バインディング (SOAP/HTTP) が BPEL プロセスまたは他のサービス・タイプのデプロイメント・タイプとして選択されると、以下のオプションが与えられました。

- 文書スタイルのデフォルトは、「**RPC / その他のオプション: DOCUMENT (RPC / other option: DOCUMENT)**」です。
- 文書使用のデフォルトは、「**ENCODED / その他のオプション: LITERAL (ENCODED / other option: LITERAL)**」です。
- ルーター・アドレスのデフォルトは、http://localhost:9080 です。

IBM Web サービス SOAP/HTTP バインディングとサービスを指定する WSDL ファイルは、サービス・プロジェクト自体ではなく、生成済み Web および EJB プロジェクトに作成されます。これはつまり、IBM Web サービス・クライアント・コードを変更する必要がある場合は、このファイルを手動で配置して、ビジネス統合モジュール・プロジェクトにコピーしなければならないということです。デフォルトでは、この WSDL ファイルは、WebContent/WEB-INF/wsdl/<ビジネス・プロセス名>_<ビジネス・プロセス・インターフェース・ポート・タイプ名>_HTTP.wsdl にある Web プロジェクトの中に作成されました。

ビジネス・プロセス・インターフェースの WSDL ポート・タイプとメッセージは、サービス・プロジェクトに定義された既存の WSDL ポート・タイプとメッセージを参照するのではなく、実際にはこの WSDL ファイルにもコピーされます。

IBM Web サービス・クライアント・コードをマイグレーション後も変更しないでおく必要がある場合は、このファイルの情報が次の手動マイグレーション・ステップに必要です。

WebSphere Studio Application Developer Integration Edition SOAP/HTTP プロセス・バインディングのマイグレーションには、2 つのオプションがあります。クライアントを SCA プログラミング・モデルにマイグレーションするか、Web サービス・クライアントとして残すかを選択する必要があります。

注: 手動マイグレーション・ステップの完了後、クライアントも新しいプログラミング・モデルにマイグレーションする必要があります。次のクライアント・タイプについては、該当するトピックを参照してください。

表 9. クライアントのマイグレーションに関する詳細情報

クライアント・タイプ	詳細については以下を参照
IBM Web サービス・クライアント	IBM Web サービス (SOAP/HTTP) クライアントのマイグレーション

IBM Web サービス (SOAP/HTTP) バインディングのマイグレーション・オプション 1:

WebSphere Studio Application Developer Integration Edition SOAP/HTTP プロセス・バインディングの最初のマイグレーション・オプションは、Web サービス・クライアントによるビジネス・プロセスのアクセスを可能にします。

Web サービス・バインディング付きエクスポートは、外部 Web サービス・クライアントによる SCA コンポーネントのアクセスを可能にします。Web サービス・バインディング付きエクスポートを作成するには、以下を行います。

1. マイグレーション・ウィザードによって作成されたモジュールをアセンブリ・エディターで開きます。
2. アセンブリ・エディターで BPEL コンポーネントを右クリックして、WebSphere Studio Application Developer Integration Edition で生成された IBM Web Service (SOAP/HTTP) バインディングのある BPEL プロセスのインターフェースごとに「SCA バインディング付きエクスポート (Export with SCA Binding)」を作成します。
3. 「エクスポート...」を選択します。
4. 「Web サービス・バインディング」を選択します。
5. コンポーネントに複数のインターフェースがある場合は、このバインディング・タイプでエクスポートするインターフェースを選択します。
6. トランスポート **soap/http** を選択します。
7. Web サービス・エクスポートが作成されたら、アセンブリ・エディターと「プロパティ」ビューでエクスポートを選択し、「説明」コンテンツ・ペインを選択します。エクスポートの名前と説明がリストされ、必要であれば変更できます。
8. アセンブリ図を保管します。
9. クライアント・コードの保存を希望する場合、以下のステップに従って、新規の Web サービス・バインディングとサービスを生成します。
 - a. ejbModule/META-INF/wsdl/ビジネス・プロセス名/ビジネス・プロセス・インターフェース・ポート・タイプ名_HTTP.wsdl にある 5.1 で生成された EJB プロジェクトから、5.1 の WSDL ファイルをビジネス・インテグレーション・モジュール・プロジェクトにコピーします。
 - b. ファイルをコピーして、モジュールを再ビルドした後、Web サービスによって使用される XML スキーマ・タイプ、WSDL メッセージ、および WSDL ポート・タイプが、5.1 の IBM Web サービス WSDL ファイルで重複するために、エラー・メッセージが表示されることがあります。このエラーを修正するには、これらの重複する定義を IBM Web サービスのバインディング/サービス WSDL から削除して、その場所に、実際のインターフェース WSDL 用の WSDL インポートを追加します。注: WebSphere Studio Application Developer Integration Edition が IBM Web サービスのデプロイメント・コードを生成したときに、一部のケースでスキーマ定義を変更することに注意することが重要です。これにより、IBM Web サービス WSDL を使用する既存のクライアントで、不整合

が発生する可能性があります。例えば、「elementFormDefault」スキーマ属性は、オリジナルのスキーマ定義が qualified ではなくても、IBM Web サービス WSDL で生成されたインライン・スキーマでは「qualified」に設定されます。これによって、実行時に「WSWS3047E: エラー: エレメントをデシリアライズできません (WSWS3047E: Error: Cannot deserialize element)」というエラーが生成されます。

- c. ビジネス・インテグレーション・モジュールにコピーしたばかりのこの WSDL ファイルを右クリックして、「アプリケーションから開く」を選択し、次に「WSDL エディター」を選択します。
- d. 「ソース」タブに進みます。このファイルの WSDL ポート・タイプおよび定義されたメッセージをすべて削除します。
- e. これで、「<binding>」バインディングに指定した「<portType> ポート・タイプが未定義です (The '<portType>' port type specified for the '<binding>' binding is undefined)」というエラーが表示されます。エラーを修正するには、「グラフ」タブの WSDL エディターで「インポート」セクションを右クリックし、「インポートの追加」を選択します。
- f. 「一般」タブの「プロパティ」ビューで、「ロケーション」フィールドの右側にある「...」ボタンをクリックします。WSDL メッセージとポート・タイプ定義があるインターフェース WSDL を参照し、「OK」をクリックして、インターフェース WSDL をサービス/バインディング WSDL にインポートします。
- g. WSDL ファイルを保管します。
- h. プロジェクトの更新/再ビルドを行います。ビジネス・インテグレーション・パースペクティブに切り替えます。モジュールのアセンブリー図をアセンブリー・エディターで開きます。
- i. プロジェクト・エクスプローラー・ビューで、マイグレーションするモジュールを展開して、「Web サービス・ポート」論理カテゴリーを展開します。これで、バインディング/サービス WSDL に存在するポートがリスト表示されます。これをアセンブリー・エディターにドラッグ・アンド・ドロップします。
- j. **Web サービス・バインディング付きエクスポート**を作成することを選択して、該当するポート名を選択します。これにより、古いバインディング/サービスを使用するエクスポートが作成され、既存の Web サービス・クライアントを変更する必要はありません。アセンブリー・エディターで作成したばかりのエクスポートを選択して「プロパティ」ビューに進んだ場合、「バインディング」タブには、5.1 のポートとサービスの名前がすでに入力されていることが分かります。
- k. 変更をすべて保管します。
- l. アプリケーションをデプロイする直前に、生成された Web プロジェクトの構成を、5.1 のサービス・アドレスに一致するように変更できます (これらの変更は、このファイルが再生成される原因となった SCA モジュールの変更を行うたびに実行する必要があります)。5.1 から再使用している IBM Web サービス WSDL サービス定義を調べている場合は、5.1 のクライアントで
`<wsdlsoap:address location="http://localhost:9080/MyServiceWeb/services/MyServicePort"/>`
とコーディングされたサービス・アドレスが表示されます。
- m. 6.0 で生成された Web プロジェクト成果物がこの古いサービス・アドレスと一致するようにするには、生成された Web プロジェクトのデプロイメント記述子を変更する必要があります。WebSphere Integration Developer でデプロイメント記述子を開き、「サーブレット (Servlets)」タブで、そのエクスポート用の既存の URL マッピングと非常に良く似た追加の URL マッピングを、同じサーブレット名で異なる URL パターンを指定して追加します。
- n. また、オリジナルのサービス・アドレスのコンテキスト・ルート (この例では、コンテキスト・ルートは「MyServiceWeb」です) と一致するように、この Web プロジェクトのコンテキスト・ルートを変更する必要がある場合は、この Web プロジェクトが入っている J2EE エンタープライズ・アプリケーションのデプロイメント記述子を開いて、古いサービス・アドレスのコンテキスト・ルートと一

致するように、その Web モジュールのコンテキスト・ルートを変更します。「CHKJ3017E: Web プロジェクト: <WEB PROJ NAME> が、EAR プロジェクト: <APP NAME> 内の無効なコンテキスト・ルート: <NEW CONTEXT ROOT> にマップされています。(CHKJ3017E: Web Project: <WEB PROJ NAME> is mapped to an invalid Context root: <NEW CONTEXT ROOT> in EAR Project: <APP NAME>.)」というエラーが表示されますが、無視できます。

IBM Web サービス (SOAP/HTTP) バインディングのマイグレーション・オプション 2:

WebSphere Studio Application Developer Integration Edition SOAP/HTTP プロセス・バインディングの 2 番目のマイグレーション・オプションは、ビジネス・プロセスによる SCA 以外のエンティティ (例えば JSP または Java クライアント) のアクセスを可能にします。

スタンドアロン参照は、外部クライアントによる SCA コンポーネントのアクセスを可能にします。スタンドアロン参照を作成するには、以下を行います。

1. マイグレーション・ウィザードによって作成されたモジュールをアセンブリ・エディターで開きます。
2. IBM Web サービス (SOAP/HTTP) バインディングが WebSphere Studio Application Developer Integration Edition に生成されたインターフェースごとに、スタンドアロン参照を作成します。
 - a. ツールバーから「**スタンドアロン参照**」項目を選択します。
 - b. アセンブリ・エディターのキャンバスをクリックして、スタンドアロン参照 SCA エンティティを作成します。
 - c. ツールバーから「**ワイヤー**」項目を選択します。
 - d. 「**スタンドアロン参照**」エンティティをクリックして、ワイヤーのソースとして選択します。
 - e. 「**SCA**」コンポーネントをクリックして、ワイヤーのターゲットとして選択します。
 - f. アラート「**ソース・ノード上にマッチング参照が作成されます。継続しますか? (Matching reference will be created on the source node. Would you like to continue?)**」が表示されたら、「**OK**」をクリックします。
 - g. 作成したばかりの「**スタンドアロン参照**」エンティティを選択して、「**プロパティ**」ビューで「**説明**」コンテンツ・ペインを選択します。
 - h. 「**参照**」リンクを展開して、いま作成したリファレンスを選択します。リファレンスの名前と説明がリストされ、必要であれば変更できます。
 - i. プロセスに複数のインターフェースがある場合は、このバインディング・タイプでエクスポートするインターフェースを選択します。
 - j. アセンブリ図を保管します。

Apache Web サービス・バインディング (SOAP/HTTP) のマイグレーション:

BPEL プロセスまたは他のサービス・タイプ用の Apache Web サービス・バインディング (SOAP/HTTP) は、推奨される SCA 構成にマイグレーションできます。

WebSphere Studio Application Developer Integration Edition では、このバインディング・タイプは、クライアントに対して、Apache Web サービスを呼び出すことによって BPEL プロセスまたは他のサービス・タイプと通信する機能を与えました。

WebSphere Studio Application Developer Integration Edition では、Apache Web サービス・バインディングが BPEL プロセスまたは他のサービス・タイプのデプロイメント・タイプとして選択されると、以下のオプションが与えられました。

- 文書スタイルでは **RPC** (他のオプションは使用できません) です。
- SOAP アクションでは **URN:WSDL** ポート・タイプ名 です。
- アドレスでは `http://localhost:9080/サービス・プロジェクト名/Web/servlet/rpcrouter` です。
- 使用エンコードのデフォルトは **はい** です (はい の場合、エンコード・スタイルは `http://schemas.xmlsoap.org/soap/encoding/` に設定されます)。

Apache SOAP バインディングとサービスを指定する WSDL ファイルは、サービス・プロジェクトに作成されます。デフォルトでは、このファイルは、`<ビジネス・プロセス名>_<ビジネス・プロセス・インターフェース・ポート・タイプ名>_SOAP.wsdl` という名前でラッピングされた、サービスと同じディレクトリに作成されます。ビジネス・プロセス・インターフェースの WSDL ポート・タイプとメッセージは、このバインディングとサービスによって、直接使用されます。マイグレーション後は、バージョン 6 で生成される新しい WSDL 内の同じ名前空間、ポート、およびサービス名を使用する以外は、この WSDL をどんなものにも使用してはなりません。

WebSphere Studio Application Developer Integration Edition Web サービス・バインディングのマイグレーションには、2 つのオプションがあります。クライアントを SCA プログラミング・モデルにマイグレーションするか、IBM Web サービス・プログラミング・モデルとして残すかを選択する必要があります。

Apache Web サービス (SOAP/HTTP) バインディング・タイプと同等のバインディングは、6 SCA プログラミング・モデルにはありません。

IBM Web サービス・エンジンを使用するには、この Apache Web サービスをマイグレーションする必要があります。このマイグレーションを実行して、IBM Web サービス (SOAP/HTTP) を作成する方法については、トピック『IBM Web サービス・バインディング (SOAP/HTTP) のマイグレーション』を参照してください。

SCA プログラミング・モデルへのマイグレーション:

WebSphere Studio Application Developer Integration Edition サービスと対話するフリー・フォーム Java コードについて、本セクションでは、WSIF プログラミング・モデルから新しい SCA プログラミング・モデル (ここでは、アプリケーションで使用されるデータが Eclipse Service Data Objects (SDO) に保管される) にマイグレーションする方法を説明します。また、もっとも一般的なクライアント・タイプを新しいプログラミング・モデルに手動でマイグレーションする方法についても説明します。

本セクションでは、Java Snippet が含まれている BPEL プロセスについて、古い Java Snippet API から、アプリケーションで使用されるデータが Eclipse Service Data Objects (SDO) に保管される新しい Java Snippet API へのマイグレーション方法を説明します。可能な場合はいつでも、snippet はマイグレーション・ウィザードによって自動的にマイグレーションされますが、マイグレーション・ウィザードが完全にはマイグレーションできない snippet があります。つまり、マイグレーションを完了するには人手によるステップが必要となります。

以下に、プログラミング・モデルの変更点を要約します。

V5.1 プログラミング・モデル

1. WSIF および WSDL ベース
2. サービスについて生成されたプロキシー
3. タイプの Bean およびフォーマット・ハンドラー

V6.0 プログラミング・モデル (より Java 中心)

1. ドックレット・タグを使用した SDO ベースの SCA サービス
2. サービス用インターフェース・バインディング

3. タイプ用 SDO およびデータ・バインディング

SDO API への WSIFMessage API 呼び出しのマイグレーション:

以下のセクションでは、古い WebSphere Business Integration Server Foundation バージョン 5.1 プログラミング・モデル (アプリケーションで使用されるデータが、強く型付けされた生成済みインターフェースを使用して WSIFMessage オブジェクトとして表される) から、新しい WPS バージョン 6.0 プログラミング・モデル (データが Service Data Object (SDO) として表され、強く型付けされたインターフェースは生成されない) にマイグレーションする方法について説明します。

表 10. SDO API への WSIFMessage API 呼び出しのマイグレーションに関する変更点およびソリューション

変更	ソリューション
WSDL メッセージ・タイプについて、WSIFMessage ベースのラッパー・クラスが生成されなくなり、複合スキーマ・タイプについて、Java Bean ヘルパー・クラスが生成されなくなりました。	SCA サービスと相互作用するコードを作成するときは、汎用 SDO API を使用して、アプリケーションで使用されるデータを保持する <code>commonj.sdo.DataObject</code> メッセージを操作する必要があります。 単一の単純タイプのパーツを持つ WSDL メッセージ定義は、実データにラッパーを使用する代わりに、直接そのパーツを表す単純 Java タイプによって表されるようになりました。単一メッセージ・パーツが複合タイプの場合、このデータは、複合タイプ定義に従う <code>DataObject</code> として表されます。 現在、複数パーツを持つ WSDL メッセージ定義は、すべてのメッセージ・パーツのプロパティを持つ <code>DataObject</code> に対応しています。この定義では、 <code>complexType</code> は親 <code>DataObject</code> の「reference-type」プロパティとして表され、 <code>complexType</code> には、 <code>getDataObject</code> および <code>setDataObject</code> メソッドを使用してアクセスできます。
WSIFMessage パーツの強く型付けされた getter メソッドおよび生成された Java Bean は、使用してはなりません。	<code>DataObject</code> プロパティを取得するには、弱く型付けされた SDO API を使用する必要があります。
BPEL 変数のメッセージ・パーツの強く型付けされた setter メソッドは、使用できなくなりました。	<code>DataObject</code> プロパティを設定するには、弱く型付けされた SDO API を使用する必要があります。
WSIFMessage プロパティの弱く型付けされた getter メソッドは、使用できなくなりました。	<code>DataObject</code> プロパティを設定するには、弱く型付けされた SDO API を使用する必要があります。
WSIFMessage プロパティの弱く型付けされた setter メソッドは、使用されなくなりました。	<code>DataObject</code> プロパティを設定するには、弱く型付けされた SDO API を使用する必要があります。
可能な場合は、すべての WSIFMessage API 呼び出しを SDO API にマイグレーションします。	可能な場合は、この呼び出しを同等の SDO API 呼び出しにマイグレーションします。可能でない場合は、ロジックを再設計します。

WebSphere Business Integration Server Foundation クライアント・コードのマイグレーション:

本セクションでは、WebSphere Business Integration Server Foundation 5.1 サービス・タイプとして考えられるさまざまなクライアント・タイプをマイグレーションする方法を説明します。

EJB クライアントのマイグレーション:

このトピックでは、EJB インターフェースを使用してサービスを呼び出すクライアントのマイグレーション方法を示します。

1. SCA バインディング付きエクスポートを、マイグレーション済みのモジュールからこの新しいモジュールのアセンブリ・エディターにドラッグ・アンド・ドロップします。これにより、SCA バインディング付きインポートが作成されます。クライアントがこのインポートを参照できるようにするには、スタンドアロン参照を作成する必要があります。
2. パレットでスタンドアロン参照項目を選択します。アセンブリ・エディターのキャンバスを一回クリックして、この新規モジュールの新しいスタンドアロン参照を作成します。
3. ワイヤ・ツールを選択し、サービス参照をクリックしてから「インポート」をクリックします。
4. ソース・ノード上にマッチング参照が作成されるというアラートが表示されたら、「OK」をクリックします。
5. 「Java クライアントが Java インターフェースを使用するには、この参照を使用するほうが簡単です。WSDL 参照を互換性のある Java 参照に変換しますか? (It is easier for a Java client to use a Java interface with this reference - would you like to convert the WSDL reference to a compatible Java reference?)」という質問が出されます。
 - a. クライアントがこのサービスをルックアップして Java クラスとしてキャストし、Java インターフェースを使用して呼び出すようにするには、「はい」と答えます。この新しい Java インターフェースは WSDL ポート・タイプから名前を取得します。インターフェースのパッケージは WSDL ポート・タイプの名前空間から派生します。WSDL ポート・タイプに定義された操作ごとにメソッドが定義され、それぞれの WSDL メッセージ・パーツはインターフェース・メソッドに対する引数として表されます。
 - b. クライアントがこのサービスをルックアップし、汎用 `com.ibm.websphere.sca.Service` インターフェースを使用して、これを汎用 SCA サービスとしての呼び出し操作で呼び出すようにするには、「いいえ」と答えます。
6. アセンブリ・エディターでスタンドアロン参照のコンポーネントを選択して、スタンドアロン参照の名前をより分かりやすい名前にします (該当する場合)。「プロパティ」ビューの「詳細」タブに移動し、作成した参照にドリルダウンし、それを選択して名前を変更します。
`com.ibm.websphere.sca.ServiceManager` インスタンスの `locateService` メソッドを呼び出すときに、クライアントがこの名前を使用する必要があるため、この参照で選択した名前を覚えておいてください。
7. 「保管」をクリックして、アセンブリ図を保管します。

サーバーで実行されているマイグレーション済み EJB モジュールにアクセスするために、クライアントはこの新規モジュールをローカル・クラスパス上に持つ必要があります。

以下に、「CustomerInfo」タイプのサービスでのクライアント・コードの例を示します。

```
// 新規 ServiceManager を作成する
ServiceManager serviceManager = ServiceManager.INSTANCE;
// CustomerInfo サービスの位置を指定する
CustomerInfo customerInfoService = (CustomerInfo) serviceManager.locateService ("<name-of-standalone-
reference-from-previous-step");
// Invoke the CustomerInfo service
System.out.println(" [getMyValue] getting customer info...");
DataObject customer = customerInfoService.getCustomerInfo(customerID);
```

クライアントは、メッセージの構成方法を変更する必要があります。以前、メッセージは `WSIFMessage` クラスに基づいていましたが、現在は `commonj.sdo.DataObject` クラスに基づいています。

EJB プロセス・バインディング・クライアントのマイグレーション:

このトピックでは、WSIF EJB プロセス・バインディングを使用して BPEL サービスにアクセスするクライアントのマイグレーション方法を示します。

EJB プロセス・バインディングを使用してビジネス・プロセスを呼び出していたクライアントは、SCA API (マイグレーション済みビジネス・プロセスに SCA バインディング付きエクスポートがあることが必要) または IBM Web サービス・クライアント API (マイグレーション済みビジネス・プロセスに Web サービス・バインディング付きエクスポートがあることが必要) のいずれかを使用してサービスを呼び出すようになります。

このようなクライアントを生成する方法については、『EJB クライアントのマイグレーション』、『IBM Web サービス (SOAP/JMS) クライアントのマイグレーション』、または『IBM Web サービス (SOAP/HTTP) クライアントのマイグレーション』の各トピックを参照してください。

IBM Web サービス (SOAP/JMS) クライアントのマイグレーション:

このトピックでは、Web サービス API (SOAP/JMS) を使用してサービスを呼び出すクライアントのマイグレーション方法を示します。

マイグレーション時に、既存のクライアントのマイグレーションを行う必要はありません。生成された Web プロジェクトを手動で変更する必要がある (新しいサブレット・マッピングを作成する)、WebSphere Business Integration Server Foundation で公開されていたアドレスとまったく同じアドレスにサービスを公開するために、エンタープライズ・アプリケーション・デプロイメント記述子で Web プロジェクトのコンテキスト・ルートを変更しなければならない場合もあることに注意してください。トピック『IBM Web サービス・バインディング (SOAP/JMS) のマイグレーション』を参照してください。

重要な点として、WSIF または RPC クライアント・プロキシーを生成することができたバージョン 5.1 とは異なり、バージョン 6.0 では、WSIF API ではなく RPC が 6.0 優先 API であるため、ツールは RPC クライアント生成しかサポートしないことに注意してください。

注: WebSphere Integration Developer から新しいクライアント・プロキシーを生成するには、WebSphere Process Server または WebSphere Application Server がインストールされている必要があります。

1. WebSphere Process Server または WebSphere Application Server がインストールされていることを確認してください。
2. 「リソース」または Java パースペクティブで、「**Web サービス・バインディング付きエクスポート (Export with Web Service Binding)**」に対応する WSDL ファイルを見つけて右クリックし、「**Web サービス**」→「**クライアントを生成**」を選択します (このウィザードは 5.1 のウィザードによく似ています)。
3. クライアント・プロキシー・タイプとして「**Java プロキシー**」を選択し、「**次へ**」をクリックします。
4. WSDL のロケーションを入力します。「**次へ**」をクリックします。
5. 次に、Web サービス・ランタイムおよびサーバー、J2EE バージョン、クライアント・タイプ (Java、EJB、Web、アプリケーション・クライアント) をはじめとした、ご使用のクライアント環境構成を指定するために、該当するオプションを選択する必要があります。「**次へ**」をクリックします。
6. 残りのステップを完了して、クライアント・プロキシーを生成します。

IBM Web サービス (SOAP/HTTP) クライアントのマイグレーション:

このトピックでは、Web サービス API (SOAP/HTTP) を使用してサービスを呼び出すクライアントのマイグレーション方法を示します。

マイグレーション時に、既存のクライアントのマイグレーションを行う必要はありません。生成された Web プロジェクトを手動で変更する必要がある (新しいサーブレット・マッピングを作成する)、WebSphere Business Integration Server Foundation で公開されていたアドレスとまったく同じアドレスにサービスを公開するために、エンタープライズ・アプリケーション・デプロイメント記述子で Web プロジェクトのコンテキスト・ルートを変更しなければならない場合もあることに注意してください。トピック『IBM Web サービス・バインディング (SOAP/HTTP) のマイグレーション』を参照してください。

設計の変更があったときに新しいクライアント・プロキシを生成したい場合は、次の手順に従ってください。重要な点として、WSIF または RPC クライアント・プロキシを生成することができたバージョン 5.1 とは異なり、バージョン 6.0 では、WSIF API ではなく RPC が 6.0 優先 API であるため、ツールは RPC クライアント生成しかサポートしないことに注意してください。

注: WebSphere Integration Developer から新しいクライアント・プロキシを生成するには、WebSphere Process Server または WebSphere Application Server がインストールされている必要があります。

1. WebSphere Process Server または WebSphere Application Server がインストールされていることを確認してください。
2. 「**Web サービス・バインディング付きエクスポート (Export with Web Service Binding)**」に対応する WSDL ファイルを選択して右クリックし、「**Web サービス**」→「**クライアントを生成**」を選択します (このウィザードは 5.1 のウィザードによく似ています)。
3. クライアント・プロキシ・タイプとして「**Java プロキシ**」を選択し、「**次へ**」をクリックします。
4. WSDL のロケーションを入力します。「**次へ**」をクリックします。
5. 次に、Web サービス・ランタイムおよびサーバー、J2EE バージョン、クライアント・タイプ (Java、EJB、Web、アプリケーション・クライアント) をはじめとした、ご使用のクライアント環境構成を指定するために、該当するオプションを選択する必要があります。「**次へ**」をクリックします。
6. 残りのステップを完了して、クライアント・プロキシを生成します。

Apache Web サービス (SOAP/HTTP) クライアントのマイグレーション:

Apache Web サービス・クライアント API は、WebSphere Integration Developer サービスの起動には適していません。IBM Web サービス (SOAP/HTTP) クライアント API を使用するには、クライアント・コードをマイグレーションする必要があります。

詳しくは、セクション『IBM Web Service (SOAP/HTTP) クライアント』を参照してください。

5.1 では、クライアント・プロキシが自動的に生成された場合、そのプロキシは WSIF API を使用してサービスと相互作用していました。6.0 では、WSIF API ではなく RPC が 6.0 優先 API であるため、ツールは RPC クライアント生成のみをサポートします。

注: WebSphere Integration Developer から新しいクライアント・プロキシを生成するには、WebSphere Process Server または WebSphere Application Server がインストールされている必要があります。

1. WebSphere Process Server または WebSphere Application Server がインストールされていることを確認してください。
2. 「**Web サービス・バインディング付きエクスポート (Export with Web Service Binding)**」に対応する WSDL ファイルを選択して右クリックし、「**Web サービス**」→「**クライアントを生成**」を選択します (このウィザードは 5.1 のウィザードによく似ています)。
3. クライアント・プロキシ・タイプとして「**Java プロキシ**」を選択し、「**次へ**」をクリックします。
4. WSDL のロケーションを入力します。「**次へ**」をクリックします。

- 次に、Web サービス・ランタイムおよびサーバー、J2EE バージョン、クライアント・タイプ (Java、EJB、Web、アプリケーション・クライアント) をはじめとした、ご使用のクライアント環境構成を指定するために、該当するオプションを選択する必要があります。「次へ」をクリックします。
- 残りのステップを完了して、クライアント・プロキシを生成します。

JMS クライアントのマイグレーション:

JMS API (JMS メッセージをキューに送信) を介して 5.1 サービスと通信するクライアントは、手動マイグレーションを必要とする場合があります。このトピックでは、JMS API (JMS メッセージをキューに送信) を使用してサービスを呼び出すクライアントのマイグレーション方法を示します。

前のステップで作成した「**JMS バインディング付きエクスポート (Export with JMS Binding)**」が、このテキスト・メッセージまたはオブジェクト・メッセージを変更せずに受け入れることができることを確認する必要があります。これを果たすために、カスタム・データ・バインディングを作成する必要がある場合があります。詳しくは、『JMS および JMS プロセス・バインディングのマイグレーション』のセクションを参照してください。

クライアントは、メッセージの構成方法を変更する必要があります。以前、メッセージは `WSIFMessage` クラスに基づいていましたが、現在は `commonj.sdo.DataObject` クラスに基づいています。この手動マイグレーションを行う方法について詳しくは、セクション『SDO API への `WSIFMessage` API 呼び出しのマイグレーション』を参照してください。

Business Process Choreographer 汎用 EJB API クライアントのマイグレーション:

このトピックでは、5.1 Process Choreographer 汎用 EJB API を使用して BPEL サービスを呼び出すクライアントのマイグレーション方法を示します。

メッセージ・フォーマットとして `DataObjects` を使用する汎用 EJB API の新しいバージョンがあります。クライアントは、メッセージの構成方法を変更する必要があります。以前、メッセージは `WSIFMessage` クラスに基づいていましたが、現在は `commonj.sdo.DataObject` クラスに基づいています。

`ClientObjectWrapper` が特定のメッセージ・フォーマットにメッセージ・ラッパーをまだ使用しているため、汎用 EJB API はあまり変更されませんでした。

```
Ex: DataObject dobj = myClientObjectWrapper.getObject();
String result = dobj.getInt("resultInt");
```

`WSIFMessage` オブジェクトを使用する古い汎用 EJB の JNDI 名は、以下のとおりです。

```
GenericProcessChoreographerEJB
JNDI Name: com/ibm/bpe/api/BusinessProcessHome
Interface: com.ibm.bpe.api.BusinessProcess
```

6.0 ではヒューマン・タスク操作を別々の EJB として使用できるため、2 つの汎用 EJB があります。これらの汎用 EJB の 6.0 JNDI 名は、以下のとおりです。

```
GenericBusinessFlowManagerEJB
JNDI Name: com/ibm/bpe/api/BusinessFlowManagerHome
Interface: com.ibm.bpe.api.BusinessFlowManager
HumanTaskManagerEJB
JNDI Name: com/ibm/task/api/TaskManagerHome
Interface: com.ibm.task.api.TaskManager
```

Business Process Choreographer 汎用 Messaging API クライアントおよび JMS プロセス・バインディング・クライアントのマイグレーション:

WebSphere Process Server 6.0 には、汎用 Messaging API がありません。『JMS および JMS プロセス・バインディングのマイグレーション』のセクションを参照して、ビジネス・プロセスをコンシューマーに公開するための別の方法を選択し、選択されたバインディングに応じてクライアントを書き直してください。

Business Process Choreographer Web クライアントのマイグレーション:

このトピックでは、5.1 Process Choreographer Web クライアント設定とカスタム JSP のマイグレーション方法を示します。

マイグレーション・ウィザードは 5.1 Web クライアント設定を保存しており、ヒューマン・タスク・エディターでこれらの設定を編集することはできません。WebSphere Integration Developer 6.0 を使用して、新しい Web クライアント設定および JSP を作成する必要があります。

Web クライアント変更のマイグレーション

5.1 では、Struts ベースの Web クライアントの JSP **Header.jsp** およびスタイル・シート **dwc.css** を変更することによって、そのルック・アンド・フィールを変更することができました。

6.0 の Web クライアント (名前を「**Business Process Choreographer エクスプローラー**」に変更) は、Struts ではなく Java Server Faces (JSF) に基づいているため、Web クライアント変更を自動的にマイグレーションすることはできません。そのため、このアプリケーションの 6.0 パージョンをカスタマイズする方法の詳細については、「Business Process Choreographer エクスプローラー」のドキュメンテーションを参照することをお勧めします。

ビジネス・プロセスおよびスタッフ・アクティビティーについてユーザー定義 JSP を定義することができます。Web クライアントはこれらの JSP を使用して、プロセスおよびアクティビティーの入出力メッセージを表示します。

これらの JSP は特に、以下の場合に役立ちます。

1. メッセージのデータ構造の使用可能度を向上させるために、メッセージに非プリミティブ・パーツがあるとき。
2. Web クライアントの能力を拡張したいとき。

6.0 プロセスの Web クライアント設定を指定するときに使用可能なオプションは数も種類も増えているため、以下のように WebSphere Integration Developer を使用して、マイグレーション済みプロセスおよびアクティビティーの Web クライアント設定を再設計する必要があります。

1. プロセス・キャンバスまたはプロセス内のアクティビティーを選択します。
2. 「プロパティー」ビューで、「クライアント」タブを選択し、Web クライアント設定を再設計します。
3. ユーザー定義 JSP をすべて手動でマイグレーションします。
 - a. プログラミング・モデルの変更点については、『SCA プログラミング・モデルへのマイグレーション』のセクションを参照してください。
 - b. Web クライアントは、汎用 API を使用してビジネス・プロセスと相互作用します。これらの汎用 API の呼び出しをマイグレーションする方法を示すセクションを参照してください。
4. プロセスについて、6.0 Web クライアント設定で新しい JSP の名前を指定します。

注: DataObjects はカスタム・マッピングを必要としないため、6.0 Business Process Choreographer エクスプローラーでは JSP のマッピングは必要ありません。

WebSphere Business Integration Server Foundation BPEL Java Snippet のマイグレーション:

Java Snippet が含まれている BPEL プロセスについて、本セクションでは、古い Java Snippet API から新しい Java Snippet API (ここでは、アプリケーションで使用されるデータが Eclipse Service Data Objects (SDO) として保管される) へのマイグレーション方法を詳しく説明します。

WSIFMessage から SDO への遷移に特有のマイグレーション・ステップについては、『SDO API への WSIFMessage API 呼び出しのマイグレーション』のセクションを参照してください。

可能な場合はいつでも、Snippet はマイグレーション・ウィザードによって自動的にマイグレーションされますが、マイグレーション・ウィザードが完全にはマイグレーションできない Snippet があります。この場合、マイグレーションを完了するには、追加の手動ステップが必要になります。手動でマイグレーションする必要がある Java Snippet のタイプについて詳しくは、『制限』のトピックを参照してください。これらの Snippet のいずれかが検出されると、マイグレーション・ウィザードは、自動的にマイグレーションできない理由を説明し、警告またはエラー・メッセージを発行します。

以下の表で、Process Choreographer バージョン 5.1 から 6.0 への BPEL Java Snippet プログラミング・モデルおよび API の変更点を詳しく説明します。

表 11. WebSphere Business Integration Server Foundation BPEL Java Snippet のマイグレーションに関する変更点およびソリューション

変更	ソリューション
WSDL メッセージ・タイプについて、WSIFMessage ベースのラッパー・クラスが生成されなくなり、複合スキーマ・タイプについて、Java Bean ヘルパー・クラスが生成されなくなりました。	<p>BPEL 変数は、名前により直接的にアクセスできます。WSDL メッセージ定義に単一パーツがある BPEL 変数は、実データにラッパーを使用する代わりに、直接そのパーツを表すようになったことに注意してください。メッセージ・タイプに複数のパーツがある変数は、これらのパーツに DataObject ラッパーを使用します (WebSphere Application Developer Integration Edition でのラッパーは、WSIFMessage でした)。</p> <p>6.0 Snippet では BPEL 変数を直接使用できるため、5.1 の場合よりローカル変数の必要が少なくなっています。</p> <p>BPEL 変数の強く型付けされた getter は、暗黙的にメッセージ・パーツの WSIFMessage ラッパー・オブジェクトを初期化していました。現在、WSDL メッセージ定義に単一のパーツしかない BPEL 変数には「ラッパー」オブジェクトはありません。この場合、BPEL 変数はそのパーツを直接表します (単一パーツが XSD 単純タイプで、BPEL 変数が java.lang.String、java.lang.Integer などの Java オブジェクト・ラッパー・タイプとして表される場合)。複数パーツ WSDL メッセージ定義を持つ BPEL 変数は、異なる方法で処理されます。依然としてパーツにラッパーがあり、この DataObject ラッパーが前の操作で設定されていない場合は、6.0 Java Snippet コードで明示的に初期化される必要があります。</p> <p>5.1 Snippet のローカル変数が BPEL 変数と同じ名前を持っている場合には、競合が発生する可能性があるため、可能であればこの状態を修正してください。</p>

表 11. WebSphere Business Integration Server Foundation BPEL Java Snippet のマイグレーションに関する変更点およびソリューション (続き)

変更	ソリューション
WSIFMessage オブジェクトは、BPEL 変数を表すために使用されなくなりました。	Java Snippet から呼び出されるカスタム Java クラスに WSIFMessage パラメーターがある場合、そのクラスは、DataObject を受け入れる/戻すようにマイグレーションする必要があります。
BPEL 変数の強く型付けされた getter メソッドは、使用できなくなりました。	変数は、名前により直接的にアクセスできます。 WSDL メッセージ定義に単一パーツがある BPEL 変数は、実データにラッパーを使用する代わりに、直接そのパーツを表すようになったことに注意してください。メッセージ・タイプに複数のパーツがある変数は、これらのパーツに DataObject ラッパーを使用します (WebSphere Application Developer Integration Edition でのラッパーは、WSIFMessage でした)。
BPEL 変数の強く型付けされた setter メソッドは、使用できなくなりました。	変数は、名前により直接的にアクセスできます。 WSDL メッセージ定義に単一パーツがある BPEL 変数は、実データにラッパーを使用する代わりに、直接そのパーツを表すようになったことに注意してください。メッセージ・タイプに複数のパーツがある変数は、これらのパーツに DataObject ラッパーを使用します (WebSphere Application Developer Integration Edition でのラッパーは、WSIFMessage でした)。

表 11. WebSphere Business Integration Server Foundation BPEL Java Snippet のマイグレーションに関する変更点およびソリューション (続き)

変更	ソリューション
<p>WSIFMessage を戻す BPEL 変数の弱く型付けされた getter メソッドは、使用できなくなりました。</p>	<p>変数は、名前により直接的にアクセスできます。 WSDL メッセージ定義に単一パーツがある BPEL 変数は、実データにラッパーを使用する代わりに、直接そのパーツを表すようになったことに注意してください。メッセージ・タイプに複数のパーツがある変数は、これらのパーツに DataObject ラッパーを使用します (WebSphere Application Developer Integration Edition でのラッパーは、WSIFMessage でした)。</p> <p>getVariableAsWSIFMessage メソッドには、2 つのバリエーションがあったことに注意してください。</p> <pre>getVariableAsWSIFMessage(String variableName) getVariableAsWSIFMessage(String variableName, boolean forUpdate)</pre> <p>Java Snippet アクティビティの場合、デフォルトのアクセスは読み取り/書き込みです。変数名のリストを使用して Snippet 内のコメントに @bpe.readOnlyVariables を指定することによって、デフォルトのアクセスを読み取り専用に変更することができます。例えば、以下のように、変数 B と変数 D を読み取り専用を設定することができます。</p> <pre>variableB.setString("/x/y/z", variableA.getString("/a/b/c")); // @bpe.readOnlyVariables names="variableA" variableD.setInt("/x/y/z", variableC.getInt("/a/b/c")); // @bpe.readOnlyVariables names="variableC"</pre> <p>さらに、Java Snippet が条件内にある場合、デフォルトでは、変数は読み取り専用ですが、@bpe.readWriteVariables... を指定することによって、変数を読み取り/書き込みにすることができます。</p>
<p>BPEL 変数の弱く型付けされた setter メソッドは、使用できなくなりました。</p>	<p>変数は、名前により直接的にアクセスできます。 WSDL メッセージ定義に単一パーツがある BPEL 変数は、実データにラッパーを使用する代わりに、直接そのパーツを表すようになったことに注意してください。メッセージ・タイプに複数のパーツがある変数は、これらのパーツに DataObject ラッパーを使用します (WebSphere Application Developer Integration Edition でのラッパーは、WSIFMessage でした)。</p>

表 11. WebSphere Business Integration Server Foundation BPEL Java Snippet のマイグレーションに関する変更点およびソリューション (続き)

変更	ソリューション
<p>BPEL 変数メッセージ・パーツの弱く型付けされた getter メソッドは単一パーツのメッセージには適切ではなく、複数パーツ・メッセージ用に変更されています。</p>	<p>BPEL 変数 (DataObject) プロパティの弱く型付けされた getter メソッドにマイグレーションします。</p> <p>WSDL メッセージ定義に単一パーツがある BPEL 変数の場合、BPEL 変数はそのパーツを直接表すこと、および、この変数には getter メソッドを使用せずに直接アクセスする必要があること注意してください。</p> <p>getVariablePartAsObject メソッドには、2 つのバリエーションがありました。</p> <pre>getVariablePartAsObject(String variableName, String partName) getVariablePartAsObject(String variableName, String partName,boolean forUpdate)</pre> <p>複数パーツ・メッセージの場合、6.0 の次のメソッドによって同等の機能が提供されています。</p> <pre>getVariableProperty(String variableName, QName propertyName);</pre> <p>6.0 では、読み取り専用アクセスに変数を使用するという概念はありません (これは、5.1 では、上記の最初のメソッドおよび forUpdate=' false' を使用する 2 番目のメソッドの場合に該当します)。変数は 6.0 Snippet で直接使用され、常に更新することが可能です。</p>
<p>BPEL 変数のメッセージ・パーツに弱く型付けされた setter メソッドは単一パーツのメッセージには適切ではなく、複数パーツ・メッセージ用に変更されています。</p>	<p>BPEL 変数 (DataObject) プロパティの弱く型付けされた setter メソッドにマイグレーションします。</p> <p>WSDL メッセージ定義に単一パーツがある BPEL 変数の場合、BPEL 変数はそのパーツを直接表すこと、および、この変数には setter メソッドを使用せずに直接アクセスする必要があること注意してください。</p> <p>次のメソッドの呼び出しをマイグレーションする必要があります。</p> <pre>setVariableObjectPart(String variableName, String partName, Object data)</pre> <p>複数パーツ・メッセージの場合、6.0 の次のメソッドによって同等の機能が提供されています。</p> <pre>setVariableProperty(String variableName, QName propertyName,Serializable value);</pre>
<p>BPEL パートナー・リンクの強く型付けされた getter メソッドは、使用できなくなりました。</p>	<p>BPEL パートナー・リンクの弱く型付けされた getter メソッドにマイグレーションします。</p>
<p>BPEL パートナー・リンクの強く型付けされた setter メソッドは、使用できなくなりました。</p>	<p>BPEL パートナー・リンクの弱く型付けされた setter メソッドにマイグレーションします。</p>

表 11. WebSphere Business Integration Server Foundation BPEL Java Snippet のマイグレーションに関する変更点およびソリューション (続き)

変更	ソリューション
BPEL 関連の強く型付けされた getter メソッドは、使用できなくなりました。	V5.1 Snippet: <pre>String corrSetPropStr = getCorrelationSetCorrSetAProperty CustomerName(); int corrSetPropInt = getCorrelationSetCorrSetBProperty CustomerId();</pre> V6.0 Snippet: <pre>String corrSetPropStr = (String) getCorrelationSetProperty("CorrSetA", new QName("CustomerName")); int corrSetPropInt = ((Integer) getCorrelationSetProperty("CorrSetB", new QName("CustomerId"))). intValue();</pre>
BPEL アクティビティ・カスタム・プロパティの弱く型付けされた getter メソッドに必要な追加パラメーター。	V5.1 Snippet: <pre>String val = getActivityCustomProperty ("propName");</pre> V6.0 Snippet: <pre>String val = getActivityCustomProperty ("name-of-current-activity", "propName");</pre>
BPEL アクティビティ・カスタム・プロパティの弱く型付けされた setter メソッドに必要な追加パラメーター。	V5.1 Snippet: <pre>String newVal = "new value"; setActivityCustomProperty("propName", newVal);</pre> V6.0 Snippet: <pre>String newVal = "new value"; setActivityCustomProperty("name-of-current- activity", "propName", newVal);</pre>
raiseFault(QName faultQName, Serializable message) メソッドはなくなりました。	可能な場合は、raiseFault(QName faultQName, String variableName) にマイグレーションします。可能でない場合は、raiseFault(QName faultQName) メソッドにマイグレーションするか、またはシリアル化可能オブジェクト用の新しい BPEL 変数を作成してください。

WebSphere Business Integration Adapter との相互作用のマイグレーション:

JMS クライアントが WebSphere Business Integration Adapter である場合、「エンタープライズ・サービス・ディスカバリー」ツールを使用して、JMS バインディング付きインポートを作成する必要があります。このインポートは、WebSphere Business Integration Adapter が要求する正確なフォーマットに SDO をシリアル化するために、特殊なデータ・バインディングを使用します。

エンタープライズ・サービス・ディスカバリー・ツールにアクセスするには、以下のようにします。

1. 「ファイル」 → 「新規」 → 「その他」 → 「ビジネス・インテグレーション」と進み、「エンタープライズ・サービス・ディスカバリー」を選択します。「次へ」をクリックします。
2. 「WBI アダプター成果物インポーター (WebSphere Business Integration Adapter Artifact Importer)」を選択します。「次へ」をクリックします。
3. WebSphere Business Integration Adapter の構成ファイル (.cfg)、およびアダプターが使用するビジネス・オブジェクトの XML スキーマが含まれるディレクトリーへのパスを入力します。「次へ」をクリックします。
4. 生成された照会を調べて、正しければ「照会の実行」をクリックします。「照会で検出されたオブジェクト」リストで、追加するオブジェクトを (1 つずつ) 選択して、「>> 追加」ボタンをクリックします。
5. ビジネス・オブジェクト用の構成パラメーターを受け入れて、「OK」をクリックします。
6. ビジネス・オブジェクトのそれぞれについて繰り返します。
7. 「次へ」をクリックします。
8. 「ランタイム・ビジネス・オブジェクト・フォーマット」で、「SDO」を選択します。「ターゲット・プロジェクト」で、マイグレーションしたばかりのモジュールを選択します。「フォルダー」フィールドはブランクのままにしてください。
9. 「終了」をクリックします。

このツールは、古い XSD を、特殊データ・バインディングが期待するフォーマットにマイグレーションします。そのため、古い WebSphere Business Integration Adapter の XSD をモジュールから除去して、新しい XSD を使用してください。モジュールがアダプターからのメッセージを受信しない場合は、このツールが生成したエクスポートを削除してください。モジュールがアダプターにメッセージを何も送信しない場合は、インポートを削除してください。このフィーチャーについての詳細は、インフォメーション・センターを参照してください。

SOAP エンコード配列型を持つ WSDL インターフェースのマイグレーション:

本セクションでは、SOAP エンコード配列型を持つ XML スキーマをマイグレーションまたは処理する方法について説明します。

RPC スタイルを持つ SOAP エンコード配列型は、6.0 では具象型のアンバインド済みシーケンスとして処理されます。プログラミング・モデルは、RPC スタイルの代わりに文書/リテラル・ラップ済みスタイルに移行されるため、どのような方法でも、soapenc:Array タイプを参照する XSD タイプを作成することは推奨されません。

SCA アプリケーションが、soapenc:Array タイプを使用する外部サービスを呼び出す必要があるケースも出てきます。場合によっては、これを避ける方法がないため、このような状態を処理する方法を次に示します。

サンプル WSDL コード:

```
<xsd:complexType name="Vendor">
<xsd:all>
<xsd:element name="name" type="xsd:string" />
<xsd:element name="phoneNumber" type="xsd:string" />
</xsd:all>
</xsd:complexType>
</xsd:schema>
<xsd:complexType name="Vendors">
<xsd:complexContent mixed="false">
<xsd:restriction base="soapenc:Array">
<xsd:attribute wsdl:arrayType="tns:Vendor[]" ref="soapenc:arrayType" xmlns:wsdl="http://schemas.
```

```

xmlsoap.org/wsdl/" />
</xsd:restriction>
</xsd:complexContent>
<xsd:complexType name="VendorsForProduct">
<xsd:all>
<xsd:element name="productId" type="xsd:string" />
<xsd:element name="vendorList" type="tns:Vendors" />
</xsd:all>
</xsd:complexType>
<xsd:complexType name="Product">
<xsd:all>
<xsd:element name="productId" type="xsd:string" />
<xsd:element name="productName" type="xsd:string" />
</xsd:all>
</xsd:complexType>
<message name="doFindVendorResponse">
<part name="returnVal" type="tns:VendorsForProduct" />
</message>
<operation name="doFindVendor">
<input message="tns:doFindVendor" />
<output message="tns:doFindVendorResponse" />
</operation>

```

この Web サービスのクライアント用サンプル・コード:

```

// ベンダー・サービスを探し、doFindVendor 操作を見つける
Service findVendor=(Service)ServiceManager.INSTANCE.locateService("vendorSearch");
OperationType doFindVendorOperationType=findVendor.getReference().getOperationType("doGoogleSearch");
// 入力 DataObject を作成する
DataObject doFindVendor=DataFactory.INSTANCE.create(doFindVendorOperationType.getInputType());
doFindVendor.setString("productId", "12345");
doFindVendor.setString("productName", "Refrigerator");
// FindVendor サービスを呼び出す
DataObject findVendorResult = (DataObject)findVendor.invoke(doFindVendorOperationType, doFindVendor);
// 結果を表示する
int resultProductId=findVendorResult.getString("productId");
DataObject resultElements=findVendorResult.getDataObject("vendorList");
Sequence results=resultElements.getSequence(0);
for (int i=0, n=results.size(); i
for (int i=0, n=results.size(); i

```

以下のもう 1 つの例は、データ・オブジェクトのルート・タイプが soapenc:Array である場合です。sampleElements DataObject が、上にリストされた 2 番目のスキーマを使用し、どのように作成されるかに注目してください。最初に DataObject のタイプが取得され、次に sampleStructElement のプロパティーが取得されています。これは実際はプレースホルダー・プロパティーで、DataObjects をシーケンスに追加するときに使用する有効なプロパティーを取得するためのみに使用されます。このようなパターンをシナリオで使うことができます。

サンプル WSDL コード:

```

<s:schema elementFormDefault="qualified" targetNamespace="http://soapinterop.org/xsd">
<s:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
<s:import namespace="http://schemas.xmlsoap.org/wsdl/" />
<s:complexType name="SOAPStruct">
<s:sequence>
<s:element minOccurs="1" maxOccurs="1" form="unqualified" name="varInt" type="s:int" />
<s:element minOccurs="1" maxOccurs="1" form="unqualified" name="varString" type="s:string" />
<s:element minOccurs="1" maxOccurs="1" form="unqualified" name="varFloat" type="s:float" />
</s:sequence>
</s:complexType>
<s:complexType name="ArrayOfSOAPStruct">
<s:complexContent mixed="false">
<s:restriction base="soapenc:Array">
<s:attribute wsdl:arrayType="s0:SOAPStruct[]" ref="soapenc:arrayType" />
</s:restriction>

```



```

</s:complexContent>
</s:complexType>
</s:schema>
<wsdl:message name="echoStructArraySoapIn">
<wsdl:part name="inputStructArray" type="s0:ArrayOfSOAPStruct" />
</wsdl:message>
<wsdl:message name="echoStructArraySoapOut">
<wsdl:part name="return" type="s0:ArrayOfSOAPStruct" />
</wsdl:message>
<wsdl:operation name="echoStructArray">
<wsdl:input message="tns:echoStructArraySoapIn" />
<wsdl:output message="tns:echoStructArraySoapOut" />
</wsdl:operation>
<schema targetNamespace="http://sample/elements"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://sample/elements">
<element name="sampleStringElement" type="string"/>
<element name="sampleStructElement" type="any"/>
</schema>

```

この Web サービスのクライアント用サンプル・コード:

```

// 入力 DataObject を作成し、任意のエレメントの SDO シーケンス
// を取得する
DataFactory dataFactory=DataFactory.INSTANCE;
DataObject arrayOfStruct = dataFactory.create("http://soapinterop.org/xsd","ArrayOfSOAPStruct");
Sequence sequence=arrayOfStruct.getSequence("any");
// シーケンスを移植するために、ここで使用したいサンプル・エレメントの
// SDO プロパティーを取得する
// このエレメントは XSD ファイルに定義済み。SampleElements.xsd を参照
DataObject sampleElements=dataFactory.create("http://sample/elements",
"DocumentRoot");
Property property = sampleElements.getType().getProperty("sampleStructElement");
// エレメントをシーケンスに追加する
DataObject item=dataFactory.create("http://soapinterop.org/xsd", "SOAPStruct");
item.setInt("varInt", 1);
item.setString("varString", "Hello");
item.setFloat("varFloat", 1.0f);
sequence.add(property, item);
item=dataFactory.create("http://soapinterop.org/xsd", "SOAPStruct");
item.setInt("varInt", 2);
item.setString("varString", "World");
item.setFloat("varFloat", 2.0f);
sequence.add(property, item);
// echoStructArray 操作を呼び出す
System.out.println("[client] invoking echoStructArray operation");
DataObject echoArrayOfStruct = (DataObject)interopTest.invoke("echoStructArray", arrayOfStruct);
// 結果を表示する
if (echoArrayOfStruct!=null) {
sequence=echoArrayOfStruct.getSequence("any");
for (int i=0, n=sequence.size(); i<n; i++) {
item=(DataObject)sequence.getValue(i);
System.out.println("[client] item varInt = "+
item.getInt("varInt")+
varString="+item.getString("varString")+
varFloat="+item.getFloat("varFloat"));

```

WebSphere Business Integration EJB プロジェクトのマイグレーション:

WebSphere Studio Application Developer Integration Edition では、EJB プロジェクトが拡張メッセージング (CMM) および CMP/A (Component-Managed Persistence Anywhere) などの特殊な WebSphere Business Integration フィーチャーを持つ場合があります。そのようなプロジェクトのデプロイメント記述子はマイグレーションする必要があるため、本セクションでは、そうしたマイグレーションを実行する方法を説明します。

このマイグレーションを実行するには、以下のステップを完了してください。

1. WebSphere Business Integration EJB プロジェクトを新しい 6.0 ワークスペースにコピーし、これを「ファイル」→「インポート」→「既存プロジェクトをワークスペースへ」ウィザードを使用して WebSphere Integration Developer にインポートします。オプションで、J2EE マイグレーション・ウィザードを実行することもできます。
2. 6.0 ワークスペースで実行中のすべての WebSphere Integration Developer インスタンスを閉じます。
3. 次のスクリプトを実行して、EJB プロジェクト内の WebSphere Business Integration デプロイメント記述子をマイグレーションします。

Windows の場合:

```
%WID_HOME%\wstools/eclipse/plugins/com.ibm.wbit.migration.wsadie_6.0.0/  
WSADIEEJBProjectMigration.bat
```

Linux の場合:

```
$WID_HOME/wstools/eclipse/plugins/com.ibm.wbit.migration.wsadie_6.0.0/  
WSADIEEJBProjectMigration.sh
```

以下のパラメーターがサポートされています。ワークスペースとプロジェクト名は必須です。

使用法: WSADIEEJBProjectMigration.bat
[-e eclipse-folder] -d workspace -p project
eclipse-folder: Eclipse フォルダのロケーション (通常は「eclipse」)
製品インストール・フォルダの下にあります。
workspace: マイグレーションする WSADIE EJB プロジェクトが含まれるワークスペース
project: マイグレーションするプロジェクトの名前

例:

```
WSADIEEJBProjectMigration.bat -e "C:\IBM\WID6\ eclipse" -d "d:\my60workspace" -p "MyWBIEJBProject"
```

4. WebSphere Integration Developer を開くときに、更新されたファイルを取得するために EJB プロジェクトを更新する必要があります。
5. EJB プロジェクト内の **ibm-web-ext.xmi** ファイルを検索します。ファイルが検出されたら、ファイル内のエレメントの下に、以下の行があることを確認します。

```
<webappext:WebAppExtension> element:  
<webApp href="WEB-INF/web.xml#WebApp"/>
```

6. 5.1 で生成された古いデプロイメント・コードを除去します。 WebSphere Application Server ガイドラインの該当する手順に従って、デプロイメント・コードを再生成します。

名前空間が衝突した場合の手動修正の実行:

WebSphere Studio Application Developer Integration Edition 5.1 では、同一の名前とターゲット名前空間を持つ 2 つの異なる XSD または WSDL タイプを定義することがサポートされていました。 WebSphere Integration Developer 6.0 では、これはサポートされていません。マイグレーションされたプロジェクトをビルドした後、重複定義エラーが発生する場合は、手動マイグレーションが必要になります。

この問題を修正するには、以下のステップを実行します。

1. 定義が同じである場合は、定義のいずれか 1 つを削除してから、プロジェクトをクリーンにして再ビルドします。既存の WSDL/XSD ファイルが、削除しなかった 定義を含んでいるファイルをポイントすることによって発生したエラーがある場合は、そのエラーを訂正します。
2. 定義が同じでなく、マイグレーションされたサービス内でその定義の両方を使用する必要がある場合は、定義名またはターゲット名前空間を名前変更します。重複ファイル全体の中に定義がほんのわずかなしかない場合には、それらの定義名を変更することが推奨されます。ファイル内のすべての定義が重複

している場合には、すべての定義のターゲット名前空間を変更することが推奨されます。プロジェクトをクリーンにして再ビルドし、変更した定義内で使用したい成果物が、新しい定義名または名前空間を参照することを確認してください。

3. WSDL ファイル内の同じ名前空間に 2 つの `import` 文がある場合には、これらの WSDL の 1 つが他方をインポートし、それが次のものをインポートするようようにチェーニングして、その結果、WSDL ファイルごとに、この名前空間に対して、インポートが 1 つだけ存在するようにすることによってこの問題を修正することができます。その後で、プロジェクトをクリーンにし、再ビルドしてください。

5.1 Web Services Invocation Framework (WSIF) 定義の手動削除:

ソース成果物のマイグレーションを完了した後、今後は使用されないすべての 5.1 WSIF バインディングおよびサービス WSDL 定義を、6.0 プロジェクトから削除する必要があります。サービス・マイグレーションの使用量シナリオは、WSIF バインディングまたはサービスがまだ使用される唯一のケースです。

以下の WSDL 名前空間は、バインディングまたはサービス定義が 5.1 WSIF サービスであり、今後使用されることがない場合、廃棄してもかまわないということを示します。

EJB WSIF 名前空間:

<http://schemas.xmlsoap.org/wsdl/ejb/>

Java WSIF 名前空間:

<http://schemas.xmlsoap.org/wsdl/java/>

JMS WSIF 名前空間:

<http://schemas.xmlsoap.org/soap/jms/>

ビジネス・プロセス WSIF 名前空間:

<http://schemas.xmlsoap.org/wsdl/process/>

変換プログラム WSIF 名前空間:

<http://schemas.xmlsoap.org/wsdl/transformer/>

IMS WSIF 名前空間:

<http://schemas.xmlsoap.org/wsdl/ims/>

CICS-ECI WSIF 名前空間:

<http://schemas.xmlsoap.org/wsdl/cicseci/>

CICS-EPI WSIF 名前空間:

<http://schemas.xmlsoap.org/wsdl/cicsepi/>

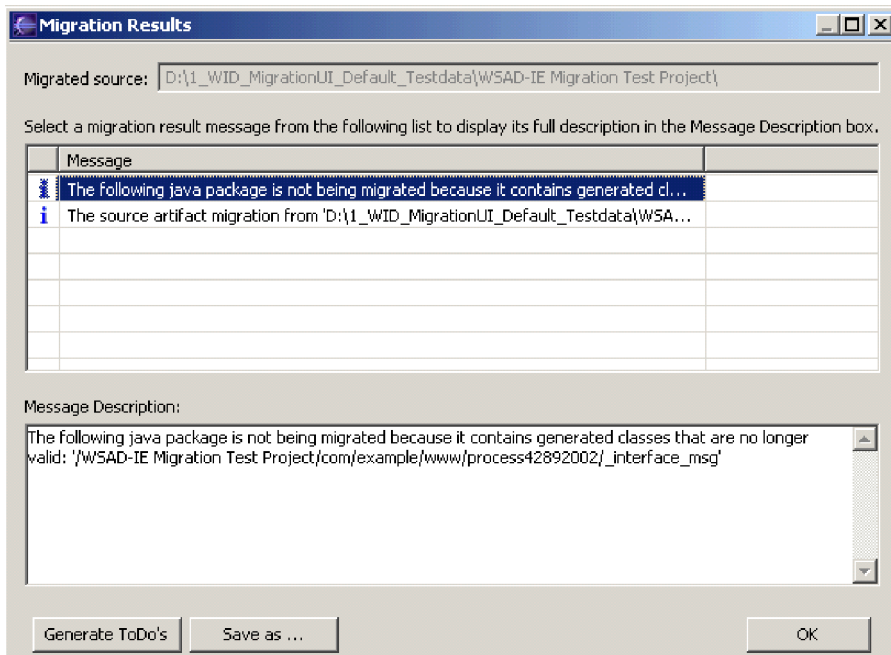
HOD WSIF 名前空間:

<http://schemas.xmlsoap.org/wsdl/hod3270/>

ソース成果物マイグレーションの検査

マイグレーション・ウィザードが正常に完了すると、エラー・メッセージ、警告メッセージ、情報メッセージのリストが表示されます。これらのメッセージを使用して、ソース成果物のマイグレーションを検査することができます。

マイグレーション・ウィザードが完了すると、次のページが表示されます。



それぞれのメッセージを調べて、完全にマイグレーションできなかった成果物を即時に修正するためにアクションを実行する必要があるかどうかを確認します。

マイグレーションの一部が完了していることを検査するために、ビジネス・インテグレーション・パースペクティブに切り替えて、古いサービス・プロジェクトからのすべてのプロセスおよび WSDL インターフェースが新規モジュール内に現れることを確認します。プロジェクトをビルドし、プロジェクトのビルドを妨げるエラーがあればそのエラーを修正します。

ビジネス・インテグレーション・アプリケーションのマイグレーションを完了するために必要な手動マイグレーション・ステップを実行した後で、そのアプリケーションを EAR ファイルとしてエクスポートし、それを WebSphere Process Server にインストールして、該当するリソースを構成します。

任意のクライアント・コードをマイグレーションするために必要な手動マイグレーション・ステップを実行するか、または WebSphere Integration Developer を使用して新規のクライアント・コードを生成します。クライアントがアプリケーションにアクセスできること、およびアプリケーションが以前のランタイム環境と同じように動作することを確認してください。

ソース成果物マイグレーション失敗時の作業

WebSphere Studio Application Developer Integration Edition からのソース成果物マイグレーションが失敗した場合、これに対処するさまざまな方法があります。

可能性のあるソース成果物マイグレーションの失敗を以下にいくつか示します。

- 以下のメッセージを受け取った場合:

「マイグレーション・エラー・メッセージ」
理由: 重大なマイグレーション障害
メッセージ: IBM 担当員に連絡してください

新しいワークスペースの .metadata フォルダにある WebSphere Integration Developer のログ・ファイルを調べて、エラーの詳細を確認してください。可能な場合は、エラーの原因を解決して、新しいワークスペースに作成されたモジュールを削除し、マイグレーションを再試行してください。

このメッセージが表示されずにマイグレーション・ウィザードが完了した場合でも、情報メッセージ、警告メッセージ、およびエラー・メッセージのリストが表示されます。これらのメッセージは、サービス・プロジェクトの一部を自動的にマイグレーションできなかったため、マイグレーションを完了するには手動で変更を行う必要があることを示します。

ソース成果物マイグレーション・プロセスのベスト・プラクティス

WebSphere Studio Application Developer Integration Edition ソース成果物マイグレーション・プロセスには多数のベスト・プラクティスがあります。

次の事例は、WebSphere Studio Application Developer Integration Edition サービスが、新しいプログラミング・モデルに確実に正しくマイグレーションされるように設計する方法を示しています。

- 可能であれば、「割り当て」アクティビティを使用してみてください (これは、拡張変換が必要な場合にのみ必要とされる変換プログラム・サービスとは対照的です)。この事例を使用しなければならない理由は、SCA モジュールが変換プログラム・サービスを呼び出すには、中間コンポーネントを構成する必要があるためです。さらに、WebSphere Integration Developer には、バージョン 5.1 で作成された変換プログラム・サービス用の特殊ツールのサポートはありません (変換プログラム・サービスの動作を変更する必要がある場合は、WSDL または XML エディターを使用して、WSDL ファイルに組み込まれた XSLT を変更する必要があります)。
- Web サービス・インターオペラビリティ (WS-I) 仕様と 6.0 優先スタイルのとおり、WSDL メッセージあたりに 1 つのパーツを指定します。
- WSDL doc-literal スタイルは 6.0 の優先スタイルであるため、これを使用してください。
- すべての複合タイプに名前が付いていること、およびそれぞれの複合タイプがそのターゲット名前空間と名前によって一意的に識別可能であることを確認してください。以下に、複合タイプおよびそのタイプのエレメントを定義するのに推奨される方法を示します (複合タイプ定義の後に、その複合タイプ定義で使用するエレメント定義が続きます)。

```
<schema attributeFormDefault="qualified"
elementFormDefault="unqualified"
targetNamespace="http://util.claimshandling.bpe.samples.websphere.ibm.com"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://util.claimshandling.bpe.samples.websphere.ibm.com">
  <complexType name="Duration">
    <all>
      <element name="hours" type="int"/>
      <element name="minutes" type="int"/>
      <element name="days" type="int"/>
    </all>
  </complexType>
  <element name="DurationElement" type="tns:Duration"/>
</schema>
```

以下の例は、SDO が XML (匿名複合タイプ定義が含まれているエレメント) にシリアルライズされるときに問題の原因となることがあるため、避けなければならない匿名複合タイプです。

```
<schema attributeFormDefault="qualified"
elementFormDefault="unqualified"
targetNamespace="http://util.claimshandling.bpe.samples.websphere.ibm.com"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://util.claimshandling.bpe.samples.websphere.ibm.com">
  <element name="DurationElement">
    <complexType>
      <all>
        <element name="hours" type="int"/>
        <element name="minutes" type="int"/>
        <element name="days" type="int"/>
      </all>
    </complexType>
  </element>
</schema>
```



```

</all>
</complexType>
</element>
</schema>

```

- 外部コンシューマー向けサービスを公開する場合は (Apache SOAP/HTTP ではなく)、IBM Web Services を使用してサービス・デプロイ・コードを生成します。これは、IBM Web Services は 6.0 で直接サポートされているが、Apache Web サービスはサポートされていないためです。
- WSDL および XSD ファイルを 5.1 で編成して、マイグレーション中に実行しなければならない再編成の量を最小限にするには、2 つの方法があります。6.0 では、WSDL ファイルや XSD ファイルなどの共用成果物を BI サービスによって参照させるために、これらを BI プロジェクト (Business Integration modules および libraries) に配置する必要があります。
 - サービス・プロジェクトが参照できる Java プロジェクト内の複数のサービス・プロジェクトによって共有されるすべての WSDL ファイルを保持します。6.0 へのマイグレーションで、5.1 の共用 Java プロジェクトと同じ名前を持つ新しいビジネス・インテグレーション・ライブラリーを作成します。5.1 の共用 Java プロジェクトから、すべての成果物をこのライブラリーにコピーして、これらの成果物を使用するサービス・プロジェクトをマイグレーションするときに、マイグレーション・ウィザードが成果物を解決できるようにしてください。
 - サービス・プロジェクトが、そのサービス・プロジェクト自体の中で参照するすべての WSDL/XSD ファイルのローカル・コピーを保持します。WebSphere Studio Application Developer Integration Edition サービス・プロジェクトは、WebSphere Integration Developer の Business Integration Module にマイグレーションされ、モジュールは他のモジュールとの依存関係を持つことができません (WSDL ファイルまたは XSD ファイルを共有するために別のサービス・プロジェクトと依存関係を持つサービス・プロジェクトは、クリーンにマイグレーションされません)。
- Business Process Choreographer Generic Messaging API (Generic MDB) は 6.0 では提供されないため、使用しないでください。MDB インターフェース・オファリング遅延バインディングは 6.0 では使用できません。
- 特定バージョンのプロセスに特有の生成済みセッション Bean を呼び出す代わりに、Business Process Choreographer Generic EJB API を使用してください。これらのセッション Bean は、V6.0.0.0 では生成されません。
- 同じ操作に対して複数の応答を持つビジネス・プロセスがある場合、いずれかにクライアント設定があれば、その操作に対するすべての応答が同じクライアント設定を持っていることを確認してください。6.0 では、操作応答ごとに 1 セットのクライアント設定のみがサポートされるためです。
- 次のガイドラインに準拠して、BPEL Java snippet を設計してください。
 - WSIFMessage パラメーターを任意のカスタム Java クラスに送らないでください。可能であれば、WSIFMessage データ・フォーマットに依存しないようにしてください。
 - 可能であれば、WSIF メタデータ API を使用しないでください。
- BPEL Java Snippet で BPEL 変数と同じ名前のローカル変数を宣言しないでください。6.0 では Snippet 内で BPEL 変数に直接アクセスできるため、同じ名前のローカル変数があると、競合が起こることがあります。
- WSDL ポート・タイプ/メッセージから生成された Java/EJB スケルトンは WSIF クラス (WSIFFormatPartImpl など) に依存するため、トップダウンの EJB または Java サービスを作成することはできるだけ避けてください。その代わりに、Java/EJB インターフェースを作成してから、Java クラス/EJB にサービスを生成してください (ボトムアップのアプローチ)。
- soapenc:Array タイプを参照する WSDL インターフェースを作成したり、使用しないようにしてください。このタイプのインターフェースは、SCA プログラミング・モデルではネイティブにサポートされないためです。

- ハイレベル・エレメント (maxOccurs 属性が 1 より大きい) が配列型のメッセージ・タイプは作成しないようにしてください。このタイプのインターフェースは、SCA プログラミング・モデルではネイティブにサポートされていないためです。
- WSDL インターフェースは正確に定義し、xsd:anyType タイプへの参照を持つ XSD complexTypes の使用はできるだけ避けてください。
- WebSphere Process Server V6 にマイグレーションするときに名前の衝突を避けるために、EJB または Java Bean から生成するすべての WSDL および XSD について、ターゲット名前空間が固有である (Java クラス名およびパッケージ名が、ターゲット名前空間によって表されている) ことを確認してください。WebSphere Process Server V6 では、同一の名前とターゲット名前空間を持つ、2 つの異なる WSDL/XSD 定義は許可されません。この状況は、ターゲット名前空間を明示して指定しないで Web サービス・ウィザードまたは Java2WSDL コマンドが使用される場合によく起こります (ターゲット名前空間が EJB または Java Bean のパッケージ名に対しては固有であるが、クラス自体に対して固有でないため、Web サービスが同一パッケージ内で 2 つ以上の EJB または Java Bean に対して生成される場合に問題が発生します)。この解決方法は、Web サービス・ウィザードで名前空間マッピングにカスタム・パッケージを指定するか、**-namespace** Java2WSDL コマンド行オプションを使用して、生成されるファイルの名前空間が所定のクラスに対して固有となるようにすることです。
- すべての WSDL ファイルに、できるだけ固有の名前空間を使用してください。WSDL 1.1 仕様に準拠した同一の名前空間を持つ 2 つの異なる WSDL ファイルのインポートについては制限があり、WebSphere Integration Developer 6.0 では、これらの制限が厳密に強制されます。

マイグレーション・プロセスの制限 (ソース成果物マイグレーション)

WebSphere Studio Application Developer Integration Edition ソース成果物マイグレーション・プロセスには、特定の制限があります。

以下のリストでは、ソース成果物マイグレーションのマイグレーション・プロセスの一部の制限について詳しく説明します。

一般的制限

- このマイグレーション・ウィザードでは、WebSphere Studio Application Developer Integration Edition のワークスペース全体を処理することはできません。これは、一度に 1 つの WebSphere Studio Application Developer Integration Edition サービス・プロジェクトをマイグレーションすることを意味します。
- マイグレーション・ウィザードでは、アプリケーション・バイナリーはマイグレーションされません。WebSphere Studio Application Developer Integration Edition サービス・プロジェクトにあるソース成果物のみがマイグレーションされます。
- ビジネス・ルール Bean は、WebSphere Process Server 6.0 で使用すべきではありませんが、WebSphere Process Server のインストール時に、使用すべきではないビジネス・ルール Bean のサポートをインストールするためのオプションがあります。これによって、ビジネス・ルール Bean が WebSphere Process Server 6.0 サーバー上で「現状どおり」実行されます。古いビジネス・ルール Bean 用のツール・サポートはありませんが、古いビジネス・ルール Bean の成果物をツールでコンパイルしたい場合は、WebSphere Integration Developer のドキュメンテーションに従って、これらの使用すべきではないフィーチャーを、組み込みの WebSphere Process Server 6.0 テスト・サーバー上にインストールしてから、推奨されない JAR ファイルを外部 JAR としてプロジェクト・クラスパスに手動で追加する必要があります。WebSphere Integration Developer で使用可能な新しいビジネス・ルールのツールを使用して、6.0 仕様に従ったビジネス・ルールを再作成する必要があります。
- 拡張メッセージング・サポートは WebSphere Process Server 6.0 で使用すべきではありませんが、WebSphere Process Server のインストール時に、使用すべきではない拡張メッセージング・フィーチャー

のサポートをインストールするためのオプションがあります。これによって、既存のアプリケーションが WebSphere Process Server 6.0 サーバー上で「現状どおり」実行できます。古い拡張メッセージング・フィーチャー用のツール・サポートはありませんが、古い拡張メッセージングの成果物をツールでコンパイルしたい場合は、WebSphere Integration Developer のドキュメンテーションに従って、これらの使用すべきではないフィーチャーを、組み込みの WebSphere Process Server 6.0 テスト・サーバー上にインストールしてから、使用すべきではない JAR ファイルを「外部 JAR」としてプロジェクト・クラスパスに手動で追加する必要があります。

- 標準で提供されている JMS データ・バインディングは、カスタム JMS ヘッダー・プロパティーへのアクセスを提供しません。カスタム JMS ヘッダー・プロパティーにアクセスするには、SCA サービス用にカスタム・データ・バインディングを作成する必要があります。

SCA プログラミング・モデルの制限

- SDO バージョン 1 仕様は、COBOL または C バイト配列へのアクセスを提供しません。これは、IMS 複数セグメントを操作するユーザーに影響することになります。
- シリアルライゼーション用 SDO バージョン 1 仕様は、COBOL 再定義または C 共用体をサポートしません。
- SCA プログラミング・モデルに従ってソース成果物を再設計するときは、文書/リテラル・ラップ済み WSDL スタイル (WebSphere Integration Developer ツールを使用して作成される新規成果物のデフォルト・スタイル) は、メソッドの多重定義をサポートしないことに注意してください。その他の WSDL スタイルは引き続きサポートされているため、このような場合は、文書/リテラル・ラップ済み以外の WSDL スタイル/エンコードを使用することをお勧めします。
- 配列のネイティブ・サポートは限定されています。soapenc:Array タイプの WSDL インターフェースを公開する外部サービス呼び出すには、「maxOccurs」属性が 1 より大きいエレメントを定義する WSDL インターフェースを作成する必要があります (これは、配列タイプの設計で推奨される方法です)。

BPEL マイグレーション・プロセスの技術的な制限

- 1 つの BPEL 操作での複数の応答 - WebSphere Business Integration Server Foundation では、ビジネス・プロセスは同じ操作に対して 1 つの受信アクティビティーと複数の応答アクティビティーを持つことができず、1 つの BPEL 操作で複数の応答を持つことができませんでした。
- BPEL Java Snippet マイグレーションの制限** - プログラミング・モデルは WebSphere Studio Application Developer Integration Edition から WebSphere Integration Developer に著しく変更されていて、サポートされている WebSphere Studio Application Developer Integration Edition API の一部は、対応する WebSphere Integration Developer API に直接マイグレーションできません。自動マイグレーション・ツールが各 Java Snippet を新しいプログラミング・モデルに変換できないように、Java ロジックは BPEL Java Snippet にあります。標準 Snippet API 呼び出しの多くは、5.1 Java Snippet プログラミング・モデルから 6.0 Java Snippet プログラミング・モデルに自動的にマイグレーションされます。可能な場合、WSIF API 呼び出しは DataObject API 呼び出しにマイグレーションされます。WSIFMessage オブジェクトを受け入れるカスタム Java クラスは、commonj.sdo.DataObject オブジェクトを代わりに受け入れて戻すように手動のマイグレーションを必要とします。
 - WSIFMessage メタデータ API** - 可能な場合、すべての WSIF API の使用を避けてください。マイグレーション・ウィザードでは、WSIFMessage メタデータおよびその他の WSIF API を SDO 同等機能に自動的にマイグレーションできないため、手動のマイグレーションが必要となります。
 - EndpointReference/EndpointReferenceType API** - これらのクラスは自動的にマイグレーションされません。パートナー・リンク getter/setter メソッドは 5.1 の `com.ibm.websphere.srm.bpel.wsaddressing.EndpointReferenceType` オブジェクトの代わりに `commonj.sdo.DataObject` オブジェクトを処理するため、手動のマイグレーションが必要になります。

- **重複名を持つ複合タイプ** - 同一の名前空間とローカル名を持つか、または名前空間は異なるが同一のローカル名を持つ複合タイプを (WSDL または XSD で) アプリケーションが宣言する場合、このタイプを使用する Java Snippet が正しくマイグレーションされない場合があります。マイグレーション・ウィザードが完了した後で Snippet が正しいかどうか検査してください。
- **java.lang パッケージ内の Java クラスと同一のローカル名を持つ複合タイプ** - J2SE 1.4.2 の java.lang パッケージ内のクラスと同一のローカル名を持つ複合タイプを (WSDL または XSD で) アプリケーションが宣言する場合、対応する java.lang クラスを使用する Java Snippet が正しくマイグレーションされない場合があります。マイグレーション・ウィザードが完了した後で Snippet が正しいかどうか検査してください。
- **BPEL Java Snippet のローカル変数と同じ名前を持つ BPEL 変数** - ローカル変数を BPEL Java Snippet の BPEL 変数と同じ名前で定義した場合は、これを手動で修正する必要があります。BPEL 変数には Java Snippet 内で名前アクセスできるようになっていて、競合が起こる可能性があるためです。
- **読み取り専用および読み取り/書き込みの BPEL 変数** - どの 5.1 Java Snippet コードでも、BPEL 変数を「読み取り専用」に設定することが可能でした。つまり、このオブジェクトに行われた変更は、BPEL 変数の値にまったく影響を与えません。また BPEL 変数を「読み取り/書き込み」に設定することも可能でした。つまり、オブジェクトに行われた変更はすべて、BPEL 変数自体に反映されます。以下に、任意の 5.1 BPEL Java Snippet で、Java Snippet に「読み取り専用」としてアクセスできる 4 つの方法を示します。

```
getMyInputVariable()
getMyInputVariable(false)
getVariableAsWSIFMessage("MyInputVariable")
getVariableAsWSIFMessage("MyInputVariable", false)
```

以下に、任意の 5.1 BPEL Java Snippet で、BPEL 変数に「読み取り/書き込み」としてアクセスできる 2 つの方法を示します。

```
getMyInputVariable(true)
getVariableAsWSIFMessage("MyInputVariable", true)
```

6.0 では、BPEL 変数に対する読み取り専用および読み取り/書き込みのアクセスは、「Snippet ごとに」に処理されます。つまり、特別なコメントを BPEL Java Snippet に追加して、Snippet が実行を終了した後、BPEL 変数に対する更新を、廃棄するかまたは保持するかを指定できます。以下に、6.0 BPEL Java Snippet タイプに対するデフォルトのアクセス設定を示します。

```
BPEL Java Snippet Activity
Default Access: read-write
Override Default Access with comment containing:
@bpe.readOnlyVariables names="variableA,variableB"
BPEL Java Snippet Expression (Used in a Timeout, Condition, etc)
Default Access: read-only
Override Default Access with comment containing:
@bpe.readWriteVariables names="variableA,variableB"
```

マイグレーション時に、変数が 6.0 のデフォルトではない方法でアクセスされた場合、これらのコメントが自動的に作成されます。競合が存在する場合 (つまり、同一 Snippet 内で BPEL 変数が「読み取り専用」および「読み取り/書き込み」としてアクセスされた場合)、警告が出され、アクセスは「読み取り/書き込み」に設定されます。そのような警告を受け取った場合は、BPEL 変数アクセスを「読み取り/書き込み」に設定することが、ご使用の状態にとって正しいことであるか確認してください。これが正しくない場合は、WebSphere Integration Developer BPEL エディターを使用して、手動で訂正する必要があります。

- **複合タイプの多値プリミティブ・プロパティー** - 5.1 では、多値プロパティーはプロパティー・タイプの配列によって表されます。そういうものだから、プロパティーを取得および設定する呼び出し

は、配列を使用します。 6.0 では、これを表すために `java.util.List` が使用されます。自動マイグレーションは、多値プロパティが、あるタイプの Java オブジェクトであるケースをすべて処理しますが、プロパティ・タイプが Java プリミティブ (整数、長形式、短形式、バイト、文字、浮動小数点、倍精度、およびブール) である場合には、配列全体を取得および設定する呼び出しは変換されません。このような場合の手動マイグレーションでは、残りの Snippet で使用するために、対応する Java ラッパー・クラスで、またはそこからプリミティブ (整数、長形式、短形式、バイト、文字、浮動小数点、倍精度、およびブール) をラップ/アンラップするためのループを追加することが必要になる場合があります。

- **複合タイプを表す生成済みクラスのインスタンス化** - 5.1 では、アプリケーションに定義されている複合タイプの生成済みクラスは、引数なしのデフォルト・コンストラクターを使用して Java Snippet で容易にインスタンス化できました。例えば、次のようになります。

```
MyProperty myProp = new MyProperty();
InputMessage myMsg = new InputMessage();
myMsg.setMyProperty(myProp);
```

6.0 では、特殊なファクトリー・クラスを使用してこれらのタイプをインスタンス化する必要があります。または、収容型のインスタンスを使用してサブタイプを作成できます。 BPEL プロセス変数 `InputVariable` がタイプ `InputMessage` を持つと定義された場合、上記 Snippet の 6.0 バージョンは以下ようになります。

```
com.ibm.websphere.bo.BOFactory boFactory=
(com.ibm.websphere.bo.BOFactory)
com.ibm.websphere.sca.ServiceManager.INSTANCE.locateService(
"com/ibm/websphere/bo/BOFactory");
commonj.sdo.DataObject myMsg =
boFactory.createByType(getVariableType("InputVariable"));
commonj.sdo.DataObject myProp =
myMsg.createDataObject("MyProperty");
```

Snippet コンバーターはこの変更を試行しますが、オリジナルのインスタンス化が行われる順序が親から子へのパターンに従わない場合には、手動のマイグレーションが必要になります (すなわち、コンバーターは、Snippet 内のインスタンス化ステートメントを自ら再配列しようとはしません)。

- **WebSphere Business Integration Server Foundation 5.1** では、動的参照は、名前空間のタイプ `EndpointReferenceType` またはエレメント `EndpointReference` の WSDL メッセージ・パーツとして表されました。

```
http://wsaddressing.bpel.srm.websphere.ibm.com
```

このような参照は、標準ビジネス・プロセス名前空間から標準 `service-ref` エレメント・タイプにマイグレーションされます。

```
http://schemas.xmlsoap.org/ws/2004/03/business-process/
```

```
http://schemas.xmlsoap.org/ws/2004/08/addressing
```

すべての参照が正しく解決されるようにこれらのスキーマ定義をプロジェクトに手動でインポートする手順については、 BPEL エディターのドキュメンテーションを参照してください。

- **BPEL 変数メッセージ・タイプ** - WSDL メッセージ・タイプは、Java Snippet で使用されるすべての BPEL 変数に対して指定する必要があります。「`messageType`」属性を指定せずに BPEL 変数にアクセスする Java Snippet は、マイグレーションすることができません。

特記事項

The XDoclet Documentation included in this IBM product is used with permission and is covered under the following copyright attribution statement: Copyright (c) 2000-2004, XDoclet Team. All rights reserved.

Portions based on *Design Patterns: Elements of Reusable Object-Oriented Software*, by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Copyright (c) 1995 by Addison-Wesley Publishing Company, Inc. All rights reserved.

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あり、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。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

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または保証条件は適用されないものとします。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。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

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。

Intellectual Property Dept. for Rational Software
IBM Corporation
20 Maguire Road
Lexington, Massachusetts 02421-3112
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 사용할 수 있습니다, 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。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性があります, その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があります、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。お客様は、IBM のアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

(C) (お客様の会社名) (年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 (C) Copyright IBM Corp. 2000, 2005. All rights reserved.

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。

プログラミング・インターフェース情報

プログラミング・インターフェース情報は、プログラムを使用してアプリケーション・ソフトウェアを作成する際に役立ちます。

一般使用プログラミング・インターフェースにより、お客様はこのプログラム・ツール・サービスを含むアプリケーション・ソフトウェアを書くことができます。

ただし、この情報には、診断、修正、および調整情報が含まれている場合があります。診断、修正、調整情報は、お客様のアプリケーション・ソフトウェアのデバッグ支援のために提供されています。

警告: 診断、修正、調整情報は、変更される場合がありますので、プログラミング・インターフェースとしては使用しないでください。

商標

<http://www.ibm.com/legal/copytrade.shtml> を参照してください。



Printed in Japan

SD88-6689-01



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