



Migration



Migration

Hinweis

Vor der Verwendung dieser Informationen und des darin beschriebenen Produkts sollten die allgemeinen Informationen unter "Bemerkungen" gelesen werden.

Zweite Ausgabe (März 2005)

Diese Veröffentlichung ist eine Übersetzung des Handbuchs

IBM® WebSphere® Application Developer Version 6.0.1 Migration Guide

herausgegeben von International Business Machines Corporation, USA

© Copyright International Business Machines Corporation 2006

© Copyright IBM Deutschland GmbH 2006

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

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

Änderung des Textes bleibt vorbehalten.

Herausgegeben von:
SW NLS Center
Kst. 2877
März 2006

© Copyright International Business Machines Corporation 2005. Alle Rechte vorbehalten.

Inhaltsverzeichnis

Migration von Anwendungen mithilfe von WebSphere Integration Developer . . . 1

PDF Version 1

Lernprogramm: Migration zu WebSphere Integration
Developer 1

Migration zu WebSphere Integration Developer. . . 1

 Migration zu WebSphere Process Server von

 WebSphere InterChange Server 1

Migration zu WebSphere Integration Developer
von WebSphere MQ Workflow 30

Migration von Quellenartefakten zu WebSphere
Integration Developer von WebSphere Studio

Application Developer Integration Edition . . . 36

Bemerkungen 101

Migration von Anwendungen mithilfe von WebSphere Integration Developer

WebSphere Integration Developer Version 6.0 bietet die für die Migration Ihrer vorhandenen Umgebung notwendigen Tools.

Die folgenden Themen beschreiben das Konzept, Verweise und schrittweise Anleitungen für die Migration zum WebSphere Integration Developer:

PDF Version

Diese Migrationsinformation ist auch als PDF erhältlich.

Dieses Dokument ist auch als PDF Datei erhältlich

Sie benötigen Adobe Acrobat, um sie anzuzeigen. Eine kostenlose Version dieser Software ist bei www.adobe.com erhältlich.

Lernprogramm: Migration zu WebSphere Integration Developer

Dieses Lernprogramm erläutert, wie Sie ein vorhandenes Serviceprojekt mit WebSphere Integration Developer in ein Modulprojekt migrieren.

Das Lernprogramm besteht aus einer Lerneinheit im Filmformat.

- Lerneinheit 1: Beim Migrieren eines Serviceprojekts lernen Sie, wie Sie ein vorhandenes Serviceprojekt in ein Modulprojekt migrieren.

Klicken Sie auf den folgenden Link, um das Lernprogramm zu starten:

Anmerkung: Der folgende Link kann auf dieser Website nicht verwendet werden. WebSphere Integration Developer muss installiert sein, damit der Link ordnungsgemäß funktioniert.

Lernprogramm: Migration auf WebSphere Integration Developer

Migration zu WebSphere Integration Developer

WebSphere Integration Developer Version 6.0 bietet die für die Migration Ihrer vorhandenen Umgebung notwendigen Tools.

Die folgenden Themen beschreiben das Konzept, Verweise und schrittweise Anleitungen für die Migration zum WebSphere Integration Developer:

Migration zu WebSphere Process Server von WebSphere InterChange Server

Die Migration von WebSphere InterChange Server zu WebSphere Process Server wird durch die folgenden Funktionen unterstützt:

Anmerkung: Angaben zu Einschränkungen, die in diesem Release von WebSphere Process Server mit der Migration verbunden sind, finden Sie in den Release-Informationen.

- Automatische Migration von Quellenartefakten durch Migrationstools, die folgendermaßen aufgerufen werden können:

- Über das Menü **'Datei' → 'Importieren'** in WebSphere Integration Developer
- Über die Eingangsanzeige von WebSphere Integration Developer
- Über den Migrationsassistenten von WebSphere Process Server - Erste Schritte
- Über das Befehlszeilendienstprogramm **reposMigrate**
- Native Unterstützung in der Laufzeit vieler APIs von WebSphere InterChange Server
- Unterstützung für die aktuelle Adaptertechnologie von WebSphere Business Integration, damit vorhandene Adapter mit WebSphere Process Server kompatibel sind

Die Migration von Quellenartefakten wird zwar unterstützt, aber es empfiehlt sich, zunächst durch intensive Analysen und Tests zu ermitteln, ob die resultierenden Anwendungen in WebSphere Process Server erwartungsgemäß funktionieren oder aber nach der Migration überarbeitet werden müssen. Diese Empfehlung hängt mit den folgenden Einschränkungen hinsichtlich der funktionalen Parität zwischen WebSphere InterChange Server und der vorliegenden Version von WebSphere Process zusammen. In dieser Version von WebSphere Process Server gibt es für die folgenden Funktionen von WebSphere InterChange Server keine äquivalente Unterstützung:

- Zugriffsschnittstelle
- Vom Adapter eingeleitete Antwort auf synchrone Anforderung
- Serviceaufruf für synchrones Senden mit Zeitlimit
- Serviceaufruf **'Async_in'**
- Isolation
- Ereignissequenzierung
- Sofortige Implementierung/Dynamische Aktualisierung
- Sicherheit - Prüfung
- Sicherheit - Differenziertes RBAC
- Gruppenunterstützung
- Scheduleroperation anhalten
- Sicherheitsdeskriptoren werden nicht migriert
- Umsetzungen von angepassten XML-Snippets werden während der Migration von Zuordnungs- und Collaboration-Schablonen nicht unterstützt

Unterstützte Migrationspfade für WebSphere InterChange Server

WebSphere Process Server-Migrationstools unterstützen die Migration von WebSphere InterChange Server Versionen 4.2.2, 4.2.3 und 4.3.

Jedes WebSphere InterChange Server Release vor Version 4.2.2 muss zunächst zu Version 4.2.2, 4.2.3 oder 4.3 vor der Migration zu WebSphere Process Server migriert werden.

Vorbereitung für die Migration von WebSphere InterChange Server

Stellen Sie vor der Migration zu WebSphere Process Server von WebSphere InterChange Server zunächst sicher, dass Sie Ihre Umgebung ordnungsgemäß vorbereitet haben. WebSphere Process Server bietet die für die Migration von WebSphere InterChange Server erforderlichen Tools.

Diese Migrationstools können folgendermaßen aufgerufen werden:

- Über das Menü **'Datei' → 'Importieren'** in WebSphere Integration Developer
- Über die Eingangsanzeige von WebSphere Integration Developer
- Über den Migrationsassistenten von WebSphere Process Server - Erste Schritte

Die Eingabe für die Migrationstools besteht aus einer Repository-JAR-Datei, die aus WebSphere InterChange Server exportiert wurde. Daher müssen Sie vor dem Zugriff auf die Migrationstools mit einer dieser Optionen zunächst Folgendes ausführen:

1. Stellen Sie sicher, dass Sie eine Version von WebSphere InterChange Server ausführen, die zum WebSphere Process Server migriert werden kann. Siehe das Thema "Unterstützte Migrationspfade für WebSphere InterChange Server".
2. Exportieren Sie Ihre Quellenartefakte aus WebSphere InterChange Server in eine Repository-Jar-Datei mit dem WebSphere InterChange Server **repos_copy**-Befehl, wie in der Dokumentation für InterChange Server wie in der Dokumentation für WebSphere InterChange Server beschrieben. Diese Jar-Datei wird den Migrationstools als Eingabe hinzugefügt.

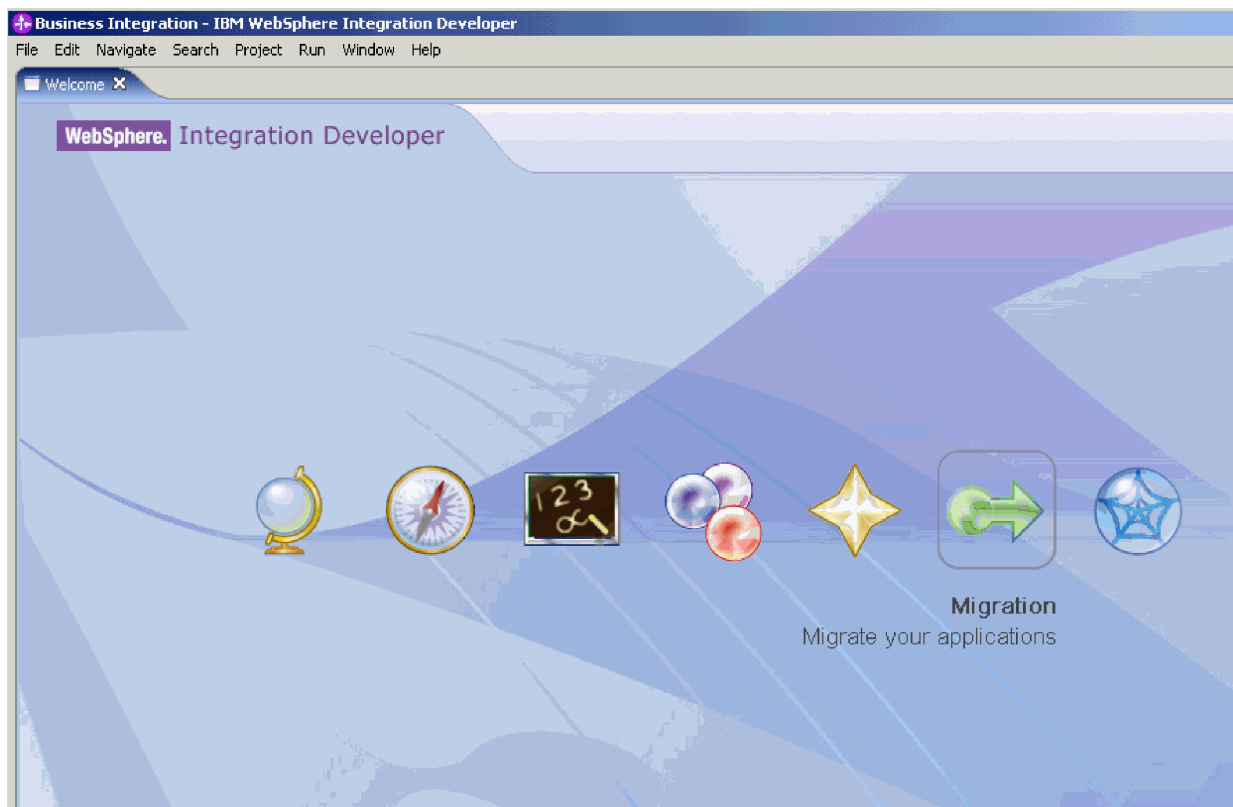
Migration von WebSphere InterChange Server mithilfe des Migrationsassistenten

Verwenden Sie den WebSphere Integration Developer-Migrationsassistenten zur Migration Ihrer vorhandenen WebSphere InterChange Server-Artefakte.

Befolgen Sie diese Schritte zur Verwendung des Migrationsassistenten zur Migration Ihrer WebSphere InterChange Server-Artefakte:



1. Klicken Sie auf der Willkommensseite auf das Symbol , um die Seite 'Migration' zu öffnen:

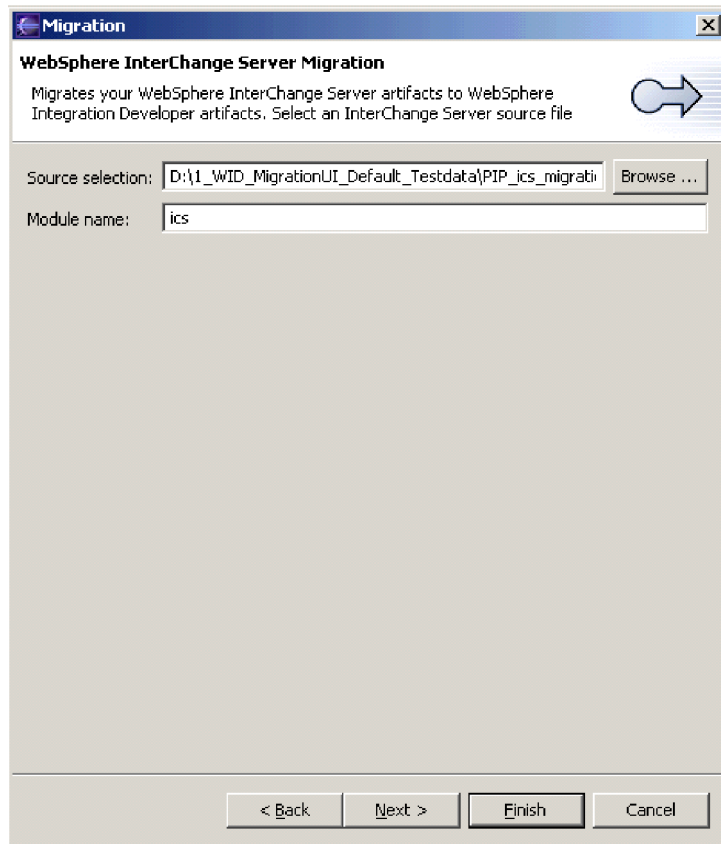


2. Wählen Sie auf der Seite 'Migration' die Option für die Migration eines WebSphere ICS-Repositorys:

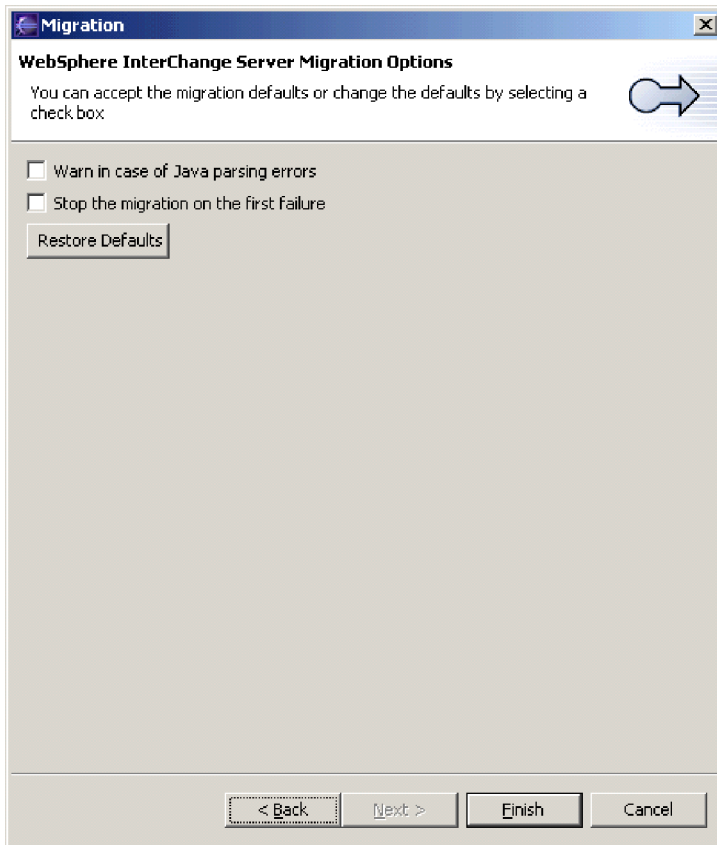


Sie können den Migrationsassistenten auch starten, indem Sie auf die Menüoptionen **'Datei' → 'Importieren'** klicken, die Option für die **WebSphere InterChange Server-JAR-Datei** auswählen und dann auf **'Weiter'** klicken.

3. Der Migrationsassistent wird geöffnet. Geben Sie den Namen der Quelldatei im Feld **'Quellenauswahl'** durch Klicken auf die Schaltfläche **'Durchsuchen'** und Navigieren zur Datei ein. Geben Sie den Modulnamen im entsprechenden Feld ein. Klicken Sie auf **'Weiter'**.



4. Die Fenster mit den Migrationsoptionen wird geöffnet. Hier können Sie die Migrationsstandardeinstellungen akzeptieren oder ein Markierungsfeld zum Ändern der Option auswählen.



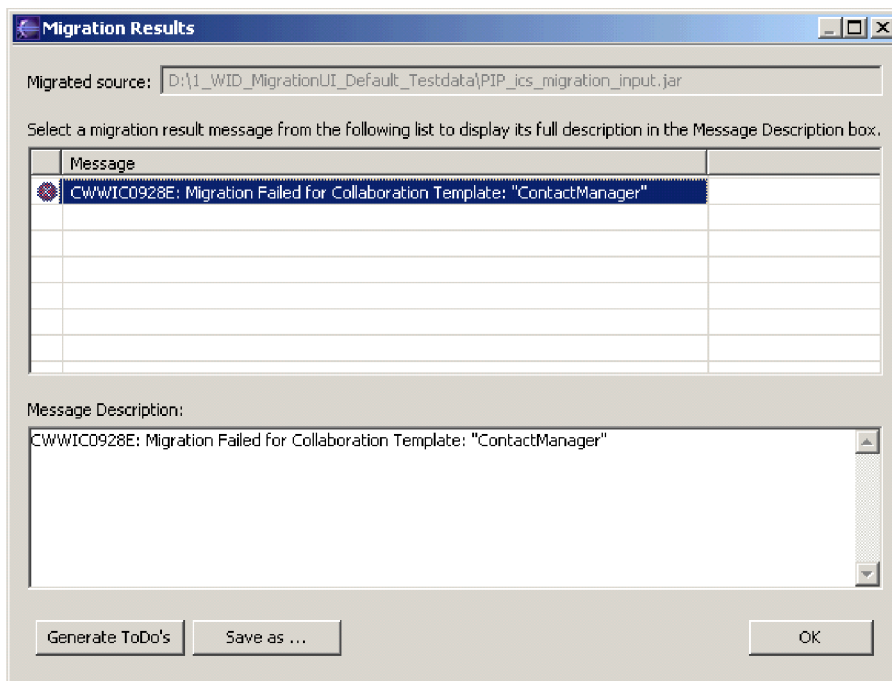
Klicken Sie auf 'Fertig stellen'.

Überprüfung der WebSphere InterChange Server Migration

Wenn während der Migration der WebSphere InterChange Server-Jar-Datei keine Fehler angezeigt werden, war die Migration der Artefakte erfolgreich. Wenn der Migrationsassistent nicht erfolgreich abgeschlossen wurde, wird eine Liste mit Fehlermeldungen, Warnungen und/oder Informationsnachrichten angezeigt. Sie können diese Nachrichten zur Überprüfung der WebSphere InterChange Server-Migration verwenden.

Anmerkung: Durch die Komplexität der Migration von WebSphere InterChange Server zu WebSphere Process Server wird Ihnen dringend empfohlen, extensive Tests der resultierenden Anwendungen, die in WebSphere Process Server ausgeführt werden, durchzuführen, bevor sie in Produktion gegeben werden, um die erwartete Funktionsweise sicherzustellen.

Die folgende Seite wird angezeigt, nachdem der Migrationsassistent beendet wurde:



Im Migrations-Ergebnisfenster wird eine Liste mit Migrationsnachrichten angezeigt. Klicken Sie auf die Nachricht, um eine vollständige Beschreibung der Einzelheiten anzuzeigen. Gegebenenfalls können Sie von dieser Nachrichtenliste eine Liste mit zu korrigierenden Elementen erstellen, indem Sie auf die Schaltfläche **ToDo's generieren** klicken.

Migrationsfehler von WebSphere InterChange Server beheben

Wenn Ihre Migration von WebSphere InterChange Server fehlschlägt, gibt es zwei Möglichkeiten, mit diesen Fehlschlägen umzugehen.

Anmerkung: Möglicherweise ziehen Sie die erste Option vor, da Sie sich anfänglich mit WebSphere InterChange Server besser auskennen. Wenn Sie jedoch mehr Erfahrungen mit WebSphere Process Server und seinen neuen Artefakten sammeln, möchten Sie die migrierten Artefakte in WebSphere Integration Developer möglicherweise reparieren.

1. Wenn die Art des Fehlers es erlaubt, können Sie die WebSphere InterChange Server-Artefakte mit dem WebSphere InterChange Server-Toolset anpassen, die jar-Datei erneut exportieren und die Migration erneut versuchen.
2. Sie können alle Fehler in den resultierenden WebSphere Process Server-Artefakten durch Bearbeiten der Artefakte in WebSphere Integration Developer korrigieren.

Durch Migrationstools verarbeitete WebSphere InterChange Server-Artefakte

Die Migrationstools können einige der Artefakte von WebSphere InterChange Server automatisch migrieren.

Die Migration ist bei den folgenden Artefakten möglich:

- **Geschäftsobjekte** werden zu WebSphere Process Server-Geschäftsobjekten
- **Zuordnungen** werden zu WebSphere Process Server-Zuordnungen
- **Beziehungen** werden zu WebSphere Process Server-Beziehungen und -Aufgabenbereichen
- **Collaboration-Schablonen** werden zu WebSphere Process Server-BPEL und -WSDL
- **Collaboration-Objekte** werden zu WebSphere Process Server-SCA-Artefakten
- **Connectordefinitionen** werden zu WebSphere Process Server-SCA-Artefakten

Die Migrationstools können Ressourcen in WebSphere Process Server für die folgenden WebSphere InterChange Server-Artefakte/Ressourcen automatisch konfigurieren:

- Datenbankverbindungspools
- Beziehungsdatenbanken
- Schemaeinträge

Die Migrationstools verarbeiten die folgenden WebSphere InterChange Server-Artefakte *nicht*:

- Benchmark-Artefakte

Unterstützte WebSphere InterChange Server-APIs

Neben den Migrationstools für WebSphere InterChange Server-Quellenartefakte, die in WebSphere Process Server und WebSphere Integration Developer bereitgestellt werden, gibt es eine Unterstützung für viele APIs, die in WebSphere InterChange Server zur Verfügung standen. Die Migrationstools arbeiten mit diesen WebSphere InterChange Server-APIs zusammen, indem sie den angepassten Snippet-Code bei der Migration so weit wie möglich erhalten.

Anmerkung: Diese APIs werden nur zur Unterstützung von migrierten WebSphere InterChange Server-Anwendungen bereitgestellt, bis diese für die Verwendung der neuen Process Server-APIs geändert werden können. Sämtliche angegebenen WebSphere InterChange Server-APIs sind veraltet und werden in einem künftigen Release entfernt.

Die unterstützten WebSphere InterChange Server-APIs in Process Server sind in Folgenden aufgelistet.

Diese APIs bieten in Process Server ähnliche Funktionen wie in WebSphere InterChange Server.

Funktionsbeschreibungen dieser APIs finden Sie in der Dokumentation von WebSphere InterChange Server.

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)

CwDBConnection

CwDBConnection/

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)
- executeStoredProc(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
- setParamDataTypeJavaObj(String)
- setParamDataTypeJDBC(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)`
- `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 Process Server-DataObject aus WebSphere InterChange Server-XML zuordnen

Falls Sie die Verbindung zu WebSphere Process Server mit den traditionellen Adaptern herstellen, können Sie anhand des folgenden Algorithmus genauer nachvollziehen, wie das WebSphere Process Server-Data-Object aus der WebSphere InterChange Server-XML erstellt wurde. Diese Informationen erläutern, wo die Datenwerte platziert wurden und welche Datenwerte als Ersatz für die in WebSphere InterChange Server verwendeten Datenwerte gewählt wurden.

Allgemeine Informationen

- Um das Verb in 'ChangeSummary' festzulegen, erfolgen alle Einstellungen mit den APIs **markCreate/Update/Delete**.
- Um das Verb in 'ChangeSummary/EventSummary' festzulegen, werden die Verben **Create**, **Update** und **Delete** in 'ChangeSummary' festgelegt, während alle anderen Verben in 'EventSummary' festgelegt werden.
- So wird das Verb aus 'ChangeSummary' abgerufen:
 - Wenn 'isDelete' den Wert 'true' hat, lautet das Verb **Delete**.

Anmerkung: Der Wert von 'isCreate' wird ignoriert, wenn 'isDelete' den Wert 'true' hat. Dies könnte eintreten, falls die Erstellung markiert wurde, bevor das DataObject am Hub eintraf, wo es gelöscht wurde, oder falls sowohl die Erstellung als auch die Löschung im Hub stattfanden.

- Wenn 'isDelete' den Wert 'false' und 'isCreate' den Wert 'true' hat, lautet das Verb **Create**.
- Wenn 'isDelete' den Wert 'false' und 'isCreate' den Wert 'false' hat, lautet das Verb **Update**.
- Damit ein DataObject anstelle des beabsichtigten **Update** nicht als **Create** identifiziert wird, müssen Sie bei aktivierter Protokollierung Folgendes ausführen:
 - Protokollierung während der Erstellung des DataObject aussetzen
 - Protokollierung bei der Aktualisierung des DataObject wieder aufnehmen (oder API **markUpdated** verwenden)

Ladevorgänge

Beim Laden wird eine Laufzeit-XML von WebSphere InterChange Server in eine BusinessGraph AfterImage-Instanz von WebSphere Business Integration geladen.

- Eine Instanz des entsprechenden BusinessGraph wird erstellt.
- Die Protokollierung von 'ChangeSummary' wird aktiviert, damit bei einer späteren Aktivierung die Einträge nicht gelöscht werden.
- Die Protokollierung von 'ChangeSummary' wird angehalten, damit keine unerwünschten Informationen in 'ChangeSummary' aufgenommen werden.
- Die Attribute des BusinessObject der höchsten Ebene werden im DataObject erstellt (siehe Abschnitt "Attributverarbeitung").
- Falls das BusinessObject der höchsten Ebene untergeordnete BusinessObjects enthält, werden diese rekursiv verarbeitet.

- Die Attribute dieser untergeordneten BusinessObjects werden im DataObject erstellt (siehe Abschnitt "Attributverarbeitung").
- Das Verb des BusinessObject der höchsten Ebene wird auf das Verb der höchsten Ebene des Business-Graph gesetzt und in den Zusammenfassungen festgelegt.
- Das Verb der untergeordneten BusinessObjects wird in den Zusammenfassungen festgelegt.

Speichervorgänge

Beim Speichern wird eine BusinessGraph AfterImage-Instanz von WebSphere Business Integration in einer Laufzeit-XML von WebSphere InterChange Server gespeichert. Falls das eingegebene BusinessGraph kein AfterImage ist, wird eine Ausnahmebedingung ausgelöst.

- Es wird eine Instanz des XML-Dokuments von WebSphere InterChange Server erstellt.
- Alle gelöschten DataObjects in 'ChangeSummary' werden so behandelt, als ob sie in ihrer ursprünglichen Position vorhanden wären.

Anmerkung: Bei dieser Rücknahme des Löschrprozesses bleibt die Listenreihenfolge nicht erhalten.

- Das Verb des BusinessObject der höchsten Ebene wird auf das Verb der höchsten Ebene des Business-Graph gesetzt.
- Die Attribute des BusinessObject der höchsten Ebene werden aus dem DataObject festgelegt (siehe Abschnitt "Attributverarbeitung").
- Falls das DataObject untergeordnete DataObjects enthält, werden diese rekursiv verarbeitet.
- Das Verb der untergeordneten DataObjects wird folgendermaßen verarbeitet:
 - Falls für das untergeordnete DataObject in 'EventSummary' oder 'ChangeSummary' kein Verb festgelegt ist, wird das Verb auf "" (leere Zeichenfolge) gesetzt.
 - Falls 'EventSummary', nicht jedoch 'ChangeSummary' ein Verb enthält, wird das Verb verwendet.
 - Falls 'ChangeSummary', nicht jedoch 'EventSummary' ein Verb enthält, wird das Verb verwendet.
 - Falls sowohl 'ChangeSummary' als auch 'EventSummary' ein Verb enthält, hat der Wert in 'ChangeSummary' Vorrang vor dem Wert in 'EventSummary'.
- Die Attribute dieser untergeordneten DataObjects werden im BusinessObject festgelegt (siehe Abschnitt "Attributverarbeitung").

Attributverarbeitung

- Alle Werte, die im Folgenden nicht behandelt werden, werden UNVERÄNDERT geladen/gespeichert.
- ObjectEventId wird aus/in 'EventSummary' geladen/gespeichert.
- Für **CxBlank** und **CxIgnore** gilt:
 - Auf der Konvertierungsseite des BusinessObject von WebSphere Business Integration werden **CxBlank** und 'CxIgnore' folgendermaßen festgelegt/angegeben:
 - **CxIgnore:** Ist nicht festgelegt oder mit dem Java-Wert Null festgelegt
 - **CxBlank:** Wert ist typabhängig (siehe Tabelle unten)
 - Auf der Konvertierungsseite des XML-Dokuments von WebSphere InterChange Server werden **CxBlank** und **CxIgnore** folgendermaßen festgelegt/angegeben:

Tabelle 1. Festlegung von CxBlank und CxIgnore

Typ	CxIgnore	CxBlank
Int	Integer.MIN_VALUE	Integer.MAX_VALUE
Float	Float.MIN_VALUE	Float.MAX_VALUE
Double	Double.MIN_VALUE	Double.MAX_VALUE
String/date/longtext	"CxIgnore"	""
Untergeordnete BusinessObjects	(leeres Element)	nicht verfügbar

Bewährte Verfahren für den WebSphere InterChange Server-Migrationsprozess

Die Richtlinien in diesem Thema sollen Ihnen bei der Entwicklung von Integrationsartefakten für WebSphere InterChange Server helfen. Wenn Sie anhand dieser Richtlinien vorgehen, können Sie die Migration von Artefakten aus WebSphere InterChange Server zu WebSphere Process Server vereinfachen.

Diese Empfehlungen sind nur als Leitfaden für die Entwicklung neuer Integrationslösungen gedacht. Es versteht sich, dass vorhandener Inhalt diese Richtlinie möglicherweise nicht beachtet. Außerdem gibt es immer wieder Fälle, in denen eine Abweichung von diesen Richtlinien erforderlich ist. In solchen Fällen sollten Sie sich so wenig wie möglich von den Richtlinien entfernen, um die zur Migration der Artefakte erforderlichen Nacharbeiten möglichst gering zu halten. Bitte beachten Sie, dass die hier dargestellten Richtlinien die bewährten Verfahren für die Entwicklung von WebSphere InterChange Server-Artefakten im Allgemeinen nicht enthalten. Sie beschränken sich vielmehr auf diejenigen Empfehlungen, die eine künftige Migration von Artefakten vereinfachen könnten.

Allgemeine Entwicklung

Es gibt verschiedene Überlegungen, die für so gut wie die meisten Integrationsartefakte zutreffend sind. Im Allgemeinen werden die Artefakte, die das Funktionsspektrum der Tools nutzen und den durch die Tools umgesetzten Metadatenmodellen entsprechen, problemlos migriert. Artefakte mit deutlichen Erweiterungen und externen Abhängigkeiten hingegen erfordern bei der Migration wahrscheinlich größere manuelle Eingriffe.

Die folgende Liste fasst die bewährten Verfahren für die allgemeine Entwicklung von auf WebSphere InterChange Server basierenden Lösungen zusammen, die die künftige Migration vereinfachen:

- Verwendung von WebSphere InterChange Server für echtzeitorientierte, automatisierte Prozessintegrationslösungen
- Dokumentierung des System- und Komponentenentwurfs
- Bearbeitung von Integrationsartefakten mit den Entwicklungstools
- Definition von Regeln mit den Tools und Java-Snippets gemäß den bewährten Verfahren

Auch wenn dieser Hinweis möglicherweise überflüssig ist: Die Integrationslösungen müssen sich unbedingt an das Programmiermodell und die Architektur halten, die durch WebSphere InterChange Server bereitgestellt werden, denn diese sind für echtzeitorientierte, automatisierte Prozessintegrationslösungen optimal geeignet. Auch die einzelnen Integrationskomponenten in WebSphere InterChange Server haben in der Architektur klar strukturierte Aufgaben. Deutliche Abweichungen von diesem Modell erschweren die Migration von Inhalt in die entsprechenden Artefakte unter WebSphere Process Server.

Ein weiteres bewährtes Verfahren, das den Erfolg künftiger Migrationsprojekte erheblich vergrößert, ist die Dokumentierung des Systementwurfs. Achten Sie darauf, Integrationsarchitektur und -entwurf zu erfassen, einschließlich des funktionalen Entwurfs und der Anforderungen an die Servicequalität (Quality of Service), der gegenseitigen Abhängigkeiten von projektübergreifend genutzten Artefakten und auch der Entwurfsentscheidungen, die während der Implementierung getroffen wurden. Dies vereinfacht die Systemanalyse während der Migration und reduziert ggfs. erforderliche Nacharbeiten.

Bei der Erstellung, Konfiguration und Änderung von Artefaktdefinitionen dürfen nur die bereitgestellten Entwicklungstools verwendet werden. Vermeiden Sie eine manuelle Bearbeitung von Artefaktmetadaten (z. B. die direkte Bearbeitung von XML-Dateien), denn dies kann das Artefakt für die Migration beschädigen.

Bei der Entwicklung von Java-Code in den Collaboration-Schablonen, Zuordnungen, Dienstprogrammen für allgemeinen Code und anderen Komponenten müssen verschiedene Punkte berücksichtigt werden:

- Verwenden Sie nur die veröffentlichten APIs.
- Verwenden Sie den Aktivitätseditor.

- Verwenden Sie Adapter für den EIS-Zugriff.
- Vermeiden Sie externe Abhängigkeiten im Java-Snippet-Code.
- Halten Sie sich an die J2EE-Entwicklungsverfahren für die Portierbarkeit.
- Starten Sie keine Threads, und verwenden Sie keine Basiselemente der Thread-Synchronisation. Für den Fall, dass der Einsatz dieser Elemente zwingend erforderlich ist, müssen sie bei der Migration konvertiert werden, damit sie asynchrone Beans verwenden.
- Führen Sie keine Platten-E/A unter Verwendung von 'java.io.*' aus. Speichern Sie Daten mit JDBC.
- Führen Sie keine Funktionen aus, die möglicherweise für einen EJB-Container reserviert sind, wie beispielsweise Socket-E/A, Laden von Klassen, Laden von nativen Bibliotheken usw. Sind solche Aktionen zwingend erforderlich, müssen diese Snippets bei der Migration manuell konvertiert werden, damit EJB-Containerfunktionen verwendet werden.

Verwenden Sie für die Artefakte nur die in der Produktdokumentation veröffentlichten APIs. Diese APIs sind in den Entwicklungshandbüchern zu WebSphere InterChange Server ausführlich beschrieben. In vielen Fällen werden zwar Kompatibilitäts-APIs in WebSphere Process Server bereitgestellt, aber nur die veröffentlichten APIs sind enthalten. Auch wenn WebSphere InterChange Server viele interne Schnittstellen enthält, deren Verwendung für einen Anwendungsentwickler wünschenswert wäre, kann die künftige Unterstützung dieser Schnittstellen nicht garantiert werden.

Achten Sie bei Analyse/Entwurf von Geschäftslogik und Konvertierungsregeln in Zuordnungen und Collaboration-Schablonen darauf, weitestgehend mit dem Aktivitätseditor zu arbeiten. Dies stellt sicher, dass die Logik durch Metadaten beschrieben wird, die einfacher in die neuen Artefakte migriert werden können. Für Operationen, die Sie in den Tools wiederverwenden wollen, sollten Sie möglichst immer die Funktion für die eigenen Objektgruppen verwenden. Vermeiden Sie nach Möglichkeit über Felder entwickelte Dienstprogrammibliotheken für allgemeinen Code, die als JAR-Datei (Java Archive) in den Klassenpfad von WebSphere InterChange Server aufgenommen werden, da diese manuell migriert werden müssen.

In jedem Java-Code-Snippet, dessen Entwicklung erforderlich ist, sollte der Code so einfach und atomar wie möglich sein. Die Qualität des Java-Codes sollte in der Scripterstellung, der Einbeziehung von Basisauswertungen, Operationen und Berechnungen, der Datenformatierung, der Typkonvertierung u. a. zum Ausdruck kommen. Wenn eine erweiterte und komplexere Anwendungslogik benötigt wird, sollte die Kapselung der Logik in EJBs erwogen werden, deren Ausführung in WebSphere Application Server erfolgt, und der Aufruf der Logik sollte aus WebSphere InterChange Server heraus über Web-Service-Aufrufe vorgenommen werden. Verwenden Sie JDK-Standardbibliotheken anstelle von Fremdanbieter- oder externen Bibliotheken, die möglicherweise separat migriert werden müssen. Fassen Sie außerdem zusammengehörige Logik in einem einzigen Code-Snippet zusammen, und vermeiden Sie die Verwendung von Logik, wenn sich Verbindungs- und Transaktionskontexte über mehrere Code-Snippets hinweg erstrecken können. Beispielsweise sollte sich bei Datenbankoperationen der Code, der mit dem Anfordern einer Verbindung, dem Starten und Beenden einer Transaktion und dem Freigeben der Verbindung zusammenhängt, in einem einzigen Snippet befinden.

Allgemein muss gewährleistet sein, dass Code, der eine Schnittstelle mit einem unternehmensweiten Informationssystem (Enterprise Information System - EIS) bilden soll, in Adapter und nicht in Zuordnungen oder Collaboration-Schablonen gestellt wird. Dies ist generell ein bewährtes Verfahren für den Architekturentwurf. Außerdem werden so Voraussetzungen für Fremdanbieterbibliotheken und zugehörige Überlegungen im Code verringert, beispielsweise die Verbindungsverwaltung und mögliche JNI-Implementierungen (Java Native Interface).

Machen Sie den Code durch Verwendung der passenden Ausnahmebedingungsbehandlung so sicher wie möglich. Achten Sie auch darauf, dass der Code für die Ausführung in einer J2EE-Anwendungsserverumgebung kompatibel ist, auch wenn er gegenwärtig in einer J2SE-Umgebung ausgeführt wird. Halten Sie sich an die J2EE-Entwicklungsverfahren wie beispielsweise die Vermeidung von statischen Variablen, des Startens von Threads und der Platten-E/A. Diese Verfahren sind nicht nur generell ausgezeichnet geeignet, sondern insbesondere für die Portierbarkeit unabdingbar.

Dienstprogramme für allgemeinen Code

Wie bereits erwähnt sollte die Entwicklung von Dienstprogramm-bibliotheken für allgemeinen Code zur übergreifenden Nutzung in Integrationsartefakten in der WebSphere InterChange Server-Umgebung nach Möglichkeit vermieden werden. In Situationen, in denen die Wiederverwendung von Code über Integrationsartefakte hinweg erforderlich ist, empfiehlt sich die Nutzung der Funktion für die eigenen Objektgruppen im Aktivitätseditor. In Betracht zu ziehen ist außerdem die Kapselung der Logik in EJBs, deren Ausführung in WebSphere Application Server erfolgt, und der Aufruf der Logik aus WebSphere InterChange Server heraus über Web-Service-Aufrufe. Es ist zwar möglich, dass einige Dienstprogramm-bibliotheken für allgemeinen Code unter WebSphere Process Server einwandfrei ausgeführt werden, für die Migration der angepassten Dienstprogramme ist jedoch der Benutzer zuständig.

Datenbankverbindungspools

Benutzerdefinierte Datenbankverbindungspools sind in Zuordnungen und Collaboration-Schablonen zur einfachen Datensuche und auch im Hinblick auf eine ausgereifte prozessinstanzübergreifende Statusverwaltung überaus nützlich. Ein Datenbankverbindungspool in WebSphere InterChange Server wird in WebSphere Process Server als JDBC-Standardressource wiedergegeben, und die grundlegende Funktion ist identisch. Es kann jedoch Unterschiede darin geben, wie Verbindungen und Transaktionen verwaltet werden.

Um eine künftige Portierbarkeit zu maximieren, sollten Datenbanktransaktionen über Java-Snippet-Knoten hinweg in einer Collaboration-Schablone oder Zuordnung nicht aktiv bleiben. Beispielsweise sollte sich der Code, der mit dem Anfordern einer Verbindung, dem Starten und Beenden einer Transaktion und dem Freigeben der Verbindung zusammenhängt, in einem einzigen Snippet befinden.

Geschäftsobjekte

Primär ist bei der Entwicklung von Geschäftsobjekten wichtig, dass nur die bereitgestellten Tools für die Konfiguration von Artefakten verwendet werden, dass für Datenattribute explizite Datentypen und Längen eingesetzt werden und dass ausschließlich dokumentierte APIs zum Einsatz kommen.

Geschäftsobjekte in WebSphere Process Server basieren auf SDOs (Service Data Object - Servicedatenobjekt), die stark typisierte Datenattribute verwenden. Bei Geschäftsobjekten in WebSphere InterChange Server und Adaptern sind die Datenattribute nicht stark typisiert, und Benutzer geben manchmal Zeichenfolgedatentypen für Datenattribute an, die nicht aus Zeichenfolgen bestehen. Um Probleme bei WebSphere Process Server zu vermeiden, müssen Sie in der Spezifikation der Datentypen explizit sein.

Da Geschäftsobjekte in WebSphere Process Server möglicherweise in der Laufzeit serialisiert werden, wenn sie zwischen Komponenten übergeben werden, ist es von erheblicher Bedeutung, die erforderlichen Längen für Datenattribute explizit anzugeben, um die Auslastung der Systemressourcen so gering wie möglich zu halten. Aus diesem Grund sollte beispielsweise für ein Zeichenfolgenattribut nicht die maximale Länge von 255 Zeichen verwendet werden. Auch die Angabe von Attributen ohne Länge, was derzeit einem Standardwert von 255 Zeichen entspricht, ist zu vermeiden. Geben Sie stattdessen die erforderliche Länge für Attribute präzise an.

Für die Namen von Geschäftsobjektattributen gelten in WebSphere Process Server die XSD-NCName-Regeln. Verwenden Sie daher in Namen von Geschäftsobjektattributen keine Leerzeichen oder Doppelpunkte ":". Namen von Geschäftsobjektattributen, die Leerzeichen oder Doppelpunkte ":" enthalten, sind in WebSphere Process Server ungültig. Benennen Sie Geschäftsobjektattribute vor der Migration um.

Wenn Sie in einem Geschäftsobjekt eine Feldgruppe verwenden, können Sie sich bei der Indexierung der Feldgruppe in Zuordnungen und/oder Beziehungen nicht auf die Reihenfolge der Feldgruppe verlassen. Die Anweisung, die Feldgruppen in WebSphere Process Server migriert, kann die Indexreihenfolge - insbesondere bei gelöschten Einträgen - nicht garantieren.

Wie bereits an früherer Stelle erwähnt, ist es wichtig, Geschäftsobjektdefinitionen ausschließlich mit dem Entwurfstool für Geschäftsobjekte zu bearbeiten und nur die veröffentlichten APIs für Geschäftsobjekte in Integrationsartefakten zu verwenden.

Collaboration-Schablonen

Viele der zuvor beschriebenen Richtlinien gelten auch für die Entwicklung von Collaboration-Schablonen.

Um sicherzustellen, dass Prozesse korrekt mit Metadaten beschrieben werden, verwenden Sie zur Erstellung und Änderung von Collaboration-Schablonen immer das Tool 'Process Designer', und vermeiden Sie es, die Metadateien manuell zu bearbeiten. Verwenden Sie nach Möglichkeit immer den Aktivitätseditor, um den Einsatz von Metadaten bei der Beschreibung der erforderlichen Logik zu maximieren.

Verwenden Sie in den Collaboration-Schablonen nur die dokumentierten APIs, denn dies hält die bei der Migration möglicherweise erforderliche manuelle Nacharbeit gering. Verwenden Sie keine statischen Variablen. Verwenden Sie stattdessen nicht statische Variablen und Collaboration-Eigenschaften, um die Anforderungen der Geschäftslogik umzusetzen. Vermeiden Sie den Einsatz der Java-Qualifikationsmerkmale 'final', 'transient' und 'native' in Java-Snippets. Diese Qualifikationsmerkmale können in den BPEL-Java-Snippets, die bei der Migration der Collaboration-Schablonen entstehen, nicht umgesetzt werden.

Um die künftige Portierbarkeit zu maximieren, verwenden Sie nach Möglichkeit keine expliziten Aufrufe für die Verbindungsfreigabe und expliziten Transaktionsangaben (d. h. explizite COMMIT-Operationen & explizite ROLLBACK-Operationen) für benutzerdefinierte Datenbankverbindungspools. Nutzen Sie stattdessen die containergesteuerte implizite Verbindungsbereinigung und die implizite Transaktionsangabe. Vermeiden Sie des Weiteren Systemverbindungen und Transaktionen, die in einer Collaboration-Schablone über Java-Snippet-Knoten hinweg aktiv sind. Dies gilt für alle Verbindungen zu einem externen System und auch für die benutzerdefinierten Datenbankverbindungspools. Wie bereits erläutert, sollten Operationen mit einem externen EIS-System innerhalb eines Adapters verwaltet werden, und der Code für Datenbankoperationen sollte in einem einzigen Code-Snippet enthalten sein. Dies kann in einer Collaboration erforderlich sein, die bei der Wiedergabe als BPEL-Geschäftsprozesskomponente möglicherweise als unterbrechbarer Ablauf selektiv implementiert wird. In diesem Fall kann der Prozess aus mehreren separaten Transaktionen zusammengesetzt sein, wobei nur Informationen zum Status und zu globalen Variablen zwischen den Aktivitäten übergeben werden. Der Kontext für alle Systemverbindungen zu zugehörigen Transaktionen, die sich über diese Prozesstransaktionen erstrecken, würde verloren gehen.

Verwenden Sie in den Namen von Eigenschaften für Collaboration-Schablonen keine Sonderzeichen. Sonderzeichen sind in den BPEL-Eigenschaftennamen, in die sie migriert werden, ungültig. Benennen Sie vor der Migration Eigenschaften um, und entfernen Sie diese Sonderzeichen aus den Namen, um Syntaxfehler in der durch die Migration generierten BPEL zu verhindern.

Verweisen Sie nicht mithilfe von "this." auf Variablen. Verwenden Sie an Stelle von "this.inputBusObj" einfach "inputBusObj".

Verwenden Sie für Variablen die Gültigkeit auf Klassenebene anstelle des Gültigkeitsbereichs für Szenarien. Der Gültigkeitsbereich für Szenarien wird während der Migration nicht weitergeleitet.

Initialisieren Sie alle Variablen, die in Java-Snippets deklariert sind, mit einem Standardwert, z. B. "Object myObject = null;". Achten Sie darauf, dass vor der Migration alle Variablen während der Deklaration initialisiert werden.

Vergewissern Sie sich, dass die Abschnitte der Collaboration-Schablone, die vom Benutzer geändert werden können, keine Java-Importanweisungen enthalten. Verwenden Sie in der Definition der Collaboration-Schablone die Importfelder, um die zu importierenden Java-Pakete anzugeben.

Zuordnungen

Viele der für Collaboration-Schablonen beschriebenen Richtlinien gelten auch für Zuordnungen.

Um sicherzustellen, dass Zuordnungen korrekt mit Metadaten beschrieben werden, verwenden Sie zur Erstellung und Änderung von Zuordnungen immer das Zuordnungsentwurfstool, und vermeiden Sie es, die Metadatendateien manuell zu bearbeiten. Verwenden Sie nach Möglichkeit immer den Aktivitätseditor, um den Einsatz von Metadaten bei der Beschreibung der erforderlichen Logik zu maximieren.

Wenn Sie in einer Zuordnung auf ein untergeordnetes Geschäftsobjekt verweisen, verwenden Sie für das untergeordnete Geschäftsobjekt eine Unterzuordnung.

Vermeiden Sie die Verwendung von Java-Code als "Wert" in einer Anweisung SET, da dies in WebSphere Process Server nicht gültig ist. Verwenden Sie stattdessen Konstanten. Wenn der SET-Wert `"xml version=" + "1.0" + " encoding=" + "UTF-8"` lautet, wird dies in WebSphere Process Server nicht ausgewertet. Ändern Sie diese Angabe in `"xml version=1.0 encoding=UTF-8"`, bevor Sie die Migration ausführen.

Verwenden Sie in den Collaboration-Schablonen nur die dokumentierten APIs, denn dies hält die bei der Migration möglicherweise erforderliche manuelle Nacharbeit gering. Verwenden Sie keine statischen Variablen. Verwenden Sie stattdessen nicht statische Variablen und Collaboration-Eigenschaften, um die Anforderungen der Geschäftslogik umzusetzen. Vermeiden Sie den Einsatz der Java-Qualifikationsmerkmale `'final'`, `'transient'` und `'native'` in Java-Snippets.

Wenn Sie in einem Geschäftsobjekt eine Feldgruppe verwenden, können Sie sich bei der Indexierung der Feldgruppe in Zuordnungen nicht auf die Reihenfolge der Feldgruppe verlassen. Die Anweisung, die Feldgruppen in WebSphere Process Server migriert, kann die Indexreihenfolge - insbesondere bei gelöschten Einträgen - nicht garantieren.

Um die künftige Portierbarkeit zu maximieren, verwenden Sie nach Möglichkeit keine expliziten Aufrufe für die Verbindungsfreigabe und expliziten Transaktionsangaben (d. h. explizite COMMIT-Operationen & explizite ROLLBACK-Operationen) für benutzerdefinierte Datenbankverbindungspools. Nutzen Sie stattdessen die containergesteuerte implizite Verbindungsbereinigung und die implizite Transaktionsangabe. Vermeiden Sie des Weiteren Systemverbindungen und Transaktionen, die über die Grenzen von Transaktionsknoten hinweg in Java-Snippet-Code aktiv sind. Dies gilt für alle Verbindungen zu einem externen System und auch für die benutzerdefinierten Datenbankverbindungspools. Wie bereits erläutert, sollten Operationen mit einem externen EIS-System innerhalb eines Adapters verwaltet werden, und der Code für Datenbankoperationen sollte in einem einzigen Code-Snippet enthalten sein.

Beziehungen

Denken Sie bei Beziehungen daran, dass Beziehungsdefinitionen zwar für die Verwendung in WebSphere Process Server migriert werden können, das Beziehungstabellenschema und die Instanzdaten jedoch möglicherweise von WebSphere Process Server wiederverwendet und außerdem von WebSphere InterChange Server und WebSphere Process Server gleichzeitig gemeinsam genutzt werden können.

Bei Beziehungen ist es am wichtigsten, nur die bereitgestellten Tools für die Konfiguration der zusammengehörigen Komponenten zu verwenden und ausschließlich die veröffentlichten APIs für Beziehungen in Integrationsartefakten einzusetzen.

Beziehungsdefinitionen sollten nur mit dem Tool 'Relationship Designer' bearbeitet werden. Außerdem sollte nur WebSphere InterChange Server die Konfiguration des Beziehungsschemas ermöglicht werden, das bei der Implementierung von Beziehungsdefinitionen automatisch generiert wird. Ändern Sie das Beziehungstabellenschema nicht direkt mit Datenbanktools oder SQL-Prozeduren.

Falls Sie Beziehungsinstanzdaten im Beziehungstabellenschema ändern müssen, achten Sie außerdem darauf, die von Relationship Designer bereitgestellten Funktionen zu verwenden.

Wie bereits erwähnt, dürfen nur die veröffentlichten APIs für Beziehungen in Integrationsartefakten verwendet werden.

Zugriffs-Framework-Clients

Entwickeln Sie keine neuen Clients, die die APIs der CORBA-IDL-Schnittstelle übernehmen. Dies wird in WebSphere Process Server nicht unterstützt.

Migration zu WebSphere Integration Developer von WebSphere MQ Workflow

WebSphere Integration Developer bietet die für die Migration von WebSphere MQ Workflow notwendigen Tools.

Der Migrationsassistent ermöglicht Ihnen die Konvertierung von FDL-Definitionen von Geschäftsprozessen, die Sie aus der Buildtime-Komponente von WebSphere MQ Workflow in entsprechende Artefakte des Geschäftsprozess-Choreographers exportiert haben. Die generierten Artefakte von Business Process Choreographer enthalten XMLSchema-Definitionen für Geschäftsobjekte, WSDL-Definitionen, BPEL und TEL-Definitionen.

Das Konvertierungstool erfordert eine semantisch komplette FDL-Definition eines Prozessmodells, das Sie aus WebSphere MQ Workflow Buildtime mit der Option '**tief exportieren**' exportieren. Diese Option stellt sicher, dass alle notwendigen Daten, Programme und Unterprozess-Spezifikationen enthalten sind. Stellen Sie außerdem sicher, dass alle benutzerdefinierten Prozessausführungs-Servderdefinitionen (UPES), auf die in Ihrem WebSphere MQ Workflow-Prozessmodell verwiesen wird, auch ausgewählt werden, wenn Sie FDL aus WebSphere MQ Workflow Buildtime exportieren.

Anmerkung: Der Migrationsassistent beinhaltet nicht die Migration von:

- WebSphere MQ Workflow-Laufzeitinstanzen
- Programmanwendungen, die von einem WebSphere MQ Workflow-Programmausführungsagenten (PEA) oder WebSphere MQ Workflow-Prozessausführungsserver (PES für z/OS) aufgerufen werden

Vorbereitung für die Migration von WebSphere MQ Workflow

Vor der Migration zu WebSphere Integration Developer von WebSphere MQ Workflow, stellen Sie sicher, dass Sie Ihre Umgebung ordnungsgemäß vorbereitet haben.

Der Geltungsbereich und die Vollständigkeit der Zuordnung hängt davon ab, wie genau Sie den folgenden Migrationsrichtlinien folgen:

- Stellen Sie sicher, dass FDL-Programmaktivitäten einem benutzerdefinierten Prozessausführungsserver(UPES) zugeordnet sind, wenn es sich nicht um reine **staff**-Aktivitäten handelt.
- Stellen Sie sicher, dass Mitarbeiterzuordnungen für WebSphere MQ Workflow-Programmaktivitäten mit TEL-Standard-**Mitarbeiterverben** kompatibel sind.
- Verwenden Sie kurze und einfache Namen, um die Lesbarkeit migrierter Prozessmodelle zu verbessern. Beachten Sie, dass es sich bei FDL-Namen um illegale BPEL-Namen handeln kann. Der Migrationsassistent unterstützt Sie bei der automatischen Konvertierung von FDL-Namen in gültige BPEL-Namen.

Der Migrationsassistent erstellt syntaktisch korrekte Geschäftsprozesseditoranweisungen, selbst für WebSphere MQ Workflow-Anweisungen, die nicht migriert werden können (PEA- oder PES-Programmaktivitäten, bestimmte dynamische Mitarbeiterzuordnungen usw.), die manuell an ausführbare Geschäftsprozesseditorartefakte angepasst werden müssen.

In der folgenden Tabelle werden die angewendeten Zuordnungsregeln aufgelistet:

Tabelle 2. Zuordnungsregeln

WebSphere MQ Workflow	Geschäftsprozess-Choreographer
Prozess	<i>Process with execution mode: longRunning; Partner links</i> für eingehende und ausgehende Schnittstellen des Prozesses
Quelle und Datensenke	<i>Variables</i> für Prozesseingabe und Prozessausgabe; <i>Receive</i> und <i>Reply</i> -Aktivität
Program-Aktivität	<i>Invoke</i> -Aktivität
Process-Aktivität	<i>Invoke</i> -Aktivität
Leervorgang	<i>Leer</i> -vorgang
Block	<i>Geltungsbereich</i> mit eingebetteten BPEL-Aktivitäten
Exit-Bedingung von Aktivität	<i>While</i> -Aktivität (unter Einschluss der aktuellen Aktivität)
Startbedingung der Aktivität	<i>Verknüpfungsbedingung</i> der Aktivität
Mitarbeiterzuordnung der Aktivität	<i>Human Task</i> -Aktivität
Eingabecontainer und Ausgabecontainer der Aktivität	<i>Variables</i> zur Angabe der Ein-/Ausgabe der <i>invoke</i> -Aktivität
Steuerungsstecker; Übergangsbedingung	<i>Link</i> ; <i>Übergangsbedingung</i>
Datenstecker	<i>Assign</i> -Aktivität
Globaler Datencontainer	<i>Variable</i>

Sie sollten den Migrationsprozess wenn möglich zunächst mit kleinen Projekten beginnen. Der Migrationsassistent vereinfacht die Konvertierung Ihrer WebSphere MQ Workflow-Prozessmodelle in Geschäftsprozesseditor-Prozessmodelle, bedenken Sie jedoch, dass die Prozesse nicht eins-zu-eins zugeordnet werden können, da Sie ein neues Programmiermodell erstellen. Die semantischen Geltungsbereiche der zugrundeliegenden Prozess-Spezifikationssprachen (FDL und BPEL) haben einen gemeinsamen Schnittpunkt, überschneiden sich jedoch nicht vollständig. Ansonsten hätte der Geschäftsprozesseditor keine neuen Vorteile zu bieten. Web-Services stellen eine vielversprechende neue Technologie dar, die den Ersatz veralteter Lösungen mit neuen anbieten.

Sie sollten die generierten Artefakte generell immer überprüfen und wenn nötig modifizieren. Möglicherweise sind zusätzliche Anstrengungen nötig, eine erfolgreiche Migration entweder zu ermöglichen oder die Migrationstask abzuschließen.

Migration von WebSphere MQ Workflow mithilfe des Migrationsassistenten

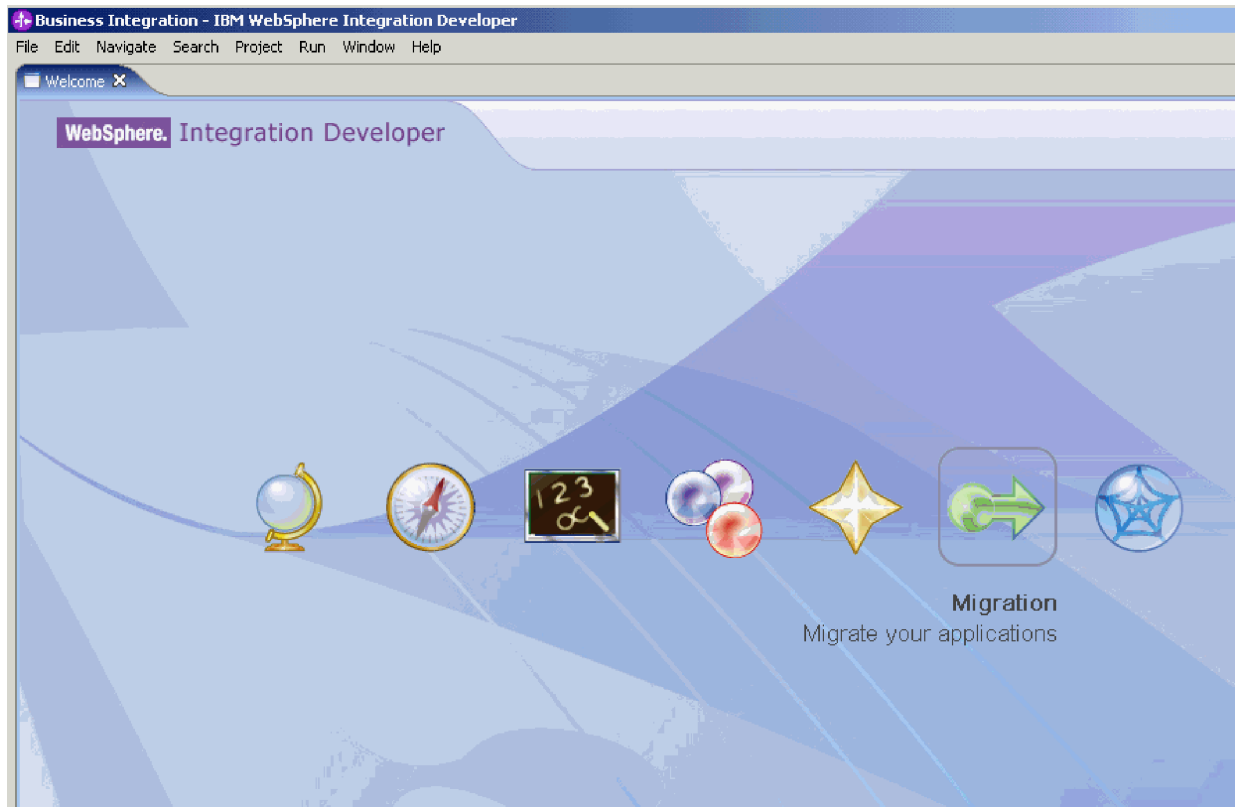
Der Migrationsassistent ermöglicht Ihnen die Konvertierung von FDL-Definitionen von Geschäftsprozessen, die Sie aus der Buildtime-Komponente von WebSphere MQ Workflow in entsprechende Artefakte des Geschäftsprozess-Choreographers exportiert haben. Die generierten Artefakte von Business Process Choreographer enthalten XMLSchema-Definitionen für Geschäftsobjekte, WSDL-Definitionen, BPEL und TEL-Definitionen.

Anmerkung: Der Migrationsassistent beinhaltet nicht die Migration von:

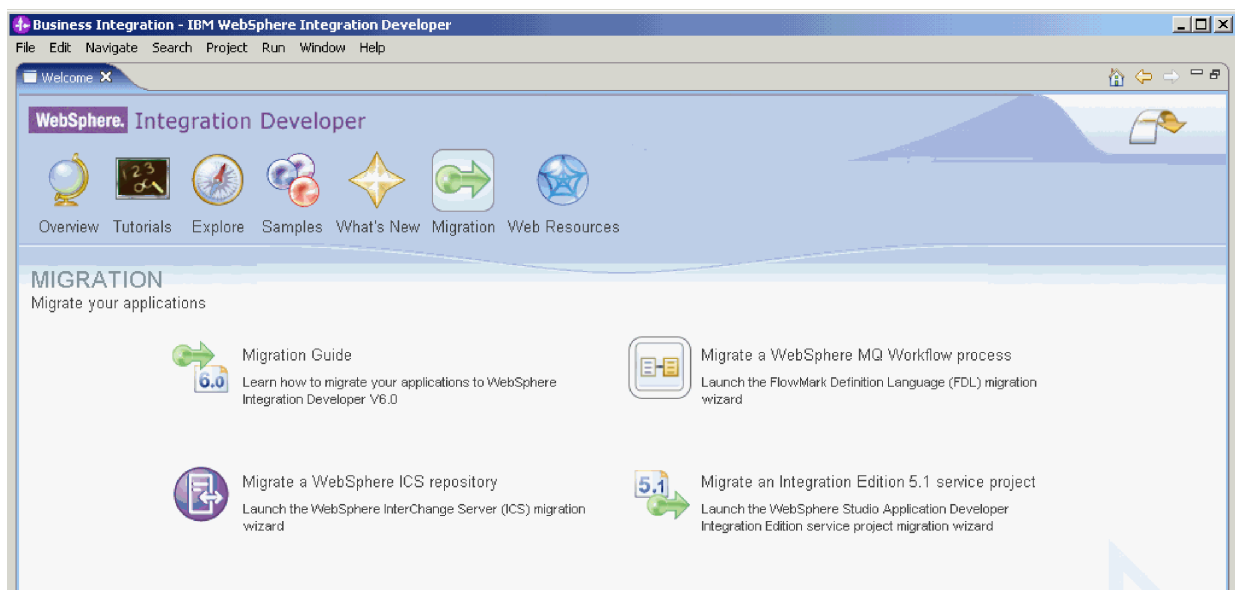
- WebSphere MQ Workflow-Laufzeitinstanzen
- Programmanwendungen, die von einem WebSphere MQ Workflow-Programmausführungsagenten (PEA) oder WebSphere MQ Workflow-Prozessausführungsserver (PES für z/OS) aufgerufen werden

Befolgen Sie diese Schritte zur Verwendung des Migrationsassistenten zur Migration Ihrer WebSphere MQ Workflow-Artefakte:

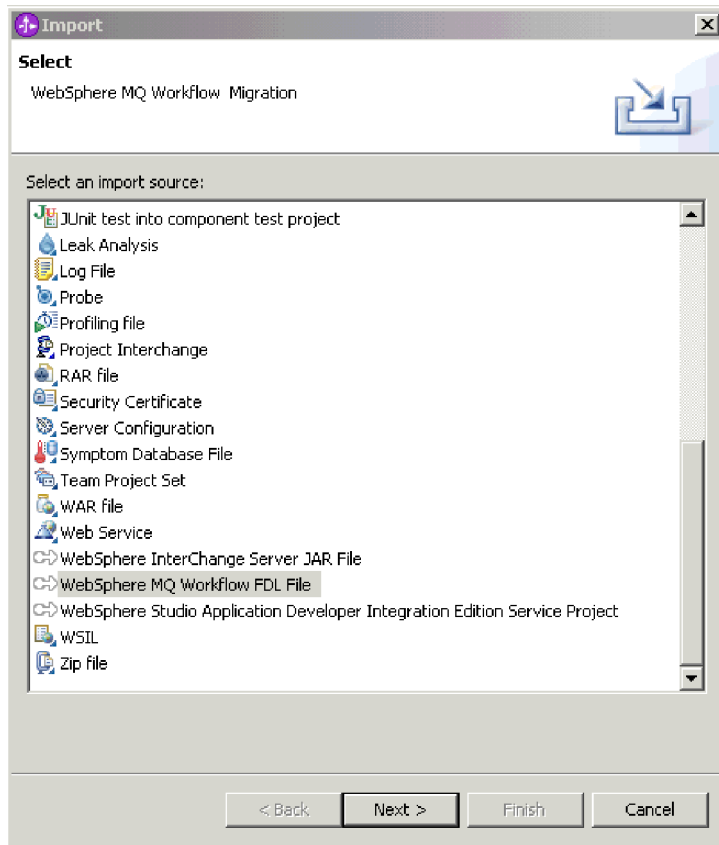
1. Klicken Sie auf der Willkommensseite auf das Symbol , um die Seite 'Migration' zu öffnen.



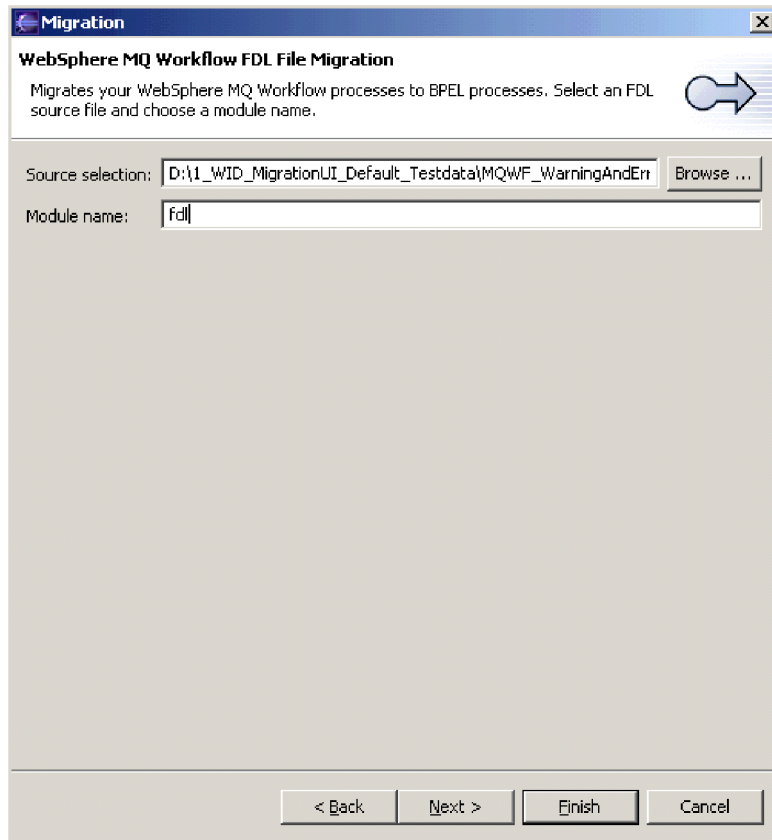
2. Wählen Sie auf der Seite 'Migration' die Option für die Migration eines WebSphere MQWorkflow-Prozesses:



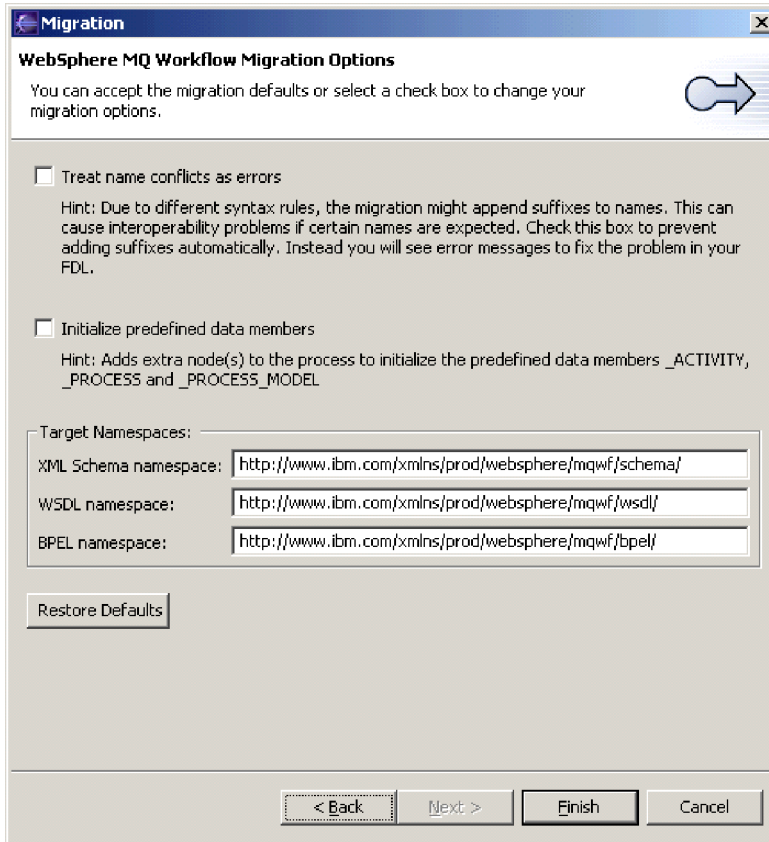
(Hinweis: Sie können den Migrationsassistenten auch durch Klicken auf 'Datei' → 'Importieren' → 'WebSphere MQ Workflow FDL-Datei' öffnen:



3. Der Migrationsassistent wird geöffnet. Geben Sie den Namen der .fdl-Datei im Feld **'Quellenauswahl'** durch Klicken auf die Schaltfläche **'Durchsuchen'** und Navigieren zur Datei ein. Geben Sie den Modulnamen im entsprechenden Feld ein. Klicken Sie auf **'Weiter'**.



4. Die Seite mit den Migrationsoptionen wird geöffnet. Hier können Sie die Migrationsstandardeinstellungen akzeptieren oder ein Markierungsfeld zum Ändern der Option auswählen. Durch Auswahl des Markierungsfelds '**Namensunverträglichkeiten als Fehler behandeln**' können Sie das automatische Hinzufügen von Suffixen verhindern, was zu Interoperabilitätsfehlern führen könnte. Das Markierungsfeld '**Vordefinierte Datenmember initialisieren**' fügt zusätzliche Knoten zum Prozess hinzu, um die vordefinierten Datenmember zu initialisieren.



Migration

WebSphere MQ Workflow Migration Options

You can accept the migration defaults or select a check box to change your migration options.

☐ 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

Target Namespaces:

XML Schema namespace:

WSDL namespace:

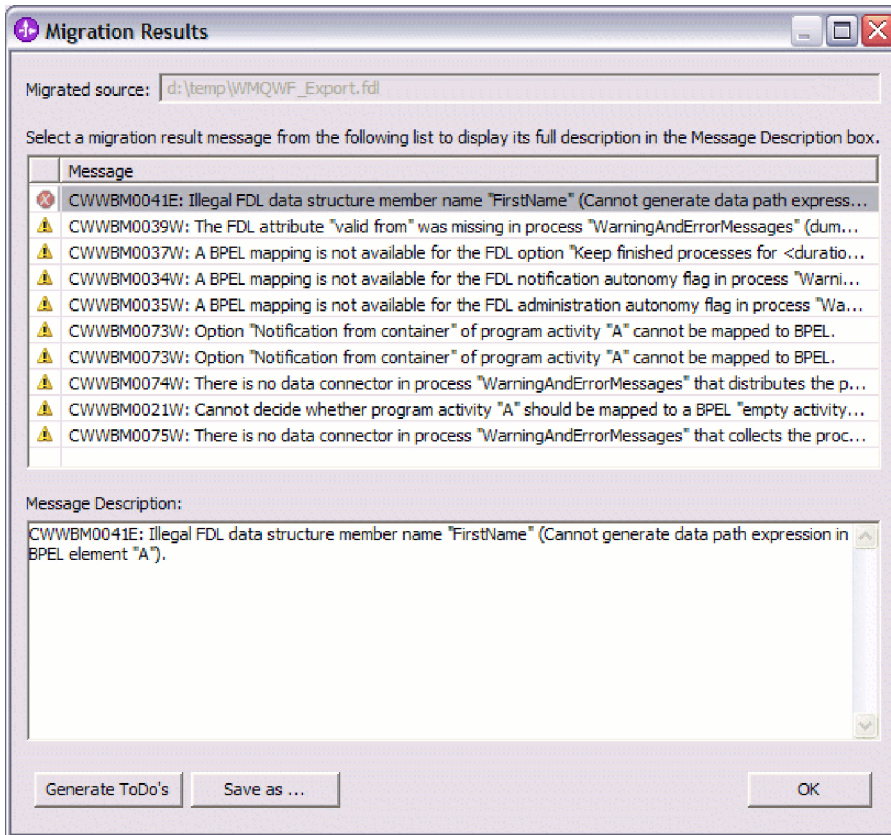
BPEL namespace:

Klicken Sie auf 'Fertig stellen'.

Überprüfung der WebSphere MQ Workflow Migration

Wenn der Migrationsassistent erfolgreich abgeschlossen wurde, wird eine Liste mit Fehlermeldungen, Warnungen und/oder Informationsnachrichten angezeigt. Sie können diese Nachrichten zur Überprüfung der WebSphere MQ Workflow-Migration verwenden.

Die folgende Seite wird angezeigt, nachdem der Migrationsassistent beendet wurde:



Im Migrations-Ergebnisfenster wird eine Liste mit Migrationsnachrichten angezeigt. Klicken Sie auf die Nachricht, um eine vollständige Beschreibung der Einzelheiten anzuzeigen. Gegebenenfalls können Sie von dieser Nachrichtenliste eine Liste mit zu korrigierenden Elementen erstellen, indem Sie auf die Schaltfläche **ToDo's generieren** klicken.

Einschränkungen des Migrationsprozesses (von WebSphere MQ Workflow)

Es gibt bestimmte Einschränkungen für den WebSphere MQ Workflow-Migrationsprozess.

Die Migration von FDL generiert Aufrufaktivitäten für UPES-Aktivitäten und die entsprechenden WSDLs. Die Laufzeitumgebung unterscheidet sich allerdings erheblich zwischen IBM WebSphere MQ Workflow und IBM WebSphere Process Server in Bezug auf die Verfahren, die für die Korrelation von Aufrufnachrichten und deren Antworten verwendet werden. Daher kann die generierte BPEL nicht erfolgreich ausgeführt werden, wenn keine UPES-Kompatibilitätsschicht verfügbar ist.

Migration von Quellenartefakten zu WebSphere Integration Developer von WebSphere Studio Application Developer Integration Edition

Quellenartefakte können von WebSphere Studio Application Developer Integration Edition zu WebSphere Integration Developer migriert werden. Die Migration von Quellenartefakten in eine Anwendung erfordert eine Migration dieser zum neuen WebSphere Integration Developer-Programmiermodell, so dass neue Funktionalitäten/Funktionen verwendet werden können. Die Anwendung kann daraufhin in WebSphere Integration Developer Version 6.0 Server neu implementiert und installiert werden.

Viele der in WebSphere Business Integration Server Foundation 5.1 verfügbaren Funktionen sind nach unten in den WebSphere-Basis-Anwendungsserver 6.0 verschoben worden. Tipps zur Migration dieser Funktionen finden Sie auf dieser Site: http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.websphere.base.doc/info/aes/ae/rins_migratpme.html

Zur kompletten Migration eines WebSphere Studio Application Developer Integration Edition-Serviceprojekts müssen zwei wichtige Tasks erfüllt werden:

1. Verwenden Sie den Migrationsassistenten zur automatischen Migration dieser Artefakte zum Geschäftsintegrations-Modulprojekt.
2. Verwenden Sie den WebSphere Integration Developer für den manuellen Abschluss dieser Migration. Dies schließt das Korrigieren von Java-Code ein, der nicht automatisch migriert werden konnte, sowie die Neuvernetzung von Artefakten.

Anmerkung: Die Laufzeitmigration (Upgradepfad) wird in WebSphere Process Server 6.0.0 nicht zur Verfügung gestellt, daher ist dieser Quellenartefakt-Migrationspfad die einzige Option für die Migration der WebSphere Studio Integration Edition-Serviceprojekte in 6.0.0.

Unterstützte Migrationspfade für die Migration von Quellenartefakten

Vor Beginn der Migration von Quellenartefakten von WebSphere Studio Application Developer Integration Edition, überprüfen Sie die unterstützten Migrationspfade, die von WebSphere Integration Developer unterstützt werden.

Der Migrationsassistent bietet die Möglichkeit der Migration eines WebSphere Studio Application Developer Integration Edition Version 5.1 (oder höher) -Serviceprojekts zur Zeit. Es wird *kein* kompletter Arbeitsbereich migriert.

Der Migrationsassistent migriert keine Anwendungs-Binaries - er migriert nur Quellenartefakte, die in einem WebSphere Studio Application Developer Integration Edition-Serviceprojekt enthalten sind.

Quellenartefakte für die Migration vorbereiten

Stellen Sie vor der Migration von Quellenartefakten zu WebSphere Integration Developer von WebSphere Studio Application Developer Integration Edition sicher, dass Sie Ihre Umgebung ordnungsgemäß vorbereitet haben.

In den folgenden Schritten wird beschrieben, wie Sie Ihre Umgebung vor der Migration von Quellenartefakten zu WebSphere Integration Developer von WebSphere Studio Application Developer Integration Edition vorbereiten:

1. Stellen Sie vor der Migration sicher, dass Sie über eine Sicherungskopie des gesamten 5.1 Arbeitsbereichs verfügen.
2. Die beste Möglichkeit für die Migration der nicht-WBI-spezifischen Projekte in Ihren Arbeitsbereich können Sie im Migrationsabschnitt des Rational Application Developer Information Center unter folgendem Pfad bestimmen: <http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.rad.migration.doc/topics/tmigratefrom51x.html>
3. Hintergrundinformationen zur Web-Service-Funktionalität in Rational Application Developer finden Sie im Web-Serviceabschnitt des Rational Application Developer Information Center unter: <http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.webservices.rad.nav.doc/developingweb.html>
4. Stellen Sie sicher, dass alle geeigneten WebSphere Integration Developer-Funktionen aktiviert sind. Wenn diese Funktionen bei Ihnen nicht aktiviert sind, können Sie möglicherweise nicht alle Menüoptionen sehen, die unten beschrieben werden. So aktivieren Sie die wichtigen Funktionen:
 - Gehen Sie im WebSphere Integrationsentwickler zum Menüelement **'Fenster'** und wählen Sie **'Benutzervorgaben'**.
 - Gehen Sie zum **Arbeitsbereich** und wählen Sie die Kategorie **'Leistungsmerkmale'**.
 - Wählen Sie alle Funktionen in den folgenden Kategorien:
 - Advanced J2EE
 - Enterprise Java
 - Integration Developer
 - Java Developer

- Web Developer (typisch)
 - Web Service Developer
 - XML Developer
 - Klicken Sie auf **OK**.
5. Verwenden Sie ein neues Arbeitsbereichsverzeichnis für WebSphere Integration Developer. Es wird abgeraten, WebSphere Integration Developer in einem alten WebSphere Studio Application Developer Integration Edition-Arbeitsbereich zu öffnen, der Serviceprojekte enthält, da diese Projekte zunächst in ein Format migriert werden müssen, das von WebSphere Integration Developer gelesen werden kann. Die empfohlenen Schritte für diese Vorgehensweise sind folgende:
- a. Kopieren Sie alle nicht-Serviceprojekte aus dem alten in den neuen Arbeitsbereich. Kopieren Sie *nicht* die mit 5.1 erstellten EJB-, Web- und EAR-Projekte, wenn Implementierungscode für ein 5.1-Serviceprojekt erstellt wurde. Der neue 6.0 Implementierungscode wird automatisch neu generiert, wenn das BI-Modul erstellt wird.
 - b. Öffnen Sie WebSphere Integration Developer im leeren Arbeitsbereich und importieren Sie alle nicht-Serviceprojekte durch Klicken auf **'Datei' → 'Importieren' → 'Vorhandenes Projekt in Arbeitsbereich'** und wählen Sie die Projekte, die Sie in den neuen Arbeitsbereich kopiert haben.
 - Wenn es sich beim Projekt um ein J2EE-Projekt handelt, sollten Sie es durch Verwendung des Rational Application Developer-Migrationsassistenten auf die Ebene von 1.4 migrieren:
 - 1) Klicken Sie mit der rechten Maustaste auf das Projekt, und wählen Sie die Option **'Migration' → 'J2EE-Migrationsassistent...'**.
 - 2) Überprüfen Sie die Warnmeldungen auf der ersten Seite und klicken Sie auf **'Weiter'**.
 - 3) Stellen Sie sicher, dass das **'J2EE-Projekt'** in der Projektliste markiert ist. Lassen Sie **'Projektstruktur migrieren'** und **'J2EE-Spezifikationsebene'** markiert. Wählen Sie J2EE-Version **1.4** und **'Zielserver WebSphere Process Server V6.0'**.
 - 4) Wählen Sie alle sonstigen Optionen, die für Ihr J2EE-Projekt geeignet sind, und klicken Sie auf **'Fertig stellen'**. Wenn dieser Schritt erfolgreich abgeschlossen wird, wird folgende Nachricht angezeigt: **Migration erfolgreich abgeschlossen**.
 - 5) Wenn nach der Migration Fehler im J2EE-Projekt vorhanden sind, entfernen Sie alle Klassenpfadeinträge, die auf V5 .jar-Dateien oder Bibliotheken verweisen und fügen Sie stattdessen die **JRE-Systembibliothek** und die **WPS-Serverziel-Bibliotheken** zum Klassenpfad hinzu (unten beschrieben). Dadurch sollten die meisten bzw. alle Fehler gelöst werden.
 - Bei WebSphere Business Integration EJB-Projekten mit erweiterter Nachrichtenübertragung (CMM) oder Container Managed Persistence over Anything (CMP/A) müssen die IBM EJB-Deskriptordateien mit der Erweiterung Jar migriert werden, nachdem das 5.1 Projekt in den 6.0 Arbeitsbereich importiert wurde. Weitere Informationen finden Sie in "Migration von WebSphere Business Integration EJB-Projekten".
 - Fixieren Sie den Klassenpfad für jedes in den Arbeitsbereich importierte nicht-Serviceprojekt. Um die Bibliotheken von JRE und WebSphere Process Server zum Klassenpfad hinzuzufügen, klicken Sie mit der rechten Maustaste auf das importierte Projekt und wählen Sie **'Eigenschaften'**. Gehen Sie zum Eintrag **'Java-Erstellungspfad'** und wählen Sie die Registerkarte **'Bibliotheken'**. Gehen Sie danach folgendermaßen vor:
 - 1) Wählen Sie **'Bibliothek hinzufügen' → 'JRE-Systembibliothek' → 'Alternative JRE - WPS-Server V6.0 JRE' → 'Fertig stellen'**.
 - 2) Wählen Sie daraufhin **'Bibliothek hinzufügen' → 'Ziel des WPS-Servers' → 'Klassenpfad des WPS-Servers konfigurieren' → 'Fertig stellen'**.
6. Um bestmögliche Ergebnisse zu erzielen, wählen Sie vor Ausführung des Migrationsassistenten den Menüpunkt **'Projekt'** und stellen Sie sicher, dass **'Automatisch erstellen'** NICHT ausgewählt ist. WebSphere Integration Developer unterscheidet sich von WebSphere Studio Application Developer Integration Edition insofern, als dass die Service-Implementierungsoptionen zur Entwicklungszeit angegeben werden. Bei der Erstellung des Projekts wird der Implementierungscode automatisch in der generierten EJB und den Web-Projekten aktualisiert, daher gibt es die Option zum manuellen **Implementierungscode erstellen** nicht mehr.

7. Um die .bpel-Dateien innerhalb eines Serviceprojekts vollständig zu migrieren, stellen Sie sicher, dass alle .wsdl- und .xsd-Dateien, auf die von den .bpel-Dateien verwiesen wird, in einem Geschäftsintegrationsprojekt im neuen Arbeitsbereich aufgelöst werden können:
 - Wenn sich die .wsdl- und/oder .xsd-Dateien in demselben Serviceprojekt wie die .bpel-Datei befinden, ist keine weitere Aktion erforderlich.
 - Wenn sich die .wsdl- und/oder .xsd-Dateien in einem anderen Serviceprojekt als dem befinden, das Sie migrieren, müssen die 5.1-Artefakte mithilfe der WebSphere Studio Application Developer Integration Edition vor der Migration reorganisiert werden. Der Grund hierfür ist, dass Geschäftsintegrations-Modulprojekte keine Artefakte gemeinsam nutzen können. Im Folgenden sind zwei Optionen für die Reorganisation der 5.1-Artefakte ausgeführt:
 - Erstellen Sie ein neues Java-Projekt in WebSphere Studio Application Developer Integration Edition, in dem alle allgemeinen Artefakte enthalten sind. Legen Sie alle .wsdl- und .xsd-Dateien, die von mehr als einem Serviceprojekt gemeinsam genutzt werden, in diesem neuen Java-Projekt ab. Fügen Sie eine Abhängigkeit von diesem neuen Java-Projekt zu allen Serviceprojekten hinzu, die diese allgemeinen Artefakte verwenden. Erstellen Sie ein neues Geschäftsintegrations-Bibliotheksprojekt in WebSphere Integration Developer mit demselben Namen wie das 5.1 gemeinsam genutzte Java-Projekt, bevor Sie eins der Serviceprojekte migrieren. Kopieren Sie die alten .wsdl- und .xsd-Dateien aus dem gemeinsam genutzten Java-Projekt von 5.1 in diesen neuen BI-Bibliotheksprojektordner. Dies muss vor der Migration der BPEL-Serviceprojekte geschehen.
 - Eine andere Option besteht darin, eine lokale Kopie dieser gemeinsam genutzten .wsdl- und .xsd-Artefakte in jedem Serviceprojekt zu behalten, so dass es zu keinen Abhängigkeiten zwischen Serviceprojekten kommt.
 - Wenn sich die .wsdl- und/oder .xsd-Dateien in einem anderen Projekttyp befinden (normalerweise andere Java-Projekte), erstellen Sie ein Geschäftsintegrations-Bibliotheksprojekt mit demselben Namen wie der des 5.1-Projekts. Sie sollten außerdem den neuen Klassenpfad des neuen Bibliotheksprojekts einrichten und die Einträge des Java-Projekts aus 5.1 hinzufügen, sofern welche vorhanden sind. Dieser Projekttyp ist nützlich für das Speichern von gemeinsam genutzten Artefakten.

Nun können Sie mit dem Migrationsprozess beginnen.

Serviceprojekte mit dem WebSphere Integration Developer-Migrationsassistenten migrieren

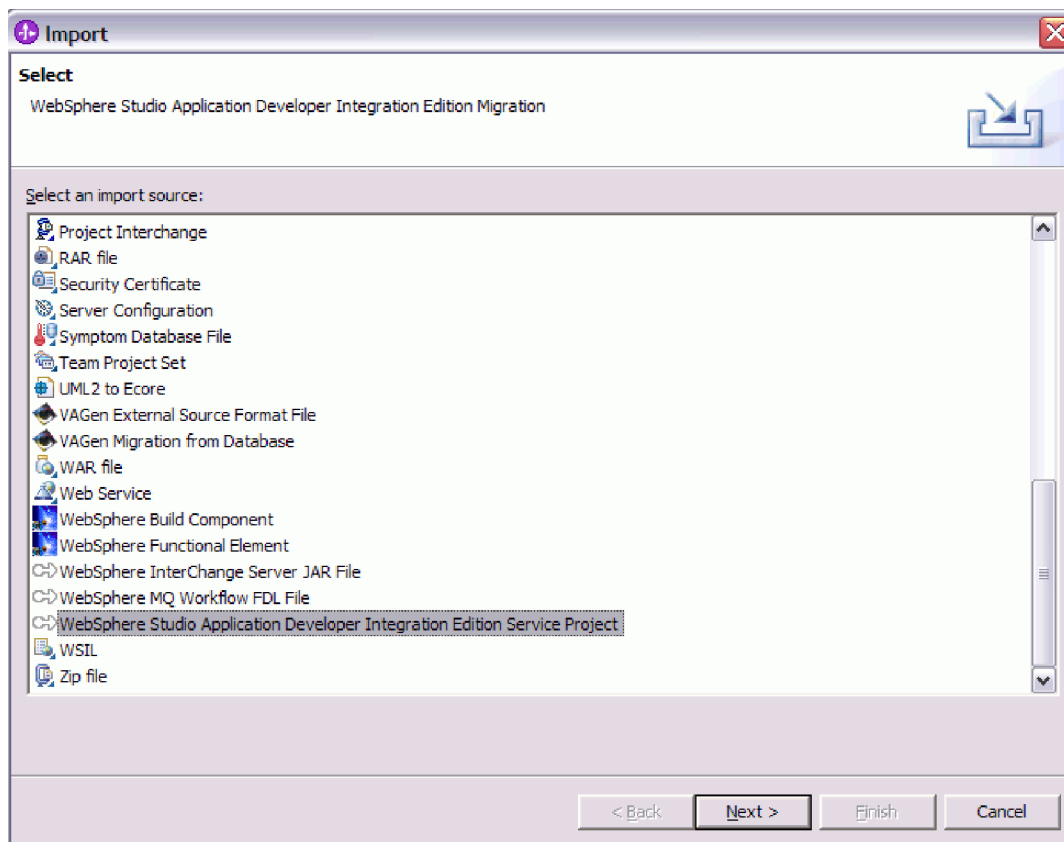
Der WebSphere Integration Developer-Migrationsassistent ermöglicht die Migration von Serviceprojekten.

Der Migrationsassistent führt die folgenden Schritte aus:

1. Erstellung eines neuen Geschäftsintegrationsmoduls (der Modulname wird von Ihnen definiert)
2. Migration der Serviceprojekt-Klassenpfadeinträge zum neuen Modul
3. Kopieren aller WebSphere Business Integration Server Foundation-Quellenartefakte aus dem ausgewählten Quellenprojekt in dieses Modul
4. Migration der BPEL-Erweiterungen zu WSDL-Dateien
5. Migration der Geschäftsprozesse (.bpel-Dateien) von der BPEL4WS-Version 1.1 zur neuen von WebSphere Process Server unterstützten Ebene, die auf BPEL4WS Version 1.1 mit wichtigen Funktionalitäten der zukünftigen WS-BPEL Version 2.0-Spezifikation aufbaut
6. Erstellung einer SCA-Komponente für jeden .bpel-Prozess
7. Erstellung einer Überwachungsdatei (.mon) für jeden BPEL-Prozess, um das Standardüberwachungsverhalten von WebSphere Studio Application Developer Integration Edition (wenn nötig) beizubehalten

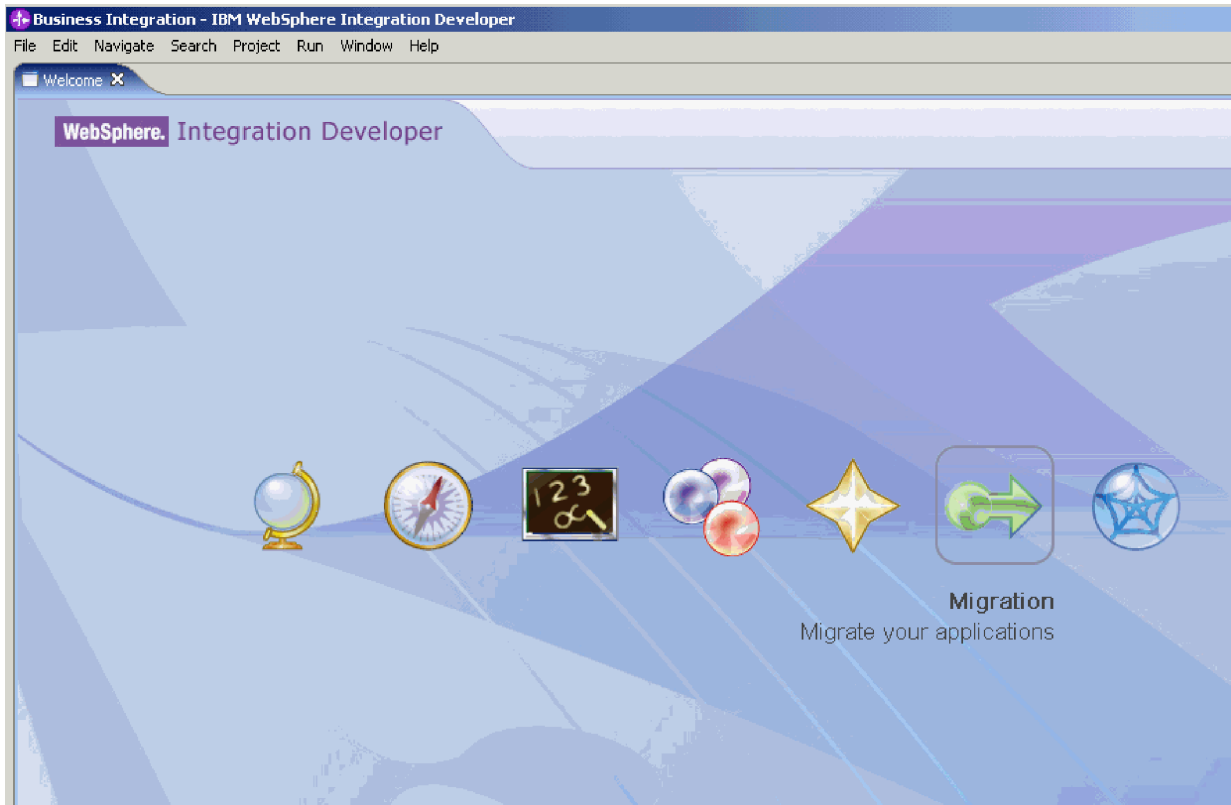
Befolgen Sie diese Schritte, um Serviceprojekte mit dem WebSphere Integration Developer-Migrationsassistenten zu migrieren:

1. Rufen Sie den Assistenten durch Auswahl von 'Datei' → 'Importieren' → 'WebSphere Studio Application Developer Integration Edition-Serviceprojekt' auf.



Sie können den Migrationsassistenten auf der Willkommenseite auch öffnen, indem Sie auf das

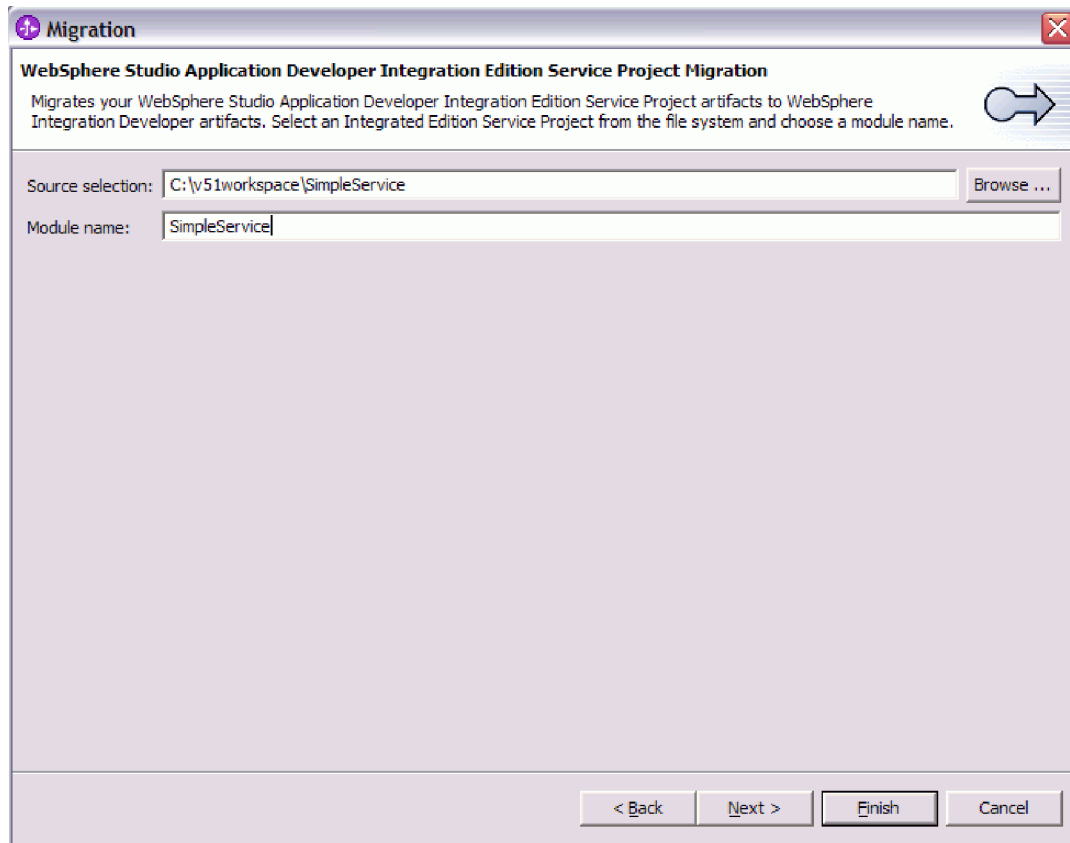
Symbol  klicken:



Auf der Seite 'Migration' wählen Sie die Option für die Migration eines Serviceprojekts aus Integration Edition 5.1:

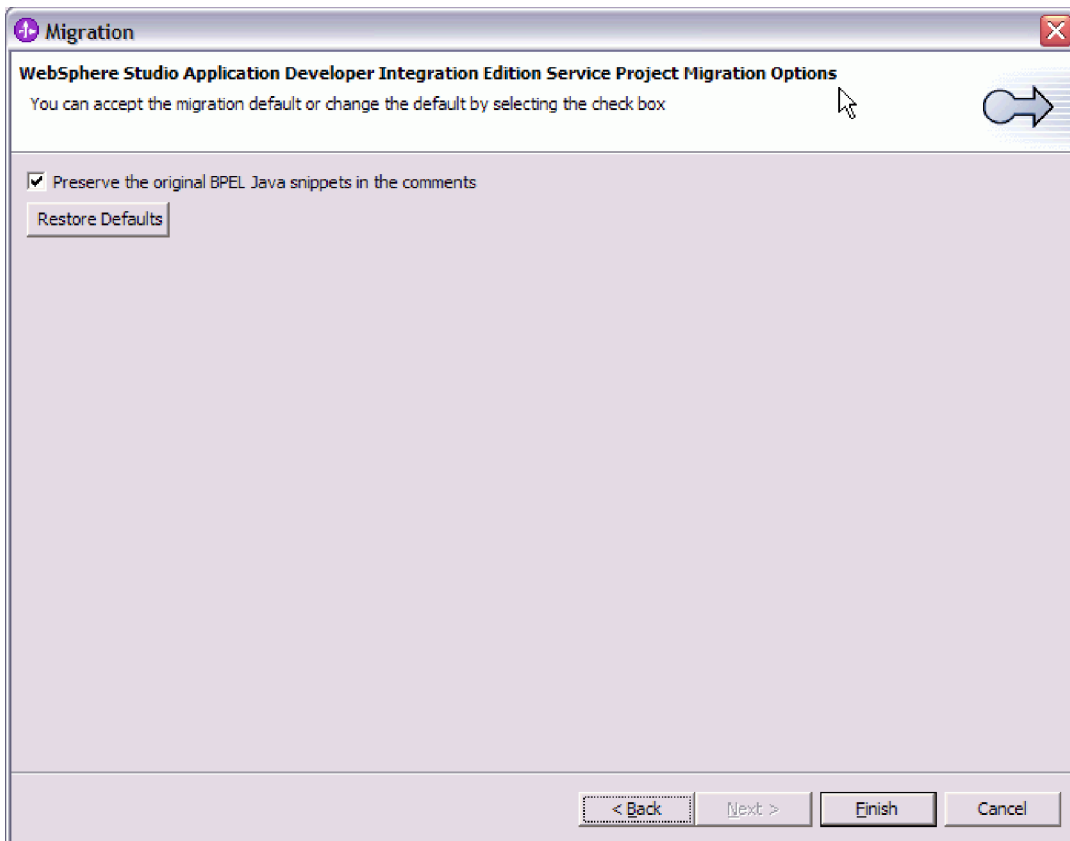


- Der Migrationsassistent wird geöffnet. Geben Sie den Pfad für die Quellenauswahl ein oder klicken Sie auf die Schaltfläche **'Durchsuchen'**, um sie zu suchen. Geben Sie außerdem den Modulnamen für die Position des zu migrierenden Serviceprojekts von WebSphere Studio Application Developer Integration Edition ein:



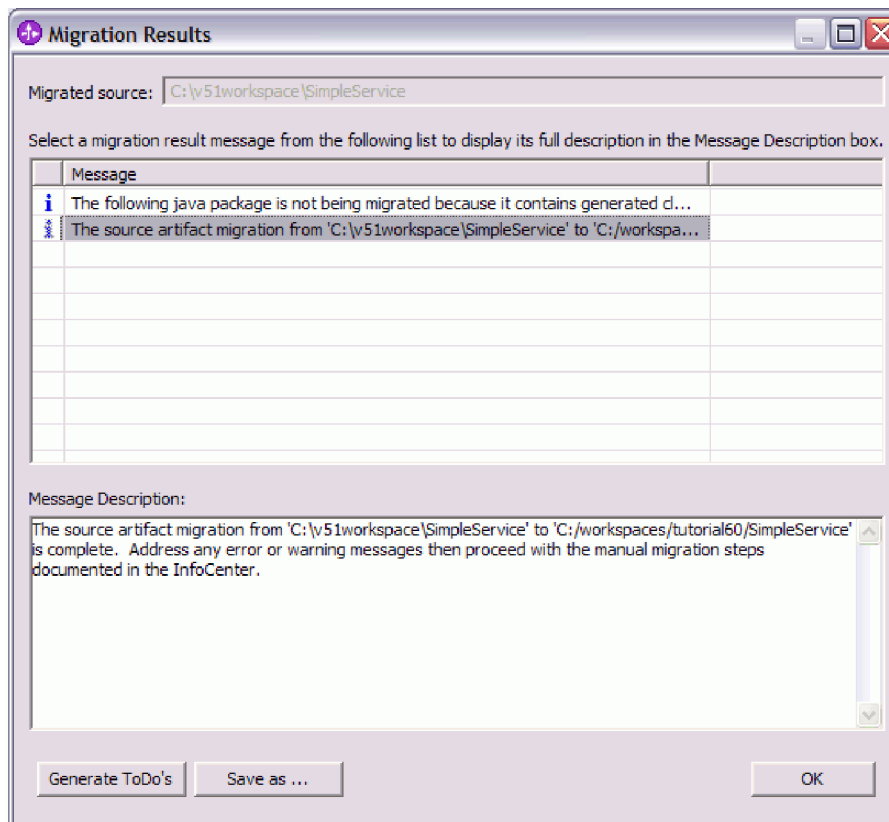
Hinweis: Es wird empfohlen, dass Sie den Namen des Serviceprojekts als Modulnamen auswählen, denn wenn sich andere Projekte im WebSphere Studio Application Developer Integration Edition-Arbeitsbereich befinden, die von diesem Projekt abhängen, müssen Sie die Klassenpfade des untergeordneten Projekts nach deren Import in WebSphere Integration Developer nicht aktualisieren.

3. Wählen Sie in den Migrationsoptionen 'Ursprüngliche BPEL Java-Snippets beibehalten' im Markierungsfeld für Kommentare:



Klicken Sie auf 'Fertig stellen'.

4. Wenn der Migrationsprozess abgeschlossen ist, wird das Fenster mit den Migrationsergebnissen geöffnet:



Es wird automatisch eine diese Migrationsnachrichten enthaltende Protokolldatei im .metadata-Ordner des 6.0 Arbeitsbereichs generiert. Die Protokolldatei erhält den Namen ".log".

Wenn der Migrationsassistent abgeschlossen ist, bauen Sie das erstellte Geschäftsintegrationsmodul auf und versuchen Sie, alle Aufbaufehler zu beheben. Überprüfen Sie alle migrierten .bpel-Dateien: stellen Sie sicher, dass diese vollständig migriert wurden und im WebSphere Integration Developer BPEL Editor geöffnet werden können. Es gibt bestimmte BPEL Java Snippets, die nicht automatisch migriert werden können. Wenn Sie Fehler in den BPEL JavaSnippets feststellen, finden Sie Informationen zur Behebung dieser Fehler in "Migration zum SCA-Programmiermodell". Wenn Sie den Migrationsassistenten außerdem für die Migration eines Serviceprojekts zu einem BI-Modul verwendet haben, öffnen Sie den Modulabhängigkeitseditor, um sicherzustellen, dass die Abhängigkeiten korrekt eingestellt sind. Dazu wechseln Sie in die Geschäftsintegrationsperspektive und doppelklicken Sie auf das BI-Modulprojekt. Hier können Sie Abhängigkeiten von Projekten für Geschäftsintegrationsbibliotheken, Java-Projekten und J2EE-Projekten hinzufügen.

Manuelle Migration der Anwendung

Wenn der Migrationsassistent die Artefakte erfolgreich in das neue Geschäftsintegrationsmodul migriert hat, müssen die Artefakte zur Erstellung einer Anwendung, die sich nach dem SCA-Modell richtet, miteinander verbunden werden.

1. Öffnen Sie WebSphere Integration Developer und wechseln Sie zur Geschäftsintegrationsperspektive. Nun sollten die Module angezeigt werden, die vom Migrationsassistenten erstellt wurden (ein Modul für jedes migrierte Serviceprojekt). Beim ersten Artefakt, das unter dem Modulprojekt aufgelistet ist, handelt es sich um die Assemblydatei des Moduls (es verfügt über den gleichen Namen wie das Modul).
2. Doppelklicken Sie auf die Assemblydatei, um sie im Assembly-Editor zu öffnen, in dem SCA-Komponenten erstellt und miteinander verbunden werden können, um eine ähnliche Funktionalität wie die der Version 5.1-Anwendung zu erhalten. Wenn im WebSphere Studio Application Developer Integra-

tion Edition-Serviceprojekt BPEL-Prozesse vorhanden waren, wurden Standard SCA-Komponenten vom Migrationsassistenten für jeden dieser Prozesse erstellt, die sich daraufhin im Assembly-Editor befinden.

3. Wählen Sie eine Komponente und gehen Sie zur Sicht 'Eigenschaften', in der die Beschreibung, Informationen und Implementierungseigenschaften angezeigt werden und bearbeitet werden können.

Die folgenden Informationen enthalten weitere Hinweise dazu, wie die Anwendung manuell mit den in WebSphere Integration Developer zur Verfügung gestellten Tools neu verbunden werden kann:

SCA-Komponenten und SCA-Importe für die Anwendungsservices zur Neuvernetzung erstellen:

Alle Projekte erfordern eine Neuvernetzung nach der Migration, um die Services wie in 5.1 neu zu verbinden. Beispielsweise müssen alle migrierten Geschäftsprozesse neu mit ihren Geschäftspartnern verbunden werden. Es muss eine SCA-Komponente oder ein Import für alle anderen Servicetypen erstellt werden. Für WebSphere Studio Application Developer Integration Edition-Serviceprojekte, die mit projektexternen Systemen oder Entitäten interagieren, kann ein SCA-Import erstellt werden, damit das migrierte Projekt auf diese Entitäten als Services gemäß dem SCA-Modell zugreifen kann.

Für WebSphere Studio Application Developer Integration Edition-Serviceprojekte, die mit Entitäten innerhalb des Projekts interagieren (beispielsweise ein Geschäftsprozess, Umsetzungsservice oder eine Java-Klasse) kann ein SCA-Import erstellt werden, damit das migrierte Projekt auf diese Entitäten als Services gemäß dem SCA-Modell zugreifen kann.

Die folgenden Abschnitte enthalten detaillierte Informationen über den zu erstellenden SCA-Import oder die SCA-Komponenten basierend auf dem Servicetyp, der migriert werden muss:

Migration eines Java-Service:

Sie können einen Java-Service zu einer SCA Java-Komponente migrieren.

Wenn das WebSphere Studio Application Developer Integration Edition-Serviceprojekt von anderen Java-Projekten abhing, kopieren Sie die vorhandenen Projekte in das neue Arbeitsbereichsverzeichnis und importieren Sie sie in den WebSphere Integration Developer mithilfe des Assistenten unter **'Datei' → 'Importieren' → 'Vorhandenes Projekt in Arbeitsbereich'**.

In WebSphere Studio Application Developer Integration Edition standen beim Generieren eines neuen Java Service aus einer vorhandenen Java-Klasse die folgenden Optionen zur Verfügung:

- XSD-Schemata für komplexe Datentypen erstellen:
 - Innerhalb der Schnittstellen-WSDL-Datei
 - Als neue Datei für jeden Datentyp
- Unterstützung der Funktionalität zur Fehlerbehandlung:
 - Fehler generieren
 - Keinen Fehler generieren
- Sonstige Informationen über den zu generierenden Service, wie beispielsweise Binding und Servicenamen

Es gibt zahlreiche neue Komponenten in 6.0, die neue Funktionen wie Datenzuordnung, Schnittstellenmediation, Geschäftsstatusmaschinen, Selektoren, Geschäftsregeln und vieles mehr bieten. Zunächst sollten Sie bestimmen, ob einer dieser neuen Komponententypen die benutzerdefinierte Java-Komponente ersetzen kann. Wenn dies nicht möglich ist, folgen Sie dem unten beschriebenen Migrationspfad.

Importieren Sie das Serviceprojekt mit dem Migrationsassistenten. Dies resultiert in der Erstellung eines Geschäftsintegrationsmoduls mit den WSDL-Nachrichten, PortTypes, Bindings und Services, die in WebSphere Studio Application Developer Integration Edition generiert werden.

Erweitern Sie in der Geschäftsintegrationsperspektive das Modul, um seinen Inhalt anzuzeigen. Öffnen Sie den Assembly-Editor durch Doppelklicken auf das erste Element unterhalb des Modulprojekts (es verfügt über denselben Namen wie das Projekt).

Es stehen die folgenden Optionen zur Verfügung:

Vor- und Nachteile für jede der Neuvernetzungsoptionen des Java-Service:

Es gibt Vor- und Nachteile für jede der Neuvernetzungsoptionen des Java-Service.

Die folgende Liste beschreibt beide Optionen mit den jeweiligen Vor- und Nachteilen:

- Die erste Option bietet wahrscheinlich eine bessere Leistung zur Laufzeit, da der Aufruf eines Web-Services langsamer ist als der Aufruf einer Java-Komponente.
- Die erste Option kann Kontext weitergeben, während ein Web-Serviceaufruf Kontext nicht auf die gleiche Weise weitergibt.
- Die zweite Option beinhaltet nicht die Erstellung eines benutzerdefinierten Codes.
- Die zweite Option ist unter Umständen für einige Java-Schnittstellendefinitionen nicht möglich, da die Erzeugung eines Java-Services Einschränkungen unterliegt. Weitere Informationen finden Sie in der Dokumentation zu Rational Application Developer unter: <http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.doc/ref/rlimit.html>
- Die zweite Option kann zu einer Schnittstellenänderung führen und damit einer Änderung des SCA-Verbrauchers.
- Die zweite Option setzt voraus, dass ein Server von WebSphere Process Server 6.0 installiert und für die Arbeit mit WebSphere Integration Developer konfiguriert ist. Um die installierten Laufzeiten anzuzeigen, die für die Arbeit mit WebSphere Integration Developer konfiguriert wurden, gehen Sie zu **'Fenster' → 'Benutzervorgaben' → 'Server' → 'Installierte Laufzeiten'**, wählen Sie den Eintrag **'WebSphere Process Server v6.0'** (falls dieser vorhanden ist) und stellen Sie sicher, dass er auf die Position verweist, an der das Produkt installiert ist. Stellen Sie sicher, dass dieser Eintrag markiert ist, wenn der Server vorhanden ist, und nicht markiert, wenn dieser Server nicht installiert ist. Sie können auch auf die Schaltfläche **'Hinzufügen...'** klicken, wenn Sie einen anderen Server hinzufügen möchten.
- Wenn die Java-Komponente in WebSphere Studio Application Developer Integration Edition mithilfe der Top-down-Methode erstellt wurde, wobei das Java-Gerüst von einer WSDL erstellt wurde, sind die Parameter in diese und aus dieser Java-Klasse wahrscheinlich eine Unterklasse von `WSIFFormatPartImpl`. Wenn dies der Fall ist, wählen Sie Option 1 für die Erzeugung eines neuen Java-Gerüsts im SCA-Stil aus den ursprünglichen WSDL/XSDs oder Option 2 für die Erzeugung eines neuen generischen Java-Gerüsts (nicht abhängig von den WSIF- oder DataObject-APIs) aus der ursprünglichen WSDL-Schnittstelle.

Die benutzerdefinierte Java-Komponente erstellen: Option 1:

Die empfohlene Migrationstechnik ist die Verwendung des WebSphere Integration Developer Java-Komponententyps, mit dem Sie den Java-Service als eine SCA-Komponente darstellen können. Während der Migration muss benutzerdefinierter Java-Code geschrieben werden, um zwischen dem SCA Java-Schnittstellenstil und dem vorhandenen Schnittstellenstil der Java-Komponente zu unterscheiden.

So erstellen Sie die benutzerdefinierte Java-Komponente:

1. Erweitern Sie im Modulprojekt die Option **'Schnittstellen'** und wählen Sie die WSDL-Schnittstelle, die für diese Java-Klasse in WebSphere Studio Application Developer Integration generiert wurde.
2. Ziehen und übergeben Sie diese Schnittstelle an den Assembly-Editor. Es wird ein Dialog angezeigt, der Sie nach dem zu erstellenden Komponententyp fragt. Wählen Sie **'Komponente mit keinem Implementierungstyp'** und klicken Sie auf **OK**.
3. Eine generische Komponente wird im Assemblydiagramm angezeigt. Markieren Sie sie und gehen Sie zur Ansicht **'Eigenschaften'**.

4. Sie können den Namen und Anzeigenamen der Komponente in einen beschreibenden Namen in der Registerkarte **'Beschreibung'** ändern.
5. Auf der Registerkarte **'Details'** können Sie sehen, dass diese Komponente über eine Schnittstelle verfügt, nämlich die, die Sie zum Assembly-Editor gezogen und an diesen übergeben haben.
6. Stellen Sie sicher, dass sich die Java-Klasse, auf die Sie versuchen, zuzugreifen, im Klassenpfad des Serviceprojekts befindet, wenn sie nicht im Projekt selbst enthalten ist.
7. Klicken Sie mit der rechten Maustaste auf das Modulprojekt und wählen Sie **'Editor für Abhängigkeiten öffnen...'**. Stellen Sie sicher, dass das Projekt, das die alte Java-Klasse enthält, im Abschnitt **'Java'** aufgelistet ist. Wenn dies nicht der Fall ist, fügen Sie sie durch Klicken auf die Schaltfläche **'Hinzufügen...'** hinzu.
8. Klicken Sie im Assembly-Editor mit der rechten Maustaste auf die Komponente, die Sie soeben erstellt haben, und wählen Sie **'Implementierung generieren...'** → **'Java'**. Wählen Sie daraufhin das Paket, in dem die Java-Implementierung generiert wird. Dadurch wird ein Java-Gerüstservice erstellt, der sich an die WSDL-Schnittstelle gemäß dem SCA-Programmiermodell hält, in dem komplexe Typen durch ein `commonj.sdo.DataObject`-Objekt dargestellt werden und einfache Typen durch ihre funktional entsprechenden Java-Objekte.

Die folgenden Code-Beispiele zeigen:

1. Relevante Definitionen aus der 5.1 WSDL-Schnittstelle
2. Die WebSphere Studio Application Developer Integration Edition 5.1 Java-Methoden, die der WSDL entsprechen
3. Die WebSphere Integration Developer 6.0 Java-Methoden für die gleiche WSDL

Der folgende Code zeigt die relevanten Definitionen aus der 5.1 WSDL-Schnittstelle:

```
<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>
```

Der folgende Code zeigt die WebSphere Studio Application Developer Integration Edition 5.1 Java-Methoden, die der WSDL entsprechen:

```
public StockInfo getStockInfo(String symbol)
{
return new StockInfo();
}
```

```

public void setStockPrice(String symbol, float newPrice)
{
    // set some things
}

```

Der folgende Code zeigt die WebSphere Integration Developer 6.0 Java-Methoden für die gleiche WSDL:

```

public DataObject getStockInfo(String aString) {
    //TODO Needs to be implemented.
    return null;
}

public void setStockPrice(String symbol, Float newPrice) {
    //TODO Needs to be implemented.
}

```

Sie müssen nun Code dort eingeben, wo die “//TODO”-Tags in der generierten Java-Implementierungsklasse angezeigt werden. Es gibt zwei Optionen:

1. Verschieben Sie die Logik von der ursprünglichen Java-Klasse in diese Klasse und passen Sie sie für die Verwendung von DataObject an.
 - Hierbei handelt es sich um die empfohlene Option, wenn Sie die Top-down-Methode in WebSphere Studio Application Developer Integration Edition ausgewählt haben und die Java-Komponente mit DataObject-Parametern arbeiten soll. Diese Nacharbeit ist notwendig, da die aus WSDL-Definitionen in WebSphere Studio Application Developer Integration Edition generierten Java-Klassen über WSIF-Abhängigkeiten verfügen, die wenn möglich eliminiert werden sollten.
2. Erstellen Sie eine private Instanz der alten Java-Klasse in dieser generierten Java-Klasse und schreiben Sie Code, um Folgendes zu tun:
 - a. Konvertieren aller Parameter der generierten Java-Implementierungsklasse in Parameter, die die alte Java-Klasse erwartet
 - b. Aufrufen der privaten Instanz der alten Java-Klasse mit den konvertierten Parametern
 - c. Konvertieren des Rückgabewerts der alten Java-Klasse in den Rückgabewerttyp, der von der generierten Java-Implementierungsmethode deklariert wird
 - d. Diese Option wird für Auslastungsszenarios empfohlen, wobei die WSIF-Service-Proxys von neuen Java-Komponenten im Stil 6.0 verbraucht werden.

Wenn Sie eine der obenstehenden Optionen abgeschlossen haben, müssen Sie den Java-Service neu verbinden. Es sollten keine “Verweise” vorhanden sein, daher müssen Sie die Schnittstelle der Java-Komponente neu verbinden:

- Wenn dieser Service durch einen Geschäftsprozess in demselben Modul aufgerufen wird, stellen Sie eine Verbindung vom entsprechenden Geschäftsprozessverweis zur Schnittstelle dieser Java-Komponente her.
- Wenn dieser Service von einem Geschäftsprozess in einem anderen Modul aufgerufen wird, erstellen Sie einen **Export mit SCA-Binding** und ziehen und über geben Sie diesem Export vom anderen Modul zum Assembly-Editor dieses Moduls, um den entsprechenden **Import mit SCA-Binding** zu erstellen. Verbinden Sie den entsprechenden Geschäftsprozessverweis mit diesem Import.
- Wenn dieser Service in WebSphere Studio Application Developer Integration Edition veröffentlicht wurde, um ihn extern zugänglich zu machen, finden Sie Informationen zur Neuveröffentlichung dieses Services im Abschnitt “SCA-Exporte für den Zugriff auf den migrierten Service erstellen”.

Einen Java-Web-Service erstellen: Option 2:

Eine alternative zu berücksichtigende Option sind die Rational Application Developer Web-Servicetools, mit denen Sie einen Web-Service um eine Java-Klasse herum erstellen können.

Anmerkung: Weitere Informationen finden Sie auf der folgenden Site, bevor Sie eine Migration mit der folgenden Methode versuchen: <http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.was.creation.ui.doc/tasks/twsbeanw.html>

Anmerkung: Diese Option erfordert die Konfiguration einer Web-Service-Laufzeit über WebSphere Integration Developer, bevor der Web-Service-Assistent aufgerufen wird.

Wenn Sie eine Bottom-up-Methode in WebSphere Studio Application Developer Integration Edition verwendet haben, um WSDL um die Java-Klasse herum zu generieren, führen Sie die folgenden Schritte durch:

1. Erstellen Sie ein neues Web-Projekt und kopieren Sie die Java-Klasse, um die Sie den Service erstellen möchten, in den Java-Quellenordner dieses Web-Projekts.
2. Klicken Sie mit der rechten Maustaste auf das Unternehmensanwendungsprojekt, das den Container für die Java-Klasse darstellt, um die herum Sie einen Service erstellen.
3. Wählen Sie **'Eigenschaften'**, gehen Sie zu den Eigenschaften **'Server'** und stellen Sie sicher, dass die **'Ziellaufzeit'** auf **'WebSphere Process Server v6.0'** und **'Standardserver'** auf den installierten **'WebSphere Process Server v6.0'** eingestellt ist.
4. Starten Sie den Testserver, implementieren Sie diese Anwendung für den Server und stellen Sie sicher, dass er erfolgreich startet.
5. Klicken Sie als nächstes mit der rechten Maustaste auf die Java-Klasse, um die herum Sie einen Service erstellen möchten, und wählen Sie **'Web-Services' → 'Web-Service erstellen'**.
6. Wählen Sie für **'Web-Service-Typ'** die Option **'Java-Bean-Web-Service'** und heben Sie die Markierung für **'Web-Service im Webprojekt starten'** auf, es sei denn, Sie möchten den Web-Service sofort implementieren. Optional haben Sie auch die Möglichkeit, ein Client-Proxy zu generieren. Klicken Sie auf **'Weiter'**.
7. Die Java-Klasse, auf die Sie mit der rechten Maustaste geklickt haben, wird angezeigt, klicken Sie auf **'Weiter'**.
8. Sie müssen nun Ihre Service-Implementierungsoptionen konfigurieren. Klicken Sie auf die Schaltfläche **'Bearbeiten...'**. Als Servertyp wählen Sie **WPS-Server V6.0** und als Web-Service-Laufzeit wählen Sie **IBM WebSphere** und J2EE Version **1.4**. Wenn Sie dadurch keine gültige Kombination auswählen können, finden Sie weitere Informationen zur Migration von J2EE-Projekten zur V1.4 Ebene im Abschnitt "Vorbereitung für die Migration". Klicken Sie auf **OK**.
9. Geben Sie den Namen des Webprojekts für das Serviceprojekt ein. Wählen Sie außerdem das entsprechende EAR-Projekt. Klicken Sie auf **'Weiter'**. Beachten Sie, dass Sie möglicherweise mehrere Minuten warten müssen.
10. Wählen Sie im Java-Bean-Identitätsfenster des Web-Services die WSDL-Datei, die die die WSDL-Definitionen enthalten wird. Wählen Sie die Methoden, die für den Web-Service zugänglich gemacht werden sollen, und wählen Sie den entsprechenden Stil/die Verschlüsselung (Dokument/Literal, RPC/Literal oder RPC/verschlüsselt). Wählen Sie die Option für die **angepasste Zuordnung für Paket zu Namensbereich**, und wählen Sie einen Namensbereich, der für die migrierte Java-Klasse für alle Java-Pakete eindeutig ist, die von der Schnittstelle dieser Java-Klasse verwendet werden. (Der Standardnamensbereich ist für den Paketnamen eindeutig, was zu Konflikten führen kann, falls Sie einen weiteren Web-Service erstellen, der dieselben Java-Klassen verwendet). Geben Sie ggfs. weitere Parameter an.
11. Klicken Sie auf **'Weiter'**. Klicken Sie in der Anzeige für die **'Zuordnung von Web-Service-Paket zu Namensbereich'** auf die Schaltfläche **'Hinzufügen'**, und geben Sie in der erstellten Zeile den Namen des Pakets der Java-Bean ein. Fügen Sie dann den angepassten Namensbereich hinzu, der diese Java-Klasse eindeutig kennzeichnet. Fügen Sie weitere Zuordnungen für alle Java-Pakete hinzu, die von der Java-Bean-Schnittstelle verwendet werden.
12. Klicken Sie auf **'Weiter'**. Beachten Sie, dass Sie möglicherweise mehrere Minuten warten müssen.
13. Klicken Sie auf **'Fertig stellen'**. Wenn Sie die Arbeit mit dem Assistenten abgeschlossen haben, kopieren Sie die den Java-Service beschreibende generierte WSDL-Datei in das Geschäftsintegrations-Modulprojekt, wenn es sich beim Serviceprojekt um einen Verbraucher des Java-Services handelte. Sie befindet sich im generierten Router-Web-Projekt im Ordner WebContent/WEB-INF/wsdl. Aktualisieren/erstellen Sie das Geschäftsintegrations-Modulprojekt erneut.

14. Wechseln Sie zur Geschäftsintegrationsperspektive und erweitern Sie erst das Modul und daraufhin die logische Kategorie **Web-Service-Ports**.
15. Wählen Sie den in den zuvor durchgeführten Schritten erstellten Port, ziehen und übergeben Sie ihn an den Assembly-Editor und markieren Sie ihn, um einen **Import mit Web-Service-Binding** zu erstellen. Wenn Sie dazu aufgefordert werden, wählen Sie die WSDL-Schnittstelle der Java-Klasse. Die SCA-Komponente, die die Java-Klasse in 5.1 verbraucht hat, kann nun mit diesem Import verbunden werden, um die Migrationsschritte für die manuelle Neuvernetzung abzuschließen.

Beachten Sie, dass die Schnittstelle sich leicht von der 5.1 Schnittstelle unterscheiden kann, und Sie möglicherweise eine Schnittstellenmediationskomponente zwischen dem 5.1 Verbraucher und dem neuen Import einfügen müssen. Klicken Sie dazu auf das Tool **'Verbinden'** im Assembly-Editor und verbinden Sie die SCA-Quellenkomponente mit diesem neuen **Import mit Web-Service-Binding**. Da die Schnittstellen sich unterscheiden, wird die folgende Meldung angezeigt: **Der Quell- und Zielknoten haben keine übereinstimmenden Schnittstellen**. Wählen Sie **'Zwischen dem Quell- und dem Zielknoten eine Schnittstellenzuordnung erstellen'**. Doppelklicken Sie auf die Zuordnungskomponente, die im Assembly-Editor erstellt wurde. Dadurch wird der Zuordnungseditor geöffnet. Anweisungen zur Erstellung einer Schnittstellenzuordnung finden Sie im Information Center.

Wenn Sie eine Top-down-Methode in WebSphere Studio Application Developer Integration Edition mit der Erstellung von Java-Klassen aus einer WSDL-Definition gewählt hatten, führen Sie die folgenden Schritte durch:

1. Erstellen Sie ein neues Webprojekt und kopieren Sie die WSDL-Datei, aus der Sie das Java-Gerüst generieren möchten, in den Quellenordner dieses Webprojekts.
2. Klicken Sie mit der rechten Maustaste auf die WSDL-Datei, die den PortType enthält, aus dem Sie das Java-Gerüst generieren möchten, und wählen Sie **'Web Services' → 'Java-Bean-Gerüst generieren'**.
3. Wählen Sie den Web-Servicetyp **Skeleton-Java-Bean-Web-Service** und beenden Sie den Assistenten.

Nach Beenden des Assistenten sollten Sie über Java-Klassen verfügen, die die Serviceschnittstelle implementieren und nicht von WSIF APIs abhängen.

Migration eines EJB-Service:

Sie können einen EJB-Service zu einem SCA-Import mit Binding für Session-Bean ohne Status migrieren.

Wenn das WebSphere Studio Application Developer Integration Edition-Serviceprojekt von einer anderen EJB, einem EJB-Client oder Java-Projekt abhängig war, importieren Sie diese vorhandenen Projekte mithilfe des Assistenten unter **'Datei' → 'Importieren' → 'Vorhandenes Projekt in Arbeitsbereich'**. Dies war normalerweise der Fall, wenn auf eine EJB von einem Serviceprojekt verwiesen wurde. Wenn WSDL- oder XSD-Dateien, auf die vom Serviceprojekt verwiesen wird, in einem anderen Projekttyp vorhanden sind, erstellen Sie eine neue Geschäftsintegrationsbibliothek mit demselben Namen wie der des alten nicht-Service-Projekts, und kopieren Sie alle Artefakte in die Bibliothek.

Importieren Sie das Serviceprojekt mit dem Migrationsassistenten. Dies resultiert in der Erstellung eines Geschäftsintegrationsmoduls mit den WSDL-Nachrichten, PortTypes, Bindings und Services, die in WebSphere Studio Application Developer Integration Edition generiert werden.

Erweitern Sie in der Geschäftsintegrationsperspektive das Modul, um seinen Inhalt anzuzeigen. Öffnen Sie den Assembly-Editor durch Doppelklicken auf das erste Element unterhalb des Modulprojekts (es verfügt über denselben Namen wie das Projekt).

Es stehen die folgenden Optionen zur Verfügung:

Vor- und Nachteile für jede der Neuvernetzungsoptionen des EJB-Service:

Es gibt Vor- und Nachteile für jede der Neuvernetzungsoptionen des EJB-Service.

Die folgende Liste beschreibt beide Optionen mit den jeweiligen Vor- und Nachteilen:

- Die erste Option bietet wahrscheinlich eine bessere Leistung zur Laufzeit, da der Aufruf eines Web-Services langsamer ist als der Aufruf eines EJB.
- Die erste Option kann Kontext weitergeben, während ein Web-Serviceaufruf Kontext nicht auf die gleiche Weise weitergibt.
- Die zweite Option beinhaltet nicht die Erstellung eines benutzerdefinierten Codes.
- Die zweite Option ist unter Umständen für einige EJB-Schnittstellendefinitionen nicht möglich, da die Erzeugung eines EJB-Services Einschränkungen unterliegt. Weitere Informationen finden Sie in der Dokumentation zu Rational Application Developer unter folgender Adresse: <http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.doc/ref/rlimit.html>
- Die zweite Option kann zu einer Schnittstellenänderung führen und damit einer Änderung des SCA-Verbrauchers.
- Die zweite Option setzt voraus, dass ein Server von WebSphere Process Server 6.0 installiert und für die Arbeit mit WebSphere Integration Developer konfiguriert ist. Um die installierten Laufzeiten anzuzeigen, die für die Arbeit mit WebSphere Integration Developer konfiguriert wurden, gehen Sie zu **'Fenster' → 'Benutzervorgaben' → 'Server' → 'Installierte Laufzeiten'**, wählen Sie den Eintrag **'WebSphere Process Server v6.0'** (falls dieser vorhanden ist) und stellen Sie sicher, dass er auf die Position verweist, an der das Produkt installiert ist. Stellen Sie sicher, dass dieser Eintrag markiert ist, wenn der Server vorhanden ist, und nicht markiert, wenn dieser Server nicht installiert ist. Sie können auch auf die Schaltfläche **'Hinzufügen...'** klicken, wenn Sie einen anderen Server hinzufügen möchten.
- Wenn die Java-Komponente in WebSphere Studio Application Developer Integration Edition mithilfe der Top-down-Methode erstellt wurde, wobei das EJB-Gerüst von einer WSDL erstellt wurde, sind die Parameter in diese und aus dieser Java-Klasse wahrscheinlich eine Unterklasse von `WSIFFormatParImpl`. Wenn dies der Fall ist, wählen Sie Option 2 für die Erzeugung eines neuen generischen EJB-Gerüsts (nicht abhängig von den WSIF oder DataObject APIs) aus der ursprünglichen WSDL-Schnittstelle.

Die benutzerdefinierte EJB-Komponente erstellen: Option 1:

Die empfohlene Migrationstechnik ist die Verwendung des WebSphere Integration Developer-Imports mit einem Session-Bindingtyp ohne Status, mit dem Sie eine Session-EJB ohne Status als SCA-Komponente aufrufen können. Während der Migration muss benutzerdefinierter Java-Code geschrieben werden, um zwischen dem SCA Java-Schnittstellenstil und dem vorhandenen EJB-Schnittstellenstil zu konvertieren.

So erstellen Sie die benutzerdefinierte EJB-Komponente:

1. Erweitern Sie im Modulprojekt die Option **'Schnittstellen'** und wählen Sie die WSDL-Schnittstelle, die für diese EJB in WebSphere Studio Application Developer Integration generiert wurde.
2. Ziehen und übergeben Sie diese Schnittstelle an den Assembly-Editor. Es wird ein Dialog angezeigt, der Sie nach dem zu erstellenden Komponententyp fragt. Wählen Sie **'Komponente mit keinem Implementierungstyp'** und klicken Sie auf **OK**.
3. Eine generische Komponente wird im Assemblydiagramm angezeigt. Markieren Sie sie und gehen Sie zur Ansicht **'Eigenschaften'**.
4. Sie können den Namen und Anzeigenamen der Komponente in einen beschreibenden Namen in der Registerkarte **'Beschreibung'** ändern. Wählen Sie einen Namen wie Ihren EJB-Namen, fügen Sie jedoch eine Erweiterung wie "JavaMed" hinzu, da es sich hierbei um eine Java-Komponente handeln wird, die zwischen der WSDL -Schnittstelle, die für EJB in WebSphere Studio Application Developer Integration generiert wurde, und der Java-Schnittstelle des EJB vermittelt.
5. Auf der Registerkarte **'Details'** können Sie sehen, dass diese Komponente über eine Schnittstelle verfügt, nämlich die, die Sie zum Assembly-Editor gezogen und an diesen übergeben haben.
6. Klicken Sie im Assembly-Editor mit der rechten Maustaste auf die Komponente, die Sie soeben erstellt haben, und wählen Sie **'Implementierung generieren...'** → **'Java'** Wählen Sie daraufhin das Paket, in dem die Java-Implementierung generiert wird. Dadurch wird ein Java-Gerüstservice erstellt, der sich

an die WSDL-Schnittstelle gemäß dem SCA-Programmiermodell hält, in dem komplexe Typen durch ein `commonj.sdo.DataObject`-Objekt dargestellt werden und einfache Typen durch ihre funktional entsprechenden Java-Objekte.

Die folgenden Code-Beispiele zeigen:

1. Relevante Definitionen aus der 5.1 WSDL-Schnittstelle
2. Die WebSphere Studio Application Developer Integration Edition 5.1 Java-Methoden, die der WSDL entsprechen
3. Die WebSphere Integration Developer 6.0 Java-Methoden für die gleiche WSDL

Der folgenden Code zeigt die relevanten Definitionen aus der 5.1 WSDL-Schnittstelle:

```
<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>
```

Der folgende Code zeigt die WebSphere Studio Application Developer Integration Edition 5.1 Java-Methoden, die der WSDL entsprechen:

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

Der folgende Code zeigt die WebSphere Integration Developer 6.0 Java-Methoden für die gleiche WSDL:

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

Sie müssen nun echten Code dort eingeben, wo die “//TODO”-Tags in der generierten Java-Implementierungsklasse angezeigt werden. Zunächst müssen Sie einen Verweis von dieser Java-Komponente zur aktuellen EJB erstellen, so dass sie auf die EJB gemäß dem SCA-Programmiermodell zugreifen kann:

1. Lassen Sie den Assembly-Editor geöffnet und wechseln Sie in die J2EE-Perspektive. Suchen Sie das EJB-Projekt, das die EJB enthält, für die Sie einen Service erstellen.
2. Erweitern Sie das Element **Implementierungsdeskriptor: <project-name>** und suchen Sie die EJB. Ziehen und übergeben Sie sie an den Assembly-Editor. Wenn Sie über Projektabhängigkeiten gewarnt werden, die aktualisiert werden müssen, wählen Sie das Kontrollkästchen **Den Editor für Modulabhängigkeiten öffnen...** und klicken Sie auf **OK**.
3. Stellen Sie sicher, dass das EJB-Projekt im J2EE-Abschnitt aufgelistet ist, und wenn dies nicht der Fall ist, fügen Sie es durch Klicken auf die Schaltfläche **Hinzufügen...** hinzu.
4. Speichern Sie die Modulabhängigkeiten und schließen Sie den Editor. Sie werden sehen, dass ein neuer Import im Assembly-Editor erstellt wurde. Markieren Sie ihn und gehen Sie zur Ansicht **Eigenschaften** in der Registerkarte **Beschreibung**, um den Namen und Anzeigenamen des Imports in einen aussagekräftigeren Namen zu ändern. Auf der Registerkarte **Binding** können Sie sehen, dass der Importtyp normalerweise auf **Binding für Session-Bean ohne Status** gesetzt ist und der JNDI-Name der EJB bereits ordnungsgemäß eingerichtet ist.
5. Wählen Sie das Verbindungstool aus der Palette im Assembly-Editor.
6. Klicken Sie auf die Java-Komponente und lassen Sie die Maustaste los.
7. Klicken Sie als nächstes auf den EJB-Import und lassen Sie die Maustaste los.
8. Wenn Sie gefragt werden **Am Quellenknoten wird ein übereinstimmender Verweis erstellt. Möchten Sie fortfahren?** Klicken Sie auf **OK**. Dadurch wird eine Verbindung zwischen den zwei Komponenten hergestellt.
9. Wählen Sie die Java-Komponente im Assembly-Editor und erweitern Sie in der Ansicht **Eigenschaften** in der Registerkarte **Details** die Option **Verweise**, und wählen Sie den Verweis zur EJB, die gerade erstellt wurde. Sie können den Namen des Verweises aktualisieren, wenn der generierte Name nicht sehr aussagekräftig oder geeignet ist. Notieren Sie den Namen dieses Verweises für die zukünftige Verwendung.
10. Speichern Sie das Assembly-Diagramm.

Sie müssen das SCA-Programmiermodell verwenden, um die EJB aus der generierten Java-Klasse aufzurufen. Öffnen Sie die generierte Java-Klasse und befolgen Sie diese Schritte, um den Code für den Aufruf des EJB-Services zu schreiben. Für die generierte Java-Implementierungsklasse:

1. Erstellen Sie eine private Variable (deren Typ der Ihrer remote EJB-Schnittstelle entspricht):

```
private YourEJBInterface ejbService = null;
```
2. Wenn komplexe Typen in Ihrer EJB-Schnittstelle vorhanden sind, erstellen Sie außerdem eine private Variable für die BOFactory:

```
private BOFactory boFactory = (BOFactory)
    ServiceManager.INSTANCE.locateService("com/ibm/websphere/bo
    /BOFactory");
```
3. Verwenden Sie die SCA APIs im Konstruktor der Java-Implementierungsklasse, um den EJB-Verweis aufzulösen (denken Sie daran, den Namen des EJB-Verweises, den Sie ein paar Schritte zuvor notiert haben, einzugeben), und stellen Sie die private Variable gemäß diesem Verweis ein:

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

Für jedes “//TODO” in der generierten Java-Implementierungsklasse:

1. Konvertieren Sie alle Parameter in die Parametertypen, die die EJB erwartet.
2. Rufen Sie die entsprechende Methode im EJB-Verweis mithilfe des SCA-Programmiermodells auf, und senden Sie die konvertierten Parameter.

3. Konvertieren Sie den Rückgabewert der EJB in den Rückgabewerttyp, der von der generierten Java-Implementierungsmethode deklariert wird

```
/**
 * 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();
    }
}
```

Einen EJB-Web-Service erstellen: Option 2:

Eine alternative zu berücksichtigende Option sind die Rational Application Developer Web-Servicetools, mit denen Sie einen Web-Service um eine EJB herum erstellen können.

Anmerkung: Weitere Informationen finden Sie auf der folgenden Site, bevor Sie eine Migration mit der folgenden Methode versuchen: <http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.was.creation.ejb.ui.doc/tasks/twsejbw.html>

Anmerkung: Diese Option erfordert die Konfiguration einer Web-Service-Laufzeit über WebSphere Integration Developer, bevor der Web-Service-Assistent aufgerufen wird.

Befolgen Sie die folgenden Schritte, um einen Web-Service um eine EJB herum zu erstellen:

1. Klicken Sie mit der rechten Maustaste auf das Unternehmensanwendungsprojekt, das den Container für die EJB darstellt, um die herum Sie einen Service erstellen.
2. Wählen Sie **'Eigenschaften'**, gehen Sie zu den Eigenschaften **'Server'** und stellen Sie sicher, dass die **'Ziellaufzeit'** auf **WebSphere Process Server v6.0** und **'Standardserver'** auf den installierten **WebSphere Process Server v6.0** eingestellt ist.
3. Starten Sie den Testserver, implementieren Sie diese Anwendung für den Server und stellen Sie sicher, dass er erfolgreich startet.
4. Erweitern Sie in der J2EE-Perspektive das **EJB-Projekt** in der Ansicht **'Projekt-Explorer'**. Erweitern Sie den **Implementierungsdeskriptor** und die Kategorie **Session-Beans**. Wählen Sie die Bean, um die der Web-Service herum erstellt werden soll.
5. Klicken Sie mit der rechten Maustaste und wählen Sie **'Web-Services' → 'Web-Services erstellen'**.

6. Wählen Sie für **'Web-Service-Typ'** die Option **'EJB-Web-Service'** und heben Sie die Markierung für **'Web-Service im Webprojekt starten'** auf, es sei denn, Sie möchten den Web-Service sofort implementieren. Klicken Sie auf **'Weiter'**.
7. Stellen Sie sicher, dass die EJB, auf die Sie mit der rechten Maustaste geklickt haben, hier ausgewählt ist, und klicken Sie auf **'Weiter'**.
8. Sie müssen nun Ihre Service-Implementierungsoptionen konfigurieren. Klicken Sie auf die Schaltfläche **'Bearbeiten...'**. Als Servertyp wählen Sie **WPS-Server V6.0** und als Web-Service-Laufzeit wählen Sie **IBM WebSphere** und J2EE Version **1.4**. Wenn Sie dadurch keine gültige Kombination auswählen können, finden Sie weitere Informationen zur Migration von J2EE-Projekten zur V1.4 Ebene im Abschnitt "Vorbereitung für die Migration". Klicken Sie auf **OK**.
9. Für das Serviceprojekt geben Sie den Namen des EJB-Projekts ein, das die EJB enthält. Wählen Sie außerdem das entsprechende EAR-Projekt. Klicken Sie auf **'Weiter'**. Beachten Sie, dass Sie möglicherweise mehrere Minuten warten müssen.
10. Wählen Sie im Fenster für die Web-Service-EJB-Konfiguration das zu verwendende Router-Projekt (wählen Sie den Namen des zu erstellenden Router-Web-Projekts, und dieses Projekt wird zu der gleichen Unternehmensanwendung wie die ursprüngliche EJB hinzugefügt). Wählen Sie den gewünschten Transport (**SOAP über HTTP** oder **SOAP über JMS**). Klicken Sie auf **'Weiter'**.
11. Wählen Sie die WSDL-Datei, die die WSDL-Definitionen enthalten wird. Wählen Sie die Methoden, die für den Web-Service zugänglich gemacht werden sollen, und wählen Sie den entsprechenden Stil/die Verschlüsselung (Dokument/Literal, RPC/Literal oder RPC/verschlüsselt). Wählen Sie die Option für die **angepasste Zuordnung für Paket zu Namensbereich** und einen Namensbereich, der für die migrierte EJB für alle Java-Pakete eindeutig ist, die von der EJB verwendet werden. (Der Standardnamensbereich ist für den Paketnamen eindeutig, was zu Konflikten führen kann, falls Sie einen weiteren Web-Service erstellen, der dieselben Java-Klassen verwendet). Geben Sie ggfs. weitere Parameter an. Es gibt für jede Kombination aus Stil/Verschlüsselung Einschränkungen. Weitere Informationen finden Sie in den Einschränkungen unter: <http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.doc/ref/rlimit.html>
12. Klicken Sie auf **'Weiter'**. Klicken Sie in der Anzeige für die **Zuordnung von Web-Service-Paket zu Namensbereich** auf die Schaltfläche **'Hinzufügen'**, und geben Sie in der erstellten Zeile den Namen des Pakets der EJB ein. Fügen Sie dann den angepassten Namensbereich hinzu, der diese EJB eindeutig kennzeichnet. Fügen Sie weiter Zuordnungen für alle Java-Pakete hinzu, die von der EJB-Schnittstelle verwendet werden.
13. Klicken Sie auf **'Weiter'**. Beachten Sie, dass Sie möglicherweise mehrere Minuten warten müssen.
14. Klicken Sie auf **'Fertig stellen'**. Wenn Sie die Arbeit mit dem Assistenten abgeschlossen haben, kopieren Sie die den EJB-Service beschreibende generierte WSDL-Datei in das Geschäftsintegrations-Modulprojekt, wenn es sich beim Serviceprojekt um einen Verbraucher des EJB-Services handelte. Sie befindet sich im generierten Router-Web-Projekt im Ordner WebContent/WEB-INF/wsdl. Aktualisieren/erstellen Sie das Geschäftsintegrations-Modulprojekt erneut.
15. Wechseln Sie zur Geschäftsintegrationsperspektive und erweitern Sie erst das migrierte Modul und daraufhin die logische Kategorie **Web-Service-Ports**.
16. Wählen Sie den in den zuvor durchgeführten Schritten generierten Port, ziehen und übergeben Sie ihn an den Assembly-Editor und markieren Sie ihn, um einen **Import mit Web-Service-Binding** zu erstellen. Wenn Sie dazu aufgefordert werden, wählen Sie die WSDL-Schnittstelle der EJB. Die SCA-Komponente, die die EJB in 5.1 verbraucht hat, kann nun mit diesem Import verbunden werden, um die Migrationsschritte für die manuelle Neuvernetzung abzuschließen.

Wenn Sie eine Top-down-Methode in WebSphere Studio Application Developer Integration Edition mit der Erstellung eines EJB-Gerüsts aus einer WSDL-Definition gewählt hatten, führen Sie die folgenden Schritte durch:

1. Erstellen Sie ein neues Webprojekt und kopieren Sie die WSDL-Datei, aus der Sie das EJB-Gerüst generieren möchten, in den Quellenordner dieses Web-Projekts.
2. Klicken Sie mit der rechten Maustaste auf die WSDL-Datei, die den PortType enthält, aus dem Sie das EJB-Gerüst generieren möchten, und wählen Sie **'Web Services' → 'Java-Bean-Gerüst generieren'**.

3. Wählen Sie den Web-Servicetyp **Skeleton-EJB-Web-Service** und beenden Sie den Assistenten.

Nach Beenden des Assistenten sollten Sie über eine EJB verfügen, die die Serviceschnittstelle implementiert und nicht von WSIF APIs abhängt.

Beachten Sie, dass die Schnittstelle sich leicht von der 5.1 Schnittstelle unterscheiden kann, und Sie möglicherweise eine Schnittstellenmediationskomponente zwischen dem 5.1 Verbraucher und dem neuen Import einfügen müssen. Klicken Sie dazu auf das Tool **'Verbinden'** im Assembly-Editor und verbinden Sie die SCA-Quellenkomponente mit diesem neuen **Import mit Web-Service-Binding**. Da die Schnittstellen sich unterscheiden, wird die folgende Meldung angezeigt: **Der Quell- und Zielknoten haben keine übereinstimmenden Schnittstellen**. Wählen Sie **'Zwischen dem Quell- und dem Zielknoten eine Schnittstellenzuordnung erstellen'**. Doppelklicken Sie auf die Zuordnungskomponente, die im Assembly-Editor erstellt wurde. Dadurch wird der Zuordnungseitor geöffnet. Anweisungen zur Erstellung einer Schnittstellenzuordnung finden Sie im Information Center.

Wenn Sie diesen Vorgang abgeschlossen haben, müssen Sie den EJB-Service neu verbinden. Es sollten keine "Verweise" vorhanden sein, daher müssen Sie die Schnittstelle der Java-Komponente neu verbinden:

- Wenn dieser Service durch einen Geschäftsprozess in demselben Modul aufgerufen wird, stellen Sie eine Verbindung vom entsprechenden Geschäftsprozessverweis zu dieser EJB-Komponente her.
- Wenn dieser Service von einem Geschäftsprozess in einem anderen Modul aufgerufen wird, erstellen Sie einen **Export mit SCA-Binding** und ziehen und übergeben Sie diesem Export vom anderen Modul zum Assembly-Editor dieses Moduls, um den entsprechenden **Import mit SCA-Binding** zu erstellen. Verbinden Sie den entsprechenden Geschäftsprozessverweis mit diesem Import.
- Wenn dieser Service in WebSphere Studio Application Developer Integration Edition veröffentlicht wurde, um ihn extern zugänglich zu machen, finden Sie Informationen zur Neuveröffentlichung dieses Services im Abschnitt "Eingehende nicht-BPEL-Servicemigration".

Migration eines Geschäftsprozesses zum Geschäftsprozess-Serviceaufruf:

Dieses Szenario ist auf einen Geschäftsprozess anwendbar, der einen anderen Geschäftsprozess aufruft, wobei der zweite Geschäftsprozess mithilfe eines WSIF-Prozessbindings aufgerufen wird. Dieser Abschnitt beschreibt die Migration von BPEL zu einem BPEL-Serviceaufruf mithilfe einer Verbindung oder eines Imports/Exports mit SCA-Binding.

Für die Migration eines Prozessbinding-Serviceprojekts (BPEL) führen Sie die folgenden Schritte durch:

1. Erweitern Sie in der Geschäftsintegrationsperspektive das Modul, um seinen Inhalt anzuzeigen. Öffnen Sie den Assembly-Editor durch Doppelklicken auf das erste Element unterhalb des Modulprojekts (es verfügt über denselben Namen wie das Projekt).
2. Es gibt mehrere Szenarien, in denen ein BPEL-Prozess einen anderen BPEL-Prozess aufrufen kann. Suchen Sie unten nach dem Szenario, das Ihre Anwendung betrifft:
 - Falls sich die aufgerufene BPEL im gleichen Modul befindet, stellen Sie eine Verbindung vom entsprechenden Verweis der ersten BPEL-Komponente zur entsprechenden Schnittstelle der BPEL-Zielkomponente her.
 - Falls sich die aufgerufene BPEL in einem anderen Modul befindet (wobei es sich bei dem anderen Modul um ein migriertes Serviceprojekt handelt), gehen Sie folgendermaßen vor:
 - a. Erstellen Sie einen **Export mit SCA-Binding** für den zweiten Geschäftsprozess in seinem Modul-Assembly-Diagramm.
 - b. Erweitern Sie das Assembly-Symbol des zweiten Moduls im Navigator der Ansicht 'Geschäftsintegration'. Der soeben von Ihnen erstellte Export sollte nun angezeigt werden.
 - c. Ziehen und übergeben (Drag und Drop) Sie den Export von der Ansicht 'Geschäftsintegration' unterhalb des zweiten Moduls in den geöffneten Assembly-Editor des ersten Moduls. Dadurch wird ein Import mit SCA-Binding im ersten Modul erstellt. Wenn dieser Service in WebSphere

Studio Application Developer Integration Edition veröffentlicht wurde, um ihn extern zugänglich zu machen, finden Sie Informationen zur im Abschnitt "SCA-Exporte für den Zugriff auf den migrierten Service erstellen".

- d. Verbinden Sie den entsprechenden Verweis des ersten Geschäftsprozesses mit dem Import, den Sie gerade in diesem Modul erstellt haben.
- e. Speichern Sie das Assembly-Diagramm.
- So erreichen Sie ein spätes Binding beim Aufruf des zweiten Geschäftsprozesses:
 - a. Stellen Sie keine Verbindung zum Verweis der ersten Geschäftsprozesskomponente her. Öffnen Sie den ersten Prozess im BPEL-Editor und wählen Sie im Abschnitt für die **Verweispartner** denjenigen Partner, der dem zweiten BPEL-Prozess entspricht, um den Aufruf mit einem späten Binding zu erreichen.
 - b. Geben Sie in der Ansicht 'Eigenschaften' auf der Registerkarte '**Beschreibung**' den Namen des zweiten Geschäftsprozesses im Feld für die **Prozess-Schablone** ein.
 - c. Speichern Sie den Geschäftsprozess. Der Aufruf mit spätem Binding ist hiermit eingerichtet.

Migration eines Web-Services (SOAP/JMS):

Sie können einen Web-Service (SOAP/JMS) zu einem SCA-Import mit Web-Service-Binding migrieren.

Führen Sie die folgenden Schritte durch, um ein SOAP/JMS-Serviceprojekt für eine ausgehende Service-migration zu migrieren:

1. Zunächst müssen Sie das Serviceprojekt mithilfe des Migrationsassistenten importieren. Dies resultiert in der Erstellung eines Geschäftsintegrationsmoduls mit den WSDL-Nachrichten, PortTypes, Bindings und Services, die in WebSphere Studio Application Developer Integration Edition generiert werden. Beachten Sie, dass wenn es sich bei dem IBM Web-Service (SOAP/JMS), der von dieser Anwendung aufgerufen wird, außerdem um einen WebSphere Studio Application Developer Integration Edition Web-Service handelt, der migriert wird, es möglicherweise Aktualisierungen für diesen Web-Service während der Migration gab. Wenn dies der Fall ist, verwenden Sie die migrierten WSDL-Dateien dieses Web-Services an dieser Stelle.
2. Erweitern Sie in der Geschäftsintegrationsperspektive das Modul, sodass Sie seinen Inhalt anzeigen können. Öffnen Sie den Assembly-Editor durch Doppelklicken auf das erste Element unterhalb des Modulprojekts (es verfügt über denselben Namen wie das Projekt).
3. Als nächstes fügen Sie einen Import hinzu, der der Anwendung ermöglicht, mit dem IBM Web Service (via SOAP/JMS) gemäß dem SCA-Programmiermodell zu interagieren. Stellen Sie sicher, dass die WSDL-Schnittstelle, Binding und Servicedefinitionen im migrierten Modul oder in einer Bibliothek, von der das Modul abhängt, vorhanden sind.
4. Erweitern Sie in der Geschäftsintegrationsperspektive das migrierte Modul und öffnen Sie das Assembly-Diagramm im Assembly-Editor.
5. Erweitern Sie die logische Kategorie des Web-Service-Ports und ziehen und übergeben Sie den Port, der dem aufzurufenden Service entspricht, an den Assembly-Editor.
6. Wählen Sie die Erstellung eines **Imports mit Web-Service-Binding**.
7. Nach Erstellung des Imports wählen Sie ihn im Assembly-Editor und gehen Sie zur Ansicht 'Eigenschaften'. In der Registerkarte 'Binding' wird der Port und Service angezeigt, an die der Import gebunden ist.
8. Speichern Sie das Assembly-Diagramm.

Wenn Sie diesen Vorgang abgeschlossen haben, müssen Sie den Service neu verbinden:

- Wenn dieser Service durch einen Geschäftsprozess in demselben Modul aufgerufen wird, stellen Sie eine Verbindung vom entsprechenden Geschäftsprozessverweis zu diesem Import her.
- Wenn dieser Service von einem Geschäftsprozess in einem anderen Modul aufgerufen wird, erstellen Sie einen **Export mit SCA-Binding** und ziehen und übergeben Sie diesem Export vom anderen Modul

zum Assembly-Editor dieses Moduls, um den entsprechenden **Import mit SCA-Binding** zu erstellen. Verbinden Sie den entsprechenden Geschäftsprozessverweis mit diesem Import.

- Speichern Sie das Assembly-Diagramm.

Migration eines Web-Services (SOAP/HTTP):

Sie können einen Web-Service (SOAP/HTTP) zu einem SCA-Import mit Web-Service-Binding migrieren.

Führen Sie die folgenden Schritte durch, um ein SOAP/HTTP-Serviceprojekt für eine ausgehende Servicemigration zu migrieren:

1. Zunächst müssen Sie das Serviceprojekt mithilfe des Migrationsassistenten importieren. Dies resultiert in der Erstellung eines Geschäftsintegrationsmoduls mit den WSDL-Nachrichten, PortTypes, Bindings und Services, die in WebSphere Studio Application Developer Integration Edition generiert werden. Beachten Sie, dass wenn es sich bei dem IBM Web-Service (SOAP/HTTP), der von dieser Anwendung aufgerufen wird, außerdem um einen WebSphere Studio Application Developer Integration Edition Web-Service handelt, der migriert wird, es möglicherweise Aktualisierungen für diesen Web-Service während der Migration gab. Wenn dies der Fall ist, verwenden Sie die migrierten WSDL-Dateien dieses Web-Services an dieser Stelle.
2. Erweitern Sie in der Geschäftsintegrationsperspektive das Modul, sodass Sie seinen Inhalt anzeigen können. Öffnen Sie den Assembly-Editor durch Doppelklicken auf das erste Element unterhalb des Modulprojekts (es verfügt über denselben Namen wie das Projekt).
3. Als nächstes fügen Sie einen Import hinzu, der der Anwendung ermöglicht, mit dem IBM Web Service (via SOAP/HTTP) gemäß dem SCA-Programmiermodell zu interagieren. Stellen Sie sicher, dass die WSDL-Schnittstelle, Binding und Servicedefinitionen im migrierten Modul oder in einer Bibliothek, von der das Modul abhängt, vorhanden sind.
4. Erweitern Sie in der Geschäftsintegrationsperspektive das migrierte Modul und öffnen Sie das Assembly-Diagramm im Assembly-Editor.
5. Erweitern Sie die logische Kategorie des Web-Service-Ports und ziehen und übergeben Sie den Port, der dem aufzurufenden Service entspricht, an den Assembly-Editor.
6. Wählen Sie die Erstellung eines **Imports mit Web-Service-Binding**.
7. Nach Erstellung des Imports wählen Sie ihn im Assembly-Editor und gehen Sie zur Ansicht 'Eigenschaften'. In der Registerkarte 'Binding' wird der Port und Service angezeigt, an die der Import gebunden ist.
8. Speichern Sie das Assembly-Diagramm.

Wenn Sie diesen Vorgang abgeschlossen haben, müssen Sie den Service neu verbinden:

- Wenn dieser Service durch einen Geschäftsprozess in demselben Modul aufgerufen wird, stellen Sie eine Verbindung vom entsprechenden Geschäftsprozessverweis zu diesem Import her.
- Wenn dieser Service von einem Geschäftsprozess in einem anderen Modul aufgerufen wird, erstellen Sie einen **Export mit SCA-Binding** und ziehen und übergeben Sie diesem Export vom anderen Modul zum Assembly-Editor dieses Moduls, um den entsprechenden **Import mit SCA-Binding** zu erstellen. Verbinden Sie den entsprechenden Geschäftsprozessverweis mit diesem Import.
- Speichern Sie das Assembly-Diagramm.

Migration eines JMS-Services:

Sie können einen JMS-Service zu einem SCA-Import mit JMS-Binding migrieren.

Anmerkung: Wenn die JMS-Nachricht an einen Adapter von WebSphere Business Integration gesendet wird, finden Sie weitere Informationen im Abschnitt "Interaktionen mit WebSphere Business Integration-Adaptern migrieren".

Führen Sie die folgenden Schritte durch, um ein JMS-Serviceprojekt für eine ausgehende Servicemigration zu migrieren:

1. Zunächst müssen Sie das Serviceprojekt mithilfe des Migrationsassistenten importieren. Dies resultiert in der Erstellung eines Geschäftsintegrationsmoduls mit den WSDL-Nachrichten, PortTypes, Bindings und Services, die in WebSphere Studio Application Developer Integration Edition generiert werden.
2. Erweitern Sie in der Geschäftsintegrationsperspektive das Modul, sodass Sie seinen Inhalt anzeigen können. Öffnen Sie den Assembly-Editor durch Doppelklicken auf das erste Element unterhalb des Modulprojekts (es verfügt über denselben Namen wie das Projekt).
3. Als nächstes fügen Sie einen Import hinzu, der der Anwendung ermöglicht, mit einer JMS-Warteschlange gemäß dem SCA-Programmiermodell zu interagieren.
4. Erweitern Sie das migrierte Modulprojekt im Assembly-Editor, erweitern Sie die Kategorie **'Schnittstellen'** und suchen Sie den WSDL PortType, der den von der Anwendung aufgerufenen Web-Service beschreibt. Ziehen und übergeben Sie sie an den Assembly-Editor.
5. Ein Dialog namens **'Erstellung von Komponenten'** ermöglicht Ihnen die Auswahl des zu erstellenden Komponententyps. Wählen Sie **'Ohne Binding importieren'**.
6. Sie werden sehen, dass ein neuer Import im Assembly-Editor erstellt wurde. Wenn Sie diesen markieren und zur Ansicht **'Eigenschaften'** gehen, können Sie den Namen und Anzeigenamen des Imports in der Registerkarte **'Beschreibung'** in einen aussagekräftigeren Namen ändern.
7. Details zum migrierten JMS-Service finden Sie in den WSDL-Bindingdateien und -Servicedateien aus 5.1. Diese Details können Sie verwenden, um die Details für den "Import mit JMS-Binding" in 6.0 anzugeben. Suchen Sie im 5.1-Serviceprojekt nach den WSDL-Dateien für 5.1-JMS-Binding und -Service (diese Dateien heißen normalerweise **'*JMSBinding.wsdl'** und **'*JMSService.wsdl'**). Untersuchen Sie die dort erfassten Binding- und Serviceinformationen. Anhand des Bindings können Sie ermitteln, ob Text- oder Objektnachrichten verwendet wurden und ob angepasste Datenformatbindings zum Einsatz kamen. In diesem Fall ist es sinnvoll, auch für den "Import mit JMS-Binding" von 6.0 ein angepasstes Datenbinding zu schreiben. Anhand des Services können Sie die Ausgangskontextfactory, den Namen der JNDI-Verbindungsfactory, den JNDI-Zieladressnamen und den Zieladressstil (Warteschlange) ermitteln.
8. Klicken Sie mit der rechten Maustaste auf den Import und wählen Sie **'Binding generieren'** und **JMS-Binding**. Sie werden aufgefordert, die folgenden Parameter einzugeben:

Wählen Sie die JMS-Nachrichtendomäne:

- Point-to-Point
- Publish-Subscribe
- Domain-Independent

Wählen Sie, wie Daten zwischen Geschäftsobjekt und JMS-Nachricht serialisiert werden:

- Text
- Objekt
- Benutzerdefiniert

Wenn 'Benutzerdefiniert' ausgewählt ist, gehen Sie folgendermaßen vor:

Geben Sie den vollständig qualifizierten Namen der `com.ibm.websphere.sca.jms.data.JMSDataBinding`-Implementierungsklasse an. Sie sollten ein benutzerdefiniertes Datenbinding angeben, wenn Ihre Anwendung JMS-Headereigenschaften festlegen muss, die normalerweise im JMS-Importbinding nicht verfügbar sind. In diesem Fall können Sie eine Klasse für das angepasste Datenbinding erstellen, die das JMS-Standarddatenbinding `"com.ibm.websphere.sca.jms.data.JMSDataBinding"` erweitert, und angepassten Code für den direkten Zugriff auf `'JMSMessage'` hinzufügen. Weitere Informationen finden Sie in den JMS-Beispielen des Abschnitts **"Bindings für Import- und Exportkomponenten erstellen und ändern"**, die Sie über den unten stehenden Link aufrufen können.

Konnektivität ankommender Daten verwendet die JMS-Standardfunktions-Selektorklasse:

`<ausgewählt>` oder `<abgewählt>`

9. Wählen Sie den soeben erstellten Export. Gehen Sie in der Ansicht 'Eigenschaften' zur Registerkarte 'Binding'. Sie können alle hier angezeigte Binding-Information manuell mit all den Werten eingeben, die Sie zuvor in WebSphere Studio Application Developer Integration Edition angegeben haben. Die Binding-Information, die Sie angeben können, lautet folgendermaßen:

- JMS-Importbinding (diese ist am wichtigsten)
- Verbindung
- Ressourcenadapter
- JMS-Ziele
- Methodenbindings

Wenn Sie diesen Vorgang abgeschlossen haben, müssen Sie den Service neu verbinden:

- Wenn dieser Service durch einen Geschäftsprozess in demselben Modul aufgerufen wird, stellen Sie eine Verbindung vom entsprechenden Geschäftsprozessverweis zu diesem Import her.
- Wenn dieser Service von einem Geschäftsprozess in einem anderen Modul aufgerufen wird, erstellen Sie einen **Export mit SCA-Binding** und ziehen und über geben Sie diesem Export vom anderen Modul zum Assembly-Editor dieses Moduls, um den entsprechenden **Import mit SCA-Binding** zu erstellen. Verbinden Sie den entsprechenden Geschäftsprozessverweis mit diesem Import.
- Speichern Sie das Assembly-Diagramm.

Migration eines J2C-IMS-Services:

Sie können einen J2C-IMS-Service zu einem SCA-Import mit EIS-Binding oder einem SCA-Import mit Web-Service-Binding migrieren.

Verwenden Sie keine der WebSphere Studio Application Developer Integration Edition-Artefakte, die für diesen IMS-Service generiert wurden. Sie müssen den Service mithilfe der Assistenten in WebSphere Integration Developer erneut erstellen und die Anwendung manuell neu verbinden.

Anmerkung: Schalten Sie automatische Erstellung ein oder erstellen Sie das Modul manuell.

Es stehen die folgenden Optionen zur Verfügung:

Anmerkung: Beachten Sie bei beiden Optionen, dass wenn ein BPEL-Service diesen IMS-Service aufruft, die BPEL leicht verändert werden muss, da die über den EIS-Service zugänglich gemachte Schnittstelle sich von der alten 5.1 Schnittstelle leicht unterscheidet. Um dies zu tun, öffnen Sie den BPEL-Editor, passen Sie den Partnerlink, der dem EIS-Service entspricht, an, und verwenden Sie die neue in den obenstehenden Schritten generierte Schnittstelle (WSDL-Datei). Nehmen Sie alle erforderlichen Änderungen an den BPEL-Aktivitäten für die neue WSDL-Schnittstelle des EIS-Services vor.

Vor- und Nachteile jeder der Neuvernetzungsoptionen des J2C-IMS-Service:

Es gibt Vor- und Nachteile für jede der Neuvernetzungsoptionen des J2C-IMS-Service.

Die folgende Liste beschreibt beide Optionen mit den jeweiligen Vor- und Nachteilen:

- Die erste Option verwendet die Standard-SCA-Komponenten zum Aufruf des IMS-Service.
- Die erste Option unterliegt den folgenden Einschränkungen:
 - Die SDO Version 1 Spezifikations-API bietet keinen Zugang zur COBOL- oder C-Bytefeldgruppe – dies betrifft Kunden, die mit IMS-Multisegmenten arbeiten.
 - Die SDO Version 1 Spezifikation für serielle Verarbeitung unterstützt keine COBOL-Neudefinitionen oder C-Datentypvariablen.
- Die zweite Option verwendet die Standard-JSR 109-Methode für die Verbindung mit dem IMS-Service. Diese Funktion ist verfügbar als Bestandteil von Rational Application Developer.

Erstellen Sie einen SCA-Import zum Aufrufen des IMS-Services: Option 1:

Sie können einen SCA-Import mit EIS-Binding erstellen, der DataObjects zur Speicherung der Nachrichten/Daten für die Kommunikation mit dem IMS-System verwendet.

Führen Sie die folgenden Schritte durch, um einen SCA-Import zum Aufrufen des IMS-Services zu erstellen:

1. Erstellen Sie ein neues Geschäftsintegrations-Modulprojekt, um diesen neuen IMS-Service zu speichern.
2. Klicken Sie zur Neuerstellung des EIS-Services auf **'Datei' → 'Neu' → 'Sonstige' → 'Geschäftsintegration' → 'Enterprise Service Discovery'**.
3. Mit diesem Assistenten können Sie einen Service aus einem EIS-System importieren. Er ähnelt dem WebSphere Studio Application Developer Integration Edition-Assistenten, der den WSIF-basierten EIS-Service in 5.1 erstellt hat. Sie können den neuen J2C IMS-Ressourcenadapter mit diesem Assistenten importieren. Durchsuchen Sie das Verzeichnis, in dem WebSphere Integration Developer installiert ist, und führen Sie eine Detailabfrage/-analyse unter **Ressourcenadapter → ims15 → imsic09102.rar** durch.

Anmerkung: Weitere Informationen zum Abschluss der Eigenschaften für Speichern und Fenster für Operationen finden Sie im Information Center. Beim Enterprise Service Discovery-Assistenten sind Sie während des Hinzufügens einer Operation in der Lage, Geschäftsobjekte für den Eingabe- oder Ausgabedatentyp der Operation zu erstellen. Dazu müssen Sie über die C- oder COBOL-Quellendatei verfügen, die Sie im WebSphere Studio Application Developer Integration Edition-Assistenten verwendet haben. Diese Dateien hätten in das alte Serviceprojekt kopiert werden sollen, so dass Sie dort auf die Quellendateien verweisen können. Sie können die Geschäftsobjekte auch mithilfe des separaten Assistenten unter **'Datei' → 'Neu' → 'Sonstige' → 'Geschäftsintegration' → 'Enterprise Data Discovery'** importieren.

4. Wenn Sie den Assistenten beendet haben, öffnen Sie die Geschäftsintegrationsperspektive und erweitern Sie das Modul, so dass Sie ihren Inhalt anzeigen können. In den Datentypen des Moduls sollten neue Geschäftsobjekte und in den Schnittstellen neue Schnittstellen aufgelistet sein.
5. Öffnen Sie den Assembly-Editor durch Doppelklicken auf das erste Element unterhalb des Modulprojekts (es verfügt über denselben Namen wie das Projekt). Im Erstellungsbereich sollte ein Import vorhanden sein, der über EIS-Binding verfügt und den soeben von Ihnen erstellten Service darstellt.

Anweisungen dazu, wie Sie diesen Service für Verbraucher zugänglich machen, finden Sie im Abschnitt "SCA-Exporte für den Zugriff auf den migrierten Service erstellen".

Einen Web-Service um den J2C-Service herum erstellen: Option 2:

Sie können einen J2C Web-Service erstellen, und wenn es sich beim Verbraucher des Services um eine SCA-Komponente handelt, können Sie den Service als einen IBM Web-Service (SOAP/HTTP oder SOAP/JMS) auslasten.

Führen Sie die folgenden Schritte durch, um einen Web-Service um den J2C-Service herum zu erstellen:

1. Erstellen Sie die J2C Java-Bean durch Klicken auf **'Datei' → 'Neu' → 'J2C' → 'J2C-JavaBean'**
2. Wählen Sie die 1.5 Version des **IMS-Connector für Java** und klicken Sie auf **'Weiter'**.
3. Markieren Sie **'Verwaltete Verbindung'** und geben Sie den JNDI Lookup-Namen ein. Klicken Sie auf **'Weiter'**.
4. Geben Sie das Projekt, das Paket und den Namen für die neue Java-Bean an. Die Bean setzt sich aus einer Schnittstelle und einer Implementierungsklasse zusammen. Klicken Sie auf **'Weiter'**.
5. Fügen Sie eine Java-Methode für jede Funktion oder jeden Service hinzu, auf den Sie vom EIS zugreifen möchten. Zusätzliche Methoden können zu einem späteren Zeitpunkt im Java-Quelleneditor über die Ansicht 'Snippets' hinzugefügt werden. Klicken Sie auf die Schaltfläche **'Hinzufügen...'**, wählen Sie den Namen für die Methode, und klicken Sie auf **'Weiter'**.

6. Wählen Sie nun **'Durchsuchen...'**, um vorhandene Typen wiederzuverwenden, oder **'Neu...'**, um den CICS/IMS Java-Datenbinding-Assistenten (in dem Sie auf eine COBOL- oder C-Quellendatei verweisen können) für die Eingabe- und Ausgabedatentypen zu starten.
7. Wenn Sie die Erstellung von Java-Methoden abgeschlossen haben, klicken Sie auf **'Weiter'**.
8. Schließen Sie die verbleibenden Schritte in diesem Assistenten ab, um Ihre J2C Java-Bean zu erstellen.
9. Erstellen Sie den Web-Service durch Klicken auf **'Datei' → 'Neu' → 'J2C' → 'Webseite, Web-Service oder EJB aus J2C-JavaBean'**, um den Service um Ihre J2C Java-Bean herum zu erstellen.
10. Beenden Sie den Assistenten.

Die Verbraucher dieses Services können nun den WSDL-Service verwenden, der mit diesem Assistenten generiert wird, um den IMS-Service aufzurufen.

Migration eines J2C-CICS ECI-Services:

Sie können einen J2C-CICS ECI-Service zu einem SCA-Import mit EIS-Binding oder einem SCA-Import mit Web-Service-Binding migrieren.

Befolgen Sie die Anweisungen im Thema "Migration eines J2C-IMS-Serviceprojekts", stellen Sie jedoch sicher, die folgende RAR-Datei *anstelle* der IMS RAR-Datei zu importieren:

- Durchsuchen Sie das Verzeichnis, in dem WebSphere Integration Developer installiert ist, und führen Sie eine Detailabfrage/-analyse unter **Ressourcenadapter → cics15 → cicseci.rar** durch.

Wenn Sie die zweite Option wählen, um einen J2C Web-Service zu erstellen, wählen Sie den V1.5 **ECIResourceAdapter** im zweiten Fenster des J2C-Assistenten für die Erstellung von Java-Beans.

Weitere Informationen finden Sie auch im Thema "Migration eines J2C-IMS-Services".

Migration eines J2C-CICS EPI-Services:

Es gibt keine direkte Unterstützung für den J2C-CICS EPI-Service in WebSphere Integration Developer. Um auf diesen Service von einem SCA-Modul aus zuzugreifen, müssen Sie mithilfe des *Auslastungsszenarios* migrieren.

Anweisungen zur Migration dieses Servicetyps zu WebSphere Integration Developer finden Sie im Thema "Auslastungsszenario für Servicemigration".

Migration eines J2C-HOD-Services:

Es gibt keine direkte Unterstützung für den J2C-HOD-Service in WebSphere Integration Developer. Um auf diesen Service von einem SCA-Modul aus zuzugreifen, müssen Sie mithilfe des *Auslastungsszenarios* migrieren.

Anweisungen zur Migration dieses Servicetyps zu WebSphere Integration Developer finden Sie im Thema "Auslastungsszenario für Servicemigration".

Migration eines Umsetzungs- Serviceprojekts:

Sie können einen Umsetzungsservice wenn möglich zu einer SCA-Datenzuordnung und Schnittstellenzuordnung migrieren. Sie können auch das *Auslastungsszenario* für den Zugriff auf diesen Service von einem SCA-Modul aus verwenden.

Die Datenzuordnungs- und Schnittstellenzuordnungskomponenten sind neu in Version 6.0. Sie bieten ähnliche Funktionen wie der Umsetzungsservice in 5.1, verfügen jedoch nicht über die vollständige XSL-Umsetzungsfunktionalität. Wenn Sie Ihren Umsetzungsservice nicht mit einer dieser Komponenten erset-

zen können, müssen Sie mithilfe des Auslastungsszenarios migrieren, da es keine direkte Unterstützung für den Umsetzungsservice in WebSphere Integration Developer gibt. Führen Sie die Schritte durch, die im Abschnitt "Auslastungsszenario für Servicemigration" dokumentiert sind, um auf diesen Service von einem SCA-Modul aus zuzugreifen.

Auslastungsszenario für Servicemigration:

Für die Fälle, in denen es keine direkte Entsprechung für einen WebSphere Studio Application Developer Integration Edition-Servicetyp gibt, wird ein Auslastungsszenario benötigt, um den alten WebSphere Studio Application Developer Integration Edition-Service ohne Wartung (auf AS-IS-Basis) beim Neuentwurf der Anwendung in WebSphere Integration Developer auszulasten.

Im Folgenden finden Sie die Schritte, die in WebSphere Studio Application Developer Integration Edition vor dem Aufruf des Migrationsassistenten durchgeführt werden müssen:

1. Erstellen Sie ein neues Java-Projekt, das diesen Client-Proxy-Code enthalten soll. Fügen Sie diesen Client-Proxy-Code nicht in das Serviceprojekt ein, da die generierten Nachrichten im 5.1-Stil und Java Bean-Klassen vom automatischen Migrationsassistenten übersprungen werden, der die Serviceprojekte migriert.
2. Öffnen Sie WebSphere Studio Application Developer Integration Edition und klicken Sie mit der rechten Maustaste auf die WSDL-Datei, die das Umsetzungsbinding und den Service enthält und wählen Sie **'Unternehmensservices' → 'Service-Proxy generieren'**. Sie werden gefragt, welchen Proxy-Typ Sie erstellen möchten, es ist jedoch nur **Web Services Invocation Framework (WSIF)** verfügbar. Klicken Sie auf **'Weiter'**.
3. Sie können nun das Paket und den Namen der zu erstellenden Service-Proxy Java-Klasse angeben (Sie erstellen den Proxy im aktuellen Serviceprojekt). Klicken Sie auf **'Weiter'**.
4. Geben Sie nun den Proxy-Stil an, wählen Sie **'Client Stub'** und die gewünschten in den Proxy einzuschließenden Operationen, und klicken Sie auf **Fertig stellen**. Dadurch wird eine Java-Klasse erstellt, die dieselben Methoden wie der WebSphere Studio Application Developer Integration Edition-Service zugänglich macht, wobei die Argumente zu den Java-Methoden einen Abschnitt der WSDL-Quellen-
nachricht darstellen.

Nun können Sie zu WebSphere Integration Developer migrieren:

1. Kopieren Sie das Client-Proxy-Java-Projekt in den neuen Arbeitsbereich und importieren Sie es durch Klicken auf **'Datei' → 'Importieren' → 'Vorhandenes Projekt in Arbeitsbereich'**.
2. Importieren Sie das Serviceprojekt mit dem Migrationsassistenten. Dies resultiert in der Erstellung eines Geschäftsintegrationsmoduls mit den WSDL-Nachrichten, PortTypes, Bindings und Services, die in WebSphere Studio Application Developer Integration Edition generiert werden.
3. Erweitern Sie in der Geschäftsintegrationsperspektive das Modul, sodass Sie seinen Inhalt anzeigen können. Öffnen Sie den Assembly-Editor durch Doppelklicken auf das erste Element unterhalb des Modulprojekts (es verfügt über denselben Namen wie das Projekt).
4. Zur Erstellung der benutzerdefinierten Java-Komponente erweitern Sie **Schnittstellen** unter dem Modulprojekt und wählen Sie die WSDL-Schnittstelle, die für diesen Umsetzungsservice in WebSphere Studio Application Developer Integration Edition generiert wurde.
5. Ziehen und übergeben (Drag-and-drop) Sie diese Schnittstelle an den Assembly-Editor. Es wird ein Dialog angezeigt, der Sie nach dem zu erstellenden Komponententyp fragt. Wählen Sie **'Komponente mit keinem Implementierungstyp'** und klicken Sie auf **OK**.
6. Eine generische Komponente wird im Assemblydiagramm angezeigt. Markieren Sie sie und gehen Sie zur Ansicht **'Eigenschaften'**.
7. In der Registerkarte **'Beschreibung'** können Sie den Namen und Anzeigenamen der Komponente in einen aussagekräftigeren Namen ändern (wählen Sie in diesem Fall etwas Ähnliches wie Ihren EJB-Namen, fügen Sie jedoch eine Erweiterung wie beispielsweise "JavaMed" hinzu, da es sich hierbei um eine Java-Komponente handeln wird, die zwischen der WSDL-Schnittstelle, die für den Umsetzungs-

service in WebSphere Studio Application Developer Integration Edition generiert wurde, und der Java-Schnittstelle des Umsetzungs-Client-Proxy vermittelt).

8. Auf der Registerkarte **'Details'** können Sie sehen, dass diese Komponente über eine Schnittstelle verfügt, nämlich die, die Sie zum Assembly-Editor gezogen und an diesen übergeben haben.
9. Klicken Sie im Assembly-Editor mit der rechten Maustaste auf die Komponente, die Sie soeben erstellt haben, und wählen Sie **'Implementierung generieren...'** → **Java**. Wählen Sie daraufhin das Paket aus, in dem die Java-Implementierung generiert wird. Dadurch wird ein Java-Gerüstservice erstellt, der sich an die WSDL-Schnittstelle gemäß dem SCA-Programmiermodell hält, in dem komplexe Typen durch ein `commonj.sdo.DataObject`-Objekt dargestellt werden und einfache Typen durch ihre funktional entsprechenden Java-Objekte.

Sie müssen nun Code dort eingeben, wo die `//TODO`-Tags in der generierten Java-Implementierungsklasse angezeigt werden. Es gibt zwei Optionen:

1. Verschieben Sie die Logik von der ursprünglichen Java-Klasse in diese Klasse und passen Sie sie an die neue Datenstruktur an.
2. Erstellen Sie eine private Instanz der alten Java-Klasse in dieser generierten Java-Klasse und schreiben Sie Code, um Folgendes zu tun:
 - a. Konvertieren aller Parameter der generierten Java-Implementierungsklasse in Parameter, die die alte Java-Klasse erwartet
 - b. Aufrufen der privaten Instanz der alten Java-Klasse mit den konvertierten Parametern
 - c. Konvertieren des Rückgabewerts der alten Java-Klasse in den Rückgabewerttyp, der von der generierten Java-Implementierungsmethode deklariert wird

Wenn Sie die obenstehenden Optionen abgeschlossen haben, müssen Sie den Client-Proxy neu verbinden. Es sollten keine "Verweise" vorhanden sein, daher müssen Sie die Schnittstelle der Java-Komponente neu verbinden:

- Wenn dieser Service durch einen Geschäftsprozess in demselben Modul aufgerufen wird, stellen Sie eine Verbindung vom entsprechenden Geschäftsprozessverweis zur Schnittstelle dieser Java-Komponente her.
- Wenn dieser Service von einem Geschäftsprozess in einem anderen Modul aufgerufen wird, erstellen Sie einen **Export mit SCA-Binding** und ziehen und übergeben Sie diesem Export vom anderen Modul zum Assembly-Editor dieses Moduls, um den entsprechenden **Import mit SCA-Binding** zu erstellen. Verbinden Sie den entsprechenden Geschäftsprozessverweis mit diesem Import.
- Wenn dieser Service in WebSphere Studio Application Developer Integration Edition veröffentlicht wurde, um ihn extern zugänglich zu machen, finden Sie Informationen zur Neuveröffentlichung dieses Services im Abschnitt "SCA-Exporte für den Zugriff auf den migrierten Service erstellen".

SCA-Exporte für den Zugriff auf den migrierten Service erstellen:

Ein SCA-Export muss erstellt werden, um den migrierten Service für externe Verbraucher gemäß dem SCA-Modell für alle Services verfügbar zu machen, für die im WebSphere Studio Application Developer Integration Edition-Serviceprojekt Implementierungscode generiert wurde. Dies schließt alle Services ein, die durch anwendungsexterne Clients aufgerufen werden.

Wenn Sie in WebSphere Studio Application Developer Integration Edition mit der rechten Maustaste auf den BPEL-Prozess oder sonstige Service WSDL geklickt haben und **'Unternehmensservices'** → **'Implementierungscode generieren'** ausgewählt haben, müssen Sie die folgenden manuellen Migrationsschritte ausführen. Beachten Sie, dass sich WebSphere Integration Developer von WebSphere Studio Application Developer Integration Edition insofern unterscheidet, als dass es alle Implementierungsoptionen speichert. Bei der Erstellung des Projekts wird der Implementierungscode automatisch in der generierten EJB und den Web-Projekten aktualisiert, daher gibt es die Option zum manuellen **'Implementierungscode erstellen'** nicht mehr.

Es wurden fünf Bindingoptionen im Abschnitt 'Schnittstellen für Partner' des Assistenten zur Generierung von BPEL-Implementiercode zur Verfügung gestellt. Die folgende eingehende BPEL Service-Migrationsinformation bietet Einzelheiten zum Exporttyp und den Eigenschaften, die basierend auf den Implementier-Bindingtypen, die in WebSphere Studio Application Developer Integration Edition ausgewählt wurden zu erstellen sind:

- EJB
- IBM Web Service (SOAP/JMS)
- IBM Web Service (SOAP/HTTP)
- Apache Web Service (SOAP/HTTP)
- JMS

Migration von EJB und EJB-Prozessbindings:

EJB und EJB-Prozessbindings können zur empfohlenen SCA-Anweisung migriert werden.

In WebSphere Studio Application Developer Integration Edition hat dieser Bindingtyp Clients die Kommunikation mit einem BPEL-Prozess oder anderen Servicetyp durch Aufruf eines EJB ermöglicht. Beachten Sie, dass dieser Bindingtyp für Mikroprozesse nicht optional war – er wurde immer ausgewählt, wenn das generierte EJB intern von den anderen Bindingtypen verwendet wurde.

Der JNDI-Name des generierten EJB wurde automatisch als eine Kombination aus BPEL-Name, Zielnamensbereich sowie 'Gültig ab'-Zeitmarke generiert. Die folgenden Attribute können beispielsweise durch Überprüfung der BPEL-Prozesseigenschaften im BPEL Editor auf den Registerkarten 'Beschreibung' und 'Serverinhalt' gefunden werden:

Tabelle 3. Generierter Namensbereich

Prozessname	MyService
Zielnamensbereich	http://www.example.com/process87787141/
Gültig ab	Jan 01 2003 02:03:04

Der generierte Namensbereich für dieses Beispiel lautet daraufhin com/example/www/process87787141/MyService20030101T020304.

In WebSphere Studio Application Developer Integration Edition gab es keine Optionen zur Auswahl, wenn EJB-Binding als Implementierungstyp ausgewählt wurde.

Es gibt vier Optionen für die Migration des WebSphere Studio Application Developer Integration Edition-Prozessbinding. Die Clienttypen, die auf den Service zugreifen, bestimmen, welche der unten aufgelisteten Migrationsoptionen durchgeführt werden:

Anmerkung: Wenn die manuellen Migrationsschritte abgeschlossen sind, muss der Client ebenso zum neuen Programmiermodell migriert werden. Weitere Informationen finden Sie im entsprechenden Thema für die folgenden Client-Typen:

Tabelle 4. Weitere Informationen für die Migration von Clients

Clienttyp	Weitere Informationen finden Sie in
EJB Client, der die generierte Session-Bean aufruft. Ein solcher Client ruft eine EJB-Methode auf, die der aufzurufenden BPEL-Operation entspricht	Migration des EJB Client
WSIF Client, der das EJB Prozessbinding verwendet	Migration des EJB-Prozessbinding-Client
Generischer Geschäftsprozess-Choreograph EJB API	Migration des generischen Geschäftsprozess-Choreograph EJB API Client

Tabelle 4. Weitere Informationen für die Migration von Clients (Forts.)

Clienttyp	Weitere Informationen finden Sie in
Generische Geschäftsprozess-Choreographer-Nachrichtenübertragungs-API	Migration des generischen Geschäftsprozess-Choreographer Messaging API Client
Ein weiterer BPEL-Prozess im gleichen Modul	N/V: BPEL-Komponenten mithilfe des Assembly-Editor verbinden
Ein weiterer BPEL-Prozess in einem anderen Modul	N/V: Einen 'Import mit SCA-Binding' im Referenzmodul erstellen, und dessen Binding mit dem Verweis auf 'Export mit SCA-Binding' , das Sie weiter unten in Option 1 erstellt haben, konfigurieren

Wenn der Geschäftsprozess einen Verweis an sich selbst außerhalb seines Moduls übergibt (über einen Serviceverweis), müssen Sie immer der untenstehenden Option 1 folgen (Sie können jederzeit mehr als eine dieser Optionen durchführen), um einen Export mit SCA-Binding für den Geschäftsprozess durchzuführen. Es kann nur ein Geschäftsprozess pro Modul seinen Serviceverweis außerhalb des Moduls übergeben, da sein Export als der Standardexport des Moduls markiert sein muss. Dies geschieht durch Angabe von "true" für das Attribut namens "default" eines Exports, wie in:

Standardendpunktverweis

Sie müssen den Export dieses Geschäftsprozesses manuell als Standard markieren, indem Sie mit der rechten Maustaste auf den Export in der Ansicht 'Geschäftsintegration' klicken und **'Öffnen mit'** sowie **'Texteditor'** auswählen.

Migrationsoption 1 für EJB und EJP-Prozessbinding:

Die erste Migrationsoption für das WebSphere Studio Application Developer Integration Edition EJB-Prozessbinding ist die Ermöglichung des Zugriffs auf Geschäftsprozesse durch eine andere Komponente in demselben Modul.

Verbinden Sie diese andere Komponente mit der BPEL-Komponente im Assembly-Editor:

1. Wählen Sie das Element **'Verbinden'** in der Symbolleiste.
2. Klicken Sie auf die andere Komponente, um diese als Quelle der Verbindung auszuwählen.
3. Klicken Sie auf die Komponente **BPEL SCA**, um diese als Ziel der Verbindung auszuwählen.
4. Speichern Sie das Assembly-Diagramm.

Migrationsoption 2 für EJB und EJP-Prozessbinding:

Die zweite Migrationsoption für das WebSphere Studio Application Developer Integration Edition EJB-Prozessbinding ist die Ermöglichung des Zugriffs auf Geschäftsprozesse durch sonstige SCA-Module und Clients.

Anmerkung: Diese Schritte sind verbindlich, wenn die generischen Geschäftsprozess-Choreographer-APIs für den Aufruf des Geschäftsprozesses verwendet werden.

Der Export mit SCA-Binding ermöglicht den Zugriff auf eine SCA-Komponente durch andere SCA-Module. So erstellen Sie einen Export mit SCA-Binding:

1. Öffnen Sie den Assembly-Editor für das vom Migrationsassistenten erstellte Modul.
2. Erstellen Sie einen Export mit SCA-Bindung für jede BPEL-Prozessschnittstelle, für die ein EJB-Binding in WebSphere Studio Application Developer Integration Edition generiert wurde:
 - a. Klicken Sie mit der rechten Maustaste auf die BPEL-Komponente im Assembly-Editor.
 - b. Wählen Sie **'Exportieren...'**.
 - c. Wählen Sie **'SCA-Binding'**.

- d. Wenn es mehrere Schnittstellen für den Prozess gibt, wählen Sie die zu exportierende Schnittstelle(n) mit diesem Bindingtyp.
- e. Wenn der SCA-Export erstellt wurde, wählen Sie den Export im Assembly-Editor und wählen Sie in der Ansicht 'Eigenschaften' das Inhaltsteilfenster **'Beschreibung'**. Der Name und die Beschreibung des Exports werden aufgelistet und können auf Wunsch geändert werden.
- f. Speichern Sie das Assembly-Diagramm.

Migrationsoption 3 für EJB und EJP-Prozessbinding:

Die dritte Migrationsoption für das WebSphere Studio Application Developer Integration Edition EJB-Prozessbinding ist es, Module durch eine nicht-SCA-Entität (beispielsweise eine JSP oder einen Java-Client) zugänglich zu machen.

Die Standalone Reference macht eine SCA-Komponente durch einen beliebigen externen Client zugänglich. So erstellen Sie eine Standalone Reference:

1. Öffnen Sie den Assembly-Editor für das vom Migrationsassistenten erstellte Modul.
2. Erstellen Sie eine Standalone Reference für jede BPEL-Prozessschnittstelle, für die ein EJB-Binding in WebSphere Studio Application Developer Integration Edition generiert wurde:
 - a. Wählen Sie das Element **'Standalone References'** in der Symbolleiste.
 - b. Klicken Sie auf den Erstellungsbereich des Assembly-Editors, um eine Standalone References SCA-Entität zu erstellen.
 - c. Wählen Sie das Element **'Verbinden'** in der Symbolleiste.
 - d. Klicken Sie auf die Entität **'Standalone References'**, um sie als Quelle für die Verbindung auszuwählen.
 - e. Klicken Sie auf die Komponente **BPEL SCA**, um diese als Ziel der Verbindung auszuwählen.
 - f. Es erscheint die Warnung **'Am Quellenknoten wird ein übereinstimmender Verweis erstellt. Möchten Sie fortfahren?'**. Klicken Sie auf **OK**.
 - g. Wählen Sie die Entität **'Standalone References'**, die soeben erstellt wurde, und wählen Sie in der Ansicht 'Eigenschaften' das Inhaltsteilfenster **'Beschreibung'**.
 - h. Erweitern Sie den Link **"Verweise"** und wählen Sie die soeben erstellte Referenz. Der Name und die Beschreibung der Referenz werden aufgelistet und können auf Wunsch geändert werden.
 - i. Wenn es mehrere Schnittstellen für den Prozess gibt, wählen Sie die zu exportierende Schnittstelle(n) mit diesem Bindingtyp.
 - j. Speichern Sie das Assembly-Diagramm.

Migrationsoption 4 für EJB und EJP-Prozessbinding:

Die vierte Migrationsoption für das WebSphere Studio Application Developer Integration Edition EJB-Prozessbinding ist die Ermöglichung des Zugriffs auf Geschäftsprozesse durch einen Web-Services-Client.

Der Export mit Web-Service-Binding ermöglicht den Zugriff auf eine SCA-Komponente über einen externen Web-Services-Client. So erstellen Sie einen Export mit Web-Service-Binding:

1. Öffnen Sie den Assembly-Editor für das vom Migrationsassistenten erstellte Modul.
2. Erstellen Sie einen Export mit SCA-Bindung für jede BPEL-Prozessschnittstelle, für die ein EJB-Binding in WebSphere Studio Application Developer Integration Edition generiert wurde:
 - a. Klicken Sie mit der rechten Maustaste auf die BPEL-Komponente im Assembly-Editor.
 - b. Wählen Sie **'Exportieren...'**.
 - c. Wählen Sie **'Web-Service-Binding'**.
 - d. Wenn es mehrere Schnittstellen für den Prozess gibt, wählen Sie die zu exportierende Schnittstelle(n) mit diesem Bindingtyp.
 - e. Wählen Sie den Transport: **soap/http** oder **soap/jms**.

- f. Wenn der Web-Services-Export erstellt wurde, wählen Sie den Export im Assembly-Editor und wählen Sie in der Ansicht 'Eigenschaften' das Inhaltsteilfenster '**Beschreibung**'. Der Name und die Beschreibung des Exports werden aufgelistet und können auf Wunsch geändert werden.
- g. Speichern Sie das Assembly-Diagramm.

Migration von JMS und JMS-Prozessbindings:

JMS und JMS-Prozessbindings können zur empfohlenen SCA-Anweisung migriert werden.

In WebSphere Studio Application Developer Integration Edition hat dieser Bindingtyp Clients die Kommunikation mit einem BPEL-Prozess oder anderen Servicetyp durch Senden einer Nachricht an ein MDB ermöglicht. Beachten Sie, dass dieser Bindingtyp für Prozesse mit langer Laufzeit nicht optional war und immer ausgewählt war. Tatsächlich war dieser Bindingtyp der **einzigste** Bindingtyp, der für Request-Response-Schnittstellen von Prozessen mit langer Laufzeit zulässig war. Bei andere Servicetypen würde eine MDB generiert werden, die den entsprechenden Service aufruft.

Der vom JMS Binding verwendete JNDI-Name war eine Kombination aus BPEL-Name, Zielnamensbereich sowie 'Gültig an'-Zeitmarke.

In WebSphere Studio Application Developer Integration Edition gab es die folgenden Optionen zur Auswahl, wenn JMS Binding als Implementiertyp für einen BPEL-Prozess ausgewählt wurde:

- **JNDI Verbindungsfactory** - Standard ist jms/BPECF (dies ist der JNDI-Name der Verbindungsfactory-Warteschlange des Geschäftsprozesscontainers)
- **JNDI-Zielwarteschlange** - Standard ist jms/BPEIntQueue (dies ist der JNDI-Name der internen Warteschlange des Geschäftsprozesscontainers)
- **JNDI Provider URL: Vom Server bereitgestellt oder benutzerdefiniert** - Sie müssen eine Adresse eingeben. Standard ist iiop://localhost:2809

Es gibt fünf Optionen für die Migration des WebSphere Studio Application Developer Integration Edition JMS-Prozessbinding. Die Clienttypen, die auf den Service zugreifen, bestimmen, welche der unten aufgelisteten Migrationsoptionen durchgeführt werden:

Anmerkung: Wenn die manuellen Migrationsschritte abgeschlossen sind, muss der Client ebenso zum neuen Programmiermodell migriert werden. Weitere Informationen finden Sie im entsprechenden Thema für die folgenden Client-Typen:

Tabelle 5. Weitere Informationen für die Migration von Clients

Clienttyp	Weitere Informationen finden Sie in
WSIF Client, der JMS-Prozess-Binding verwendet	Migration des generischen Geschäftsprozess-Choreographer Messaging API Client und des JMS-Prozessbinding-Client
Generischer Geschäftsprozess-Choreographer EJB API	Migration des generischen Geschäftsprozess-Choreographer EJB API Client
Migration des Unternehmens durch generische Geschäftsprozess-Choreographer-Nachrichtenübertragungs-API	Migration des generischen Geschäftsprozess-Choreographer Messaging API Client
Ein weiterer BPEL-Prozess im gleichen Modul	N/V: BPEL-Komponenten mithilfe des Assembly-Editor verbinden
Ein weiterer BPEL-Prozess in einem anderen Modul	N/V: Einen 'Import mit SCA-Binding' im Referenzmodul erstellen, und dessen Binding mit dem Verweis auf 'Export mit SCA-Binding', das Sie weiter unten in Option 1 erstellt haben, konfigurieren

Wenn der Geschäftsprozess einen Verweis an sich selbst außerhalb seines Moduls übergibt

(über einen Serviceverweis), müssen Sie immer der untenstehenden Option 1 folgen (Sie können jederzeit mehr als eine dieser Optionen durchführen), um einen Export mit SCA-Binding für den Geschäftsprozess durchzuführen. Es kann nur ein Geschäftsprozess pro Modul seinen Serviceverweis außerhalb des Moduls übergeben, da sein Export als der Standardexport des Moduls markiert sein muss. Dies geschieht durch Angabe von "true" für das Attribut namens "default" eines Exports, wie in:

Standardendpunktverweis

Sie müssen den Export dieses Geschäftsprozesses manuell als Standard markieren, indem Sie mit der rechten Maustaste auf den Export in der Ansicht 'Geschäftsintegration' klicken und **'Öffnen mit'** sowie **'Texteditor'** auswählen.

Migrationsoption 1 für JMS und JMS-Prozessbinding:

Die erste Migrationsoption für das WebSphere Studio Application Developer Integration Edition JMS-Prozessbinding ist die Ermöglichung des Zugriffs auf Geschäftsprozesse durch eine andere Komponente in demselben Modul.

Verbinden Sie diese andere Komponente mit der BPEL-Komponente im Assembly-Editor:

1. Wählen Sie das Element **'Verbinden'** in der Symbolleiste.
2. Klicken Sie auf die andere Komponente, um diese als Quelle der Verbindung auszuwählen.
3. Klicken Sie auf die Komponente **BPEL SCA**, um diese als Ziel der Verbindung auszuwählen.
4. Speichern Sie das Assembly-Diagramm.

Migrationsoption 2 für JMS und JMS-Prozessbinding:

Die zweite Migrationsoption für das WebSphere Studio Application Developer Integration Edition JMS-Prozessbinding ist die Ermöglichung des Zugriffs auf Geschäftsprozesse durch sonstige SCA-Module und Clients.

Der Export mit SCA-Binding ermöglicht den Zugriff auf eine SCA-Komponente durch andere SCA-Module. So erstellen Sie einen Export mit SCA-Binding:

1. Öffnen Sie den Assembly-Editor für das vom Migrationsassistenten erstellte Modul.
2. Erstellen Sie einen Export mit SCA-Binding für jede BPEL-Prozessschnittstelle, für die ein JMS-Binding in WebSphere Studio Application Developer Integration Edition generiert wurde:
 - a. Klicken Sie mit der rechten Maustaste auf die BPEL-Komponente im Assembly-Editor.
 - b. Wählen Sie **'Exportieren...'**.
 - c. Wählen Sie **'SCA-Binding'**.
 - d. Wenn es mehrere Schnittstellen für den Prozess gibt, wählen Sie die zu exportierende Schnittstelle(n) mit diesem Bindingtyp.
 - e. Wenn der SCA-Export erstellt wurde, wählen Sie den Export im Assembly-Editor und wählen Sie in der Ansicht 'Eigenschaften' das Inhaltsteilfenster **'Beschreibung'**. Der Name und die Beschreibung des Exports werden aufgelistet und können auf Wunsch geändert werden.
 - f. Speichern Sie das Assembly-Diagramm.

Migrationsoption 3 für JMS und JMS-Prozessbinding:

Die dritte Migrationsoption für das WebSphere Studio Application Developer Integration Edition JMS-Prozessbinding ist es, Geschäftsprozesse durch eine nicht-SCA-Entität (beispielsweise eine JSP oder einen Java-Client) zugänglich zu machen.

Die Standalone Reference macht eine SCA-Komponente durch einen beliebigen externen Client zugänglich. So erstellen Sie eine Standalone Reference:

1. Öffnen Sie den Assembly-Editor für das vom Migrationsassistenten erstellte Modul.
2. Erstellen Sie eine Standalone Reference für jede BPEL-Prozessschnittstelle, für die ein JMS-Binding in WebSphere Studio Application Developer Integration Edition generiert wurde:
 - a. Wählen Sie das Element **'Standalone References'** in der Symbolleiste.
 - b. Klicken Sie auf den Erstellungsbereich des Assembly-Editors, um eine Standalone References SCA-Entität zu erstellen.
 - c. Wählen Sie das Element **'Verbinden'** in der Symbolleiste.
 - d. Klicken Sie auf die Entität **'Standalone References'**, um sie als Quelle für die Verbindung auszuwählen.
 - e. Klicken Sie auf die Komponente **BPEL SCA**, um diese als Ziel der Verbindung auszuwählen.
 - f. Es erscheint die Warnung **'Am Quellenknoten wird ein übereinstimmender Verweis erstellt. Möchten Sie fortfahren?'**. Klicken Sie auf **OK**.
 - g. Wählen Sie die Entität **'Standalone References'**, die soeben erstellt wurde, und wählen Sie in der Ansicht **'Eigenschaften'** das Inhaltsteilfenster **'Beschreibung'**.
 - h. Erweitern Sie den Link **'Verweise'** und wählen Sie die soeben erstellte Referenz. Der Name und die Beschreibung der Referenz werden aufgelistet und können auf Wunsch geändert werden.
 - i. Wenn es mehrere Schnittstellen für den Prozess gibt, wählen Sie die zu exportierende Schnittstelle(n) mit diesem Bindingtyp.
 - j. Speichern Sie das Assembly-Diagramm.

Migrationsoption 4 für JMS und JMS-Prozessbinding:

Die vierte Migrationsoption für das WebSphere Studio Application Developer Integration Edition JMS-Prozessbinding ist die Ermöglichung des Zugriffs auf Geschäftsprozesse durch einen Web-Services-Client.

Der Export mit Web-Service-Binding ermöglicht den Zugriff auf eine SCA-Komponente über einen externen Web-Services-Client. So erstellen Sie einen Export mit Web-Service-Binding:

1. Öffnen Sie den Assembly-Editor für das vom Migrationsassistenten erstellte Modul.
2. Erstellen Sie einen Export mit SCA-Binding für jede BPEL-Prozessschnittstelle, für die ein JMS-Binding in WebSphere Studio Application Developer Integration Edition generiert wurde:
 - a. Klicken Sie mit der rechten Maustaste auf die BPEL-Komponente im Assembly-Editor.
 - b. Wählen Sie **'Exportieren...'**.
 - c. Wählen Sie **'Web-Service-Binding'**.
 - d. Wenn es mehrere Schnittstellen für den Prozess gibt, wählen Sie die zu exportierende Schnittstelle(n) mit diesem Bindingtyp.
 - e. Wählen Sie den Transport: **soap/http** oder **soap/jms**.
 - f. Wenn der Web-Services-Export erstellt wurde, wählen Sie den Export im Assembly-Editor und wählen Sie in der Ansicht **'Eigenschaften'** das Inhaltsteilfenster **'Beschreibung'**. Der Name und die Beschreibung des Exports werden aufgelistet und können auf Wunsch geändert werden.
 - g. Speichern Sie das Assembly-Diagramm.

Migrationsoption 5 für JMS und JMS-Prozessbinding:

Die fünfte Migrationsoption für das WebSphere Studio Application Developer Integration Edition JMS-Prozessbinding ist die Ermöglichung des Zugriffs auf Geschäftsprozesse durch einen JMS-Client.

Der Export mit JMS-Binding ermöglicht den Zugriff auf eine SCA-Komponente über einen externen JMS-Client. So erstellen Sie einen Export mit JMS-Binding:

1. Sie müssen für BPEL-Services neue Warteschlangenressourcen erstellen und auf diese Ressourcen verweisen, da das 5.1-JMS-Prozessbinding deutlich vom 5.1-JMS-Standardbinding abweicht. Bei anderen Servicetypen können Sie die Werte, die Sie für den JMS-Implementierungscode in WebSphere

Studio Application Developer Integration Edition 5.1 ausgewählt haben, ermitteln, indem Sie nach der WSDL-Datei namens **JMSBinding.wsdl** und **JMSService.wsdl** im entsprechenden Paket unterhalb des Ordners **ejbModule/META-INF** für das generierte EJB-Projekt suchen und die dort erfassten Binding- und Serviceinformationen untersuchen. Anhand des Bindings können Sie ermitteln, ob Text- oder Objektnachrichten verwendet wurden und ob angepasste Datenformatbindings zum Einsatz kamen. In diesem Fall ist es sinnvoll, auch für den **Export mit JMS-Binding** von 6.0 ein angepasstes Datenbinding zu schreiben. Anhand des Services können Sie die Ausgangskontextfactory, den Namen der JNDI-Verbindungsfactory, den JNDI-Zieladressnamen und den Zieladressstil (Warteschlange) ermitteln.

2. Öffnen Sie den Assembly-Editor für das vom Migrationsassistenten erstellte Modul.
3. Erstellen Sie einen Export mit JMS-Binding für jede BPEL-Prozessschnittstelle, für die ein JMS-Binding in WebSphere Studio Application Developer Integration Edition generiert wurde, durch Klicken mit der rechten Maustaste auf die BPEL-Komponente im Assembly-Editor.
4. Wählen Sie **'Exportieren...'**.
5. Wählen Sie **'JMS-Binding'**.
6. Wenn es mehrere Schnittstellen für den Prozess gibt, wählen Sie die zu exportierende Schnittstelle(n) mit diesem Bindingtyp.
7. Wählen Sie in der nächsten Anzeige (JMS-Exportbindingattribute) die Option **'JMS-Nachrichtendomäne'**. Definieren Sie dieses Attribut als **Punkt-zu-Punkt**.
8. Wählen Sie, **wie Daten zwischen dem Geschäftsobjekt und der JMS-Nachricht serialisiert** werden, und geben Sie die folgenden Werte ein (es empfiehlt sich, die Option **'Text'** anstelle von **'Objekt'** auszuwählen, weil der (normalerweise in XML stehende) Text von der Laufzeit unabhängig ist und die Serviceintegration zwischen unterschiedlichen und voneinander unabhängigen Systemen ermöglicht):
 - a. Für **'Text'**, wählen Sie die Verwendung des **JMS-Standardfunktionsselektors** oder geben Sie den vollständig qualifizierten Namen der FunctionSelector-Implementierungsklasse ein.
 - b. Für **'Object'**, wählen Sie die Verwendung des **JMS-Standardfunktionsselektors** oder geben Sie den vollständig qualifizierten Namen der FunctionSelector-Implementierungsklasse ein.
 - c. Für **'Vom Benutzer bereitgestellt'** geben Sie den vollständig qualifizierten Namen der JMSData-Binding-Implementierungsklasse ein. Die Auswahl von **'Vom Benutzer bereitgestellt'** ist erforderlich, wenn Ihre Anwendung auf JMS-Headereigenschaften zugreifen muss, die im JMS-Importbinding nicht verfügbar sind. In diesem Fall müssen Sie dann eine Klasse für das angepasste Datenbinding erstellen, die das JMS-Standarddatenbinding **com.ibm.websphere.sca.jms.data.JMSDataBinding** erweitert, und angepassten Code für den direkten Zugriff auf **'JMSMessage'** hinzufügen. Danach geben Sie den Namen der angepassten Klasse für dieses Feld an. Weitere Informationen finden Sie in den JMS-Beispielen des Abschnitts "Bindings für Import- und Exportkomponenten erstellen und ändern", die Sie über den unten stehenden Link aufrufen können.
 - d. Für **'Vom Benutzer bereitgestellt'**, wählen Sie die Verwendung des **JMS-Standardfunktionsselektors** oder geben Sie den vollständig qualifizierten Namen der FunctionSelector-Implementierungsklasse ein.
9. Wenn der Export mit JMS-Binding erstellt wurde, wählen Sie den Export im Assembly-Editor und wählen Sie in der Ansicht 'Eigenschaften' das Inhaltsteilfenster **'Beschreibung'**. Der Name und die Beschreibung des Exports werden aufgelistet und können auf Wunsch geändert werden.
10. Wählen Sie das Inhaltsteilfenster 'Binding', um weitere Optionen anzuzeigen.
11. Speichern Sie das Assembly-Diagramm.

Migration des IBM Web-Service-Binding (SOAP/JMS):

Das IBM Web-Service-Binding (SOAP/JMS) für einen BPEL-Prozess oder einen anderen Servicetyp kann zur empfohlenen SCA-Anweisung migriert werden.

In WebSphere Studio Application Developer Integration Edition, gab dieser Bindingtyp Clients die Möglichkeit, mit einem BPEL-Prozess oder einem anderen Servicetyp durch Aufruf des IBM Web Service zu kommunizieren, wobei es sich beim Übertragungsprotokoll um JMS handelte und die Nachricht die SOAP-Codierregeln befolgte.

Das folgende Beispiel veranschaulicht die Konventionen, die bei der Generierung eines IBM Web-Services (SOAP/JMS) für einen 5.1-BPEL-Service zu Anwendung kommen. Der JNDI-Name des generierten IBM Web Service war eine Kombination aus BPEL-Name, Zielnamensbereich und 'Gültig-ab'-Zeitmarke, sowie dem Namen der Schnittstelle (WSDL-Porttyp, für den der Implementiercode generiert wurde). Die folgenden Attribute können beispielsweise durch Überprüfung der BPEL-Prozesseigenschaften im BPEL-Editor auf den Registerkarten 'Beschreibung' und 'Serverinhalt' gefunden werden:

Tabelle 6. Generierter Namensbereich

Prozessname	MyService
Zielnamensbereich	http://www.example.com/process87787141/
Gültig ab	Jan 01 2003 02:03:04
Schnittstelle	ProcessPortType

Der generierte Namensbereich für dieses Beispiel lautet daraufhin com/example/www/process87787141/MyService20030101T020304_ProcessPortTypePT.

In WebSphere Studio Application Developer Integration Edition gab es die folgenden Optionen zur Auswahl, wenn das IBM Web-Service-Binding (SOAP/JMS) als Implementiertyp für einen BPEL-Prozess oder einen anderen Servicetyp ausgewählt wurde:

- Für Dokumentstil war der Standard **'DOCUMENT / andere Option: RPC'**
- Für Dokumentverwendung war der Standard **'LITERAL / andere Option: ENCODED'**
- Für JNDI Provider URL war es entweder **'Server angegeben'** oder **'Benutzerdefiniert'** (eine Adresse muss eingegeben werden, Standard ist iiop://localhost:2809)
- Für Zieladressstil war der Standard **'Warteschlange / andere Option war Thema'**
- Für JNDI-Verbindungsfactory war der Standard **jms/qcf** (dies ist der JNDI-Name der Warteschlangen-Verbindungsfactory für die generierte MDB-Warteschlange)
- Für JNDI-Zielwarteschlange war der Standard **'jms/Warteschlange'** (dies ist der JNDI-Name der generierten MDB-Warteschlange)
- Für MDB Listener-Port war der Standard **<Serviceprojektname>MdbListenerPort**.

Eine WSDL-Datei, die das IBM Web Service SOAP/JMS-Binding und den Service angibt, wird im generierten EJB-Projekt, jedoch nicht im Serviceprojekt selbst erstellt. Das bedeutet, dass Sie die Datei manuell suchen und in Ihr Geschäftsintegrations-Modulprojekt kopieren müssen, wenn der IBM Web-Service-Client-Code nicht geändert werden darf. Standardmäßig wurde diese WSDL-Datei im EJB-Projekt unter `ejbModule/META-INF/wsdl/<Geschäftsprozessname>_<Geschäftsprozessschnittstellen-Porttypname>_JMS.wsdl` erstellt.

Der WSDL PortType und Nachrichten der Geschäftsprozessschnittstelle werden ebenso in diese WSDL-Datei kopiert, statt auf den vorhandenen WSDL PortType und Nachrichten, die im Serviceprojekt definiert sind, zu verweisen.

Wenn der IBM Web-Service-Client-Code nach der Migration unverändert sein soll, wird die Information in dieser Datei für die unten aufgeführten manuellen Migrationsschritte benötigt.

Es gibt zwei Optionen für die Migration des WebSphere Studio Application Developer Integration Edition SOAP/JMS-Prozessbinding. Es muss entschieden werden, ob der Client zum SCA-Programmiermodell migriert wird oder ob er als Web-Services-Client belassen wird:

Anmerkung: Wenn die manuellen Migrationsschritte abgeschlossen sind, muss der Client ebenso zum neuen Programmiermodell migriert werden. Weitere Informationen finden Sie im entsprechenden Thema für die folgenden Client-Typen:

Tabelle 7. Weitere Informationen für die Migration von Clients

Clienttyp	Weitere Informationen finden Sie in
IBM Web-Service-Client	Migration des IBM Web-Service (SOAP/JMS) -Client

Migrationsoption 1 für das IBM Web-Service-Binding (SOAP/JMS):

Die erste Migrationsoption für das WebSphere Studio Application Developer Integration Edition SOAP/JMS-Binding ermöglicht einem Web-Services-Client den Zugriff auf den Service.

Der Export mit Web-Service-Binding ermöglicht den Zugriff auf eine SCA-Komponente über einen externen Web-Services-Client. So erstellen Sie einen Export mit Web-Service-Binding:

1. Öffnen Sie den Assembly-Editor für das vom Migrationsassistenten erstellte Modul.
2. Erstellen Sie einen Export mit SCA-Binding für jede Serviceschnittstelle, für die ein IBM Web Service (SOAP/JMS)-Binding in WebSphere Studio Application Developer Integration Edition generiert wurde:
 - a. Klicken Sie mit der rechten Maustaste auf die SCA-Komponente im Assembly-Editor.
 - b. Wählen Sie **'Exportieren...'**.
 - c. Wählen Sie **'Web-Service-Binding'**.
 - d. Wenn es mehrere Schnittstellen für die Komponente gibt, wählen Sie die zu exportierende Schnittstelle(n) mit diesem Bindingtyp.
 - e. Wählen Sie die Transport-**soap/jms**.
3. Wenn der Web Services Export erstellt wurde, wählen Sie den Export im Assembly-Editor und wählen Sie in der Ansicht 'Eigenschaften' das Inhaltsteilfenster **'Beschreibung'**. Der Name und die Beschreibung des Exports werden aufgelistet und können auf Wunsch geändert werden.
4. Speichern Sie das Assembly-Diagramm.
5. Wählen Sie das Inhaltsteilfenster 'Binding', in dem Sie sehen können, dass ein IBM Web Service WSDL-Binding und Service direkt im Projektordner des Moduls erstellt wurde. Es erhält den Namen *component-that-was-exportedExportWSDL PortType name Jms_Service.wsdl*. Wenn Sie diese Datei untersuchen, werden Sie feststellen, dass das eingeschlossene Dokument/Literal-Binding standardmäßig verwendet wird, da dies der bevorzugte Stil in 6.0 ist. Dies ist die WSDL, die von IBM Web-Service-Clients für den Aufruf des Service verwendet werden wird.
6. Führen Sie diese Schritte durch, um ein neues Web-Service-Binding und einen Service zu generieren, wenn das Beibehalten des Client-Codes gewünscht ist:
 - a. Kopieren Sie die 5.1 WSDL-Datei aus dem generierten 5.1 EJB-Projekt unter `ejbModule/META-INF/wsdl/Geschäftsprozessname/Geschäftsprozessschnittstellen-PorttypnameJMS.wsdl` in das Geschäftsintegrations-Modulprojekt.
 - b. Nach dem Kopieren der Datei und der Neuerstellung des Moduls werden möglicherweise Fehlermeldungen angezeigt, da die vom Web-Service verwendeten XML-Schematypen, WSDL-Nachrichten und WSDL-Porttypen in der WSDL-Datei des IBM Web-Services in 5.1 doppelt vorhanden sind. Um dies zu beheben, löschen Sie diese doppelt vorhandenen Definitionen aus der Binding/Service-WSDL des IBM Web-Services und fügen Sie an deren Stelle einen WSDL-Import für die echte Schnittstellen-WSDL hinzu. **Hinweis:** Beachten Sie, dass die Schemendefinitionen in bestimmten Fällen geändert wurden, als der IBM Web-Service-Implementierungscode von WebSphere Studio Application Developer Integration Edition generiert wurde. Dies führt unter Umständen zu Inkonsistenzen für vorhandene Clients, die die IBM Web-Service-WSDL verwenden. Das Schemaattribut "elementFormDefault" beispielsweise wurde in dem Inline-Schema, das in der IBM Web-Service-WSDL generiert wurde, auf "qualifiziert" gesetzt, auch wenn die ursprüngliche Schemadefinition nicht qualifiziert war. Dadurch wurde der folgende Fehler während der Laufzeit generiert: WWS3047E: Fehler: Element kann nicht entrealisiert werden.

- c. Klicken Sie mit der rechten Maustaste auf diese WSDL-Datei, die Sie soeben in das Geschäftsintegrationsmodul kopiert haben, und wählen Sie **Öffnen mit** und **WSDL-Editor**.
- d. Gehen Sie zur Registerkarte 'Quelle'. Löschen Sie alle in dieser Datei definierten WSDL PortTypes und Nachrichten.
- e. Daraufhin wird folgender Fehler angezeigt: Der für das '<binding>' Binding angegebene '<portType>' Porttyp ist nicht definiert. Um dies zu beheben, klicken Sie im WSDL-Editor in der Diagrammsicht mit der rechten Maustaste auf den Importabschnitt und wählen Sie **Import hinzufügen**.
- f. Klicken Sie in der Ansicht 'Eigenschaften' in der Registerkarte 'Allgemein' auf die Schaltfläche ... rechts neben dem Feld für die Speicherposition. Blättern Sie zur Schnittstellen-WSDL, in der sich die WSDL-Nachricht und die Porttypdefinitionen befinden und klicken Sie auf **OK**, um die Schnittstellen-WSDL in die Service/Binding-WSDL zu kopieren.
- g. Speichern Sie die WSDL-Datei.
- h. Aktualisieren/erstellen Sie das Projekt neu. Wechseln Sie in die Geschäftsintegrationsperspektive. Öffnen Sie das Assembly-Diagramm des Moduls im Assembly-Editor.
- i. Erweitern Sie das zu migrierende Modul in der Ansicht 'Projektexplorer', und erweitern Sie die logische Kategorie **Web-Service-Ports**. Der in der Binding/Service-WSDL vorhandene Port sollte nun angezeigt werden. Ziehen und übergeben Sie ihn an den Assembly-Editor.
- j. Wählen Sie die Erstellung eines **Exports mit Web-Service-Binding** und wählen Sie den entsprechenden Portnamen. Dadurch wird der Export erstellt, der das alte Binding/Service verwendet, sodass vorhandene Web-Service-Clients nicht geändert werden müssen. Wenn Sie den Export auswählen, den Sie soeben im Assembly-Editor erstellt haben und zur Ansicht 'Eigenschaften' gehen, wird in der Registerkarte 'Binding' angezeigt, dass die 5.1 Port- und Servicenamen für Sie eingetragen wurden.
- k. Speichern Sie alle Änderungen.
- l. Kurz vor der Implementierung der Anwendung können Sie die Konfiguration des generierten Web-Projekts ändern, so dass Sie der 5.1 Serviceadresse entspricht (Sie müssen diese Änderungen jedes Mal durchführen, wenn Sie Änderungen am SCA-Modul vornehmen, durch die diese Datei erneut generiert wird). Wenn Sie die *Service*definition der IBM Web-Service-WSDL, die Sie aus 5.1 wieder verwenden, anzeigen, wird die Serviceadresse angezeigt, zu der der 5.1-Client codiert ist:
`<wsdl:soap:address location="http://localhost:9080/MyServiceWeb/services/MyServicePort"/>`
- m. Damit die generierten 6.0 Webprojekt-Artefakte mit dieser alten Serviceadresse übereinstimmen, sollten Sie den generierten Implementierungsdeskriptor des Webprojekts ändern. Öffnen Sie den Implementierungsdeskriptor in WebSphere Integration Developer und fügen Sie in der Registerkarte 'Servlets' eine zusätzliche URL-Zuordnung hinzu, die der vorhandenen URL-Zuordnung für diesen Export ähnelt, mit dem gleichen Servlet-Namen aber einem anderen URL-Muster.
- n. Wenn Sie außerdem das Kontextstammverzeichnis dieses Web-Projekts ändern müssen, so dass es dem Kontextstammverzeichnis in der ursprünglichen Serviceadresse entspricht (in diesem Beispiel ist das Kontextstammverzeichnis "MyServiceWeb"), öffnen Sie den Implementierungsdeskriptor für die J2EE-Unternehmensanwendung, in dem sich dieses Webprojekt befindet, und ändern Sie das Kontextstammverzeichnis dieses Webmoduls, so dass es dem Kontextstammverzeichnis der alten Serviceadresse entspricht. Möglicherweise wird der folgende Fehler angezeigt, den Sie ignorieren können: CHKJ3017E: Webprojekt: <WEB PROJ NAME> ist einem ungültigen Kontextstammverzeichnis zugeordnet: <NEW CONTEXT ROOT> in EAR-Projekt: <APP NAME>.

Migrationsoption 2 für das IBM Web-Service-Binding (SOAP/JMS):

Die zweite Migrationsoption für das WebSphere Studio Application Developer Integration Edition SOAP/JMS-Prozessbinding ist es, Geschäftsprozesse für eine nicht-SCA-Entität (beispielsweise eine JSP oder einen Java-Client) zugänglich zu machen.

Die Standalone Reference macht eine SCA-Komponente durch einen beliebigen externen Client zugänglich. So erstellen Sie eine Standalone Reference:

1. Öffnen Sie den Assembly-Editor für das vom Migrationsassistenten erstellte Modul.
2. Erstellen Sie eine Standalone Reference für jede BPEL-Prozessschnittstelle, für die ein IBM Web Service (SOAP/JMS)-Binding in WebSphere Studio Application Developer Integration Edition generiert wurde:
 - a. Wählen Sie das Element **'Standalone References'** in der Symbolleiste.
 - b. Klicken Sie auf den Erstellungsbereich des Assembly-Editors, um eine Standalone References SCA-Entität zu erstellen.
 - c. Wählen Sie das Element **'Verbinden'** in der Symbolleiste.
 - d. Klicken Sie auf die Entität **'Standalone References'**, um sie als Quelle für die Verbindung auszuwählen.
 - e. Klicken Sie auf die Komponente **BPEL SCA**, um diese als Ziel der Verbindung auszuwählen.
 - f. Es erscheint die Warnung **'Am Quellenknoten wird ein übereinstimmender Verweis erstellt. Möchten Sie fortfahren?'**. Klicken Sie auf **OK**.
 - g. Wählen Sie die Entität **'Standalone References'**, die soeben erstellt wurde, und wählen Sie in der Ansicht 'Eigenschaften' das Inhaltsteilfenster **'Beschreibung'**.
 - h. Erweitern Sie den Link **'Verweise'** und wählen Sie die soeben erstellte Referenz. Der Name und die Beschreibung der Referenz werden aufgelistet und können auf Wunsch geändert werden.
 - i. Wenn es mehrere Schnittstellen für den Prozess gibt, wählen Sie die zu exportierende Schnittstelle(n) mit diesem Bindingtyp.
 - j. Speichern Sie das Assembly-Diagramm.

Migration des IBM Web-Service-Binding (SOAP/HTTP):

Das IBM Web-Service-Binding (SOAP/HTTP) für einen BPEL-Prozess oder einen anderen Servicetyp kann zur empfohlenen SCA-Anweisung migriert werden.

In WebSphere Studio Application Developer Integration Edition, gab dieser Bindingtyp Clients die Möglichkeit, mit einem BPEL-Prozess oder einem anderen Servicetyp durch Aufruf des IBM Web Service zu kommunizieren, wobei es sich beim Übertragungsprotokoll um HTTP handelte und die Nachricht die SOAP-Codierregeln befolgte.

Das folgende Beispiel veranschaulicht die Konventionen, die bei der Generierung eines IBM Web-Services (SOAP/HTTP) für einen 5.1-BPEL-Service zu Anwendung kommen. Der JNDI-Name des generierten IBM Web Service war eine Kombination aus BPEL-Name, Zielnamensbereich und 'Gültig-ab'-Zeitmarke, sowie dem Namen der Schnittstelle (WSDL-Porttyp, für den der Implementiercode generiert wurde). Die folgenden Attribute können beispielsweise durch Überprüfung der BPEL-Prozesseigenschaften im BPEL-Editor auf den Registerkarten 'Beschreibung' und 'Serverinhalt' gefunden werden:

Tabelle 8. Generierter Namensbereich

Prozessname	MyService
Zielnamensbereich	http://www.example.com/process87787141/
Gültig ab	Jan 01 2003 02:03:04
Schnittstelle	ProcessPortType

Der generierte Namensbereich für dieses Beispiel lautet daraufhin com/example/www/process87787141/MyService20030101T020304_ProcessPortTypePT.

In WebSphere Studio Application Developer Integration Edition gab es die folgenden Optionen zur Auswahl, wenn das IBM Web-Service-Binding (SOAP/HTTP) als Implementiertyp für einen BPEL-Prozess oder einen anderen Servicetyp ausgewählt wurde:

- Für Dokumentstil war der Standard **'RPC / andere Option: DOCUMENT'**
- Für Dokumentverwendung war der Standard **'ENCODED / andere Option: LITERAL'**

- Für Router-Adresse war der Standard `http://localhost:9080`

Eine WSDL-Datei, die das IBM Web Service SOAP/HTTP-Binding und den Service angibt, wird in den generierten Web- und EJB-Projekten, jedoch nicht im Serviceprojekt selbst erstellt. Das bedeutet, dass Sie die Datei manuell suchen und in Ihr Geschäftsintegrations-Modulprojekt kopieren müssen, wenn der IBM Web-Service-Client-Code nicht geändert werden darf. Standardmäßig wurde diese WSDL-Datei im Webprojekt unter `WebContent/WEB-INF/wsdl/<Geschäftsprozessname>_<Geschäftsprozessschnittstellen-Porttypname>_HTTP.wsdl` erstellt.

Der WSDL PortType und Nachrichten der Geschäftsprozessschnittstelle werden ebenso in diese WSDL-Datei kopiert, statt auf den vorhandenen WSDL PortType und Nachrichten, die im Serviceprojekt definiert sind, zu verweisen.

Wenn der IBM Web-Service-Client-Code nach der Migration unverändert sein soll, wird die Information in dieser Datei für die unten aufgeführten manuellen Migrationsschritte benötigt.

Es gibt zwei Optionen für die Migration des WebSphere Studio Application Developer Integration Edition SOAP/HTTP-Prozessbinding. Es muss entschieden werden, ob der Client zum SCA-Programmiermodell migriert wird oder ob er als Web-Services-Client belassen wird:

Anmerkung: Wenn die manuellen Migrationsschritte abgeschlossen sind, muss der Client ebenso zum neuen Programmiermodell migriert werden. Weitere Informationen finden Sie im entsprechenden Thema für die folgenden Client-Typen:

Tabelle 9. Weitere Informationen für die Migration von Clients

Clienttyp	Weitere Informationen finden Sie in
IBM Web-Service-Client	Migration des IBM Web-Service (SOAP/HTTP)-Client

Migrationsoption 1 für das IBM Web-Service-Binding (SOAP/HTTP):

Die erste Migrationsoption für das WebSphere Studio Application Developer Integration Edition SOAP/HTTP-Prozessbinding ist die Ermöglichung des Zugriffs auf Geschäftsprozesse durch einen Web-Services-Client.

Der Export mit Web-Service-Binding ermöglicht den Zugriff auf eine SCA-Komponente über einen externen Web-Services-Client. So erstellen Sie einen Export mit Web-Service-Binding:

1. Öffnen Sie den Assembly-Editor für das vom Migrationsassistenten erstellte Modul.
2. Erstellen Sie einen Export mit SCA-Binding für jede BPEL-Prozessschnittstelle, für die ein IBM Web-Service-Binding (SOAP/HTTP) in WebSphere Studio Application Developer Integration Edition generiert wurde, durch Klicken mit der rechten Maustaste auf die BPEL-Komponente im Assembly-Editor.
3. Wählen Sie **'Exportieren...'**.
4. Wählen Sie **'Web-Service-Binding'**.
5. Wenn es mehrere Schnittstellen für die Komponente gibt, wählen Sie die zu exportierende Schnittstelle(n) mit diesem Bindingtyp.
6. Wählen Sie die Transport-**soap/http**.
7. Wenn der Web Services Export erstellt wurde, wählen Sie den Export im Assembly-Editor und wählen Sie in der Ansicht 'Eigenschaften' das Inhaltsteilfenster **'Beschreibung'**. Der Name und die Beschreibung des Exports werden aufgelistet und können auf Wunsch geändert werden.
8. Speichern Sie das Assembly-Diagramm.
9. Führen Sie diese Schritte durch, um ein neues Web-Service-Binding und einen Service zu generieren, wenn das Beibehalten des Client-Codes gewünscht ist:

- a. Kopieren Sie die 5.1 WSDL-Datei aus dem generierten 5.1 EJB-Projekt unter `ejbModule/META-INF/wsdl/Geschäftsprozessname/Geschäftsprozessschnittstellen-Porttypname_HTTP.wsdl` in das Geschäftsintegrations-Modulprojekt.
- b. Nach dem Kopieren der Datei und der Neuerstellung des Moduls werden möglicherweise Fehlermeldungen angezeigt, da die vom Web-Service verwendeten XML-Schematypen, WSDL-Nachrichten und WSDL-Porttypen in der WSDL-Datei des IBM Web-Services in 5.1 doppelt vorhanden sind. Um dies zu beheben, löschen Sie diese doppelt vorhandenen Definitionen aus der Binding/Service-WSDL des IBM Web-Services und fügen Sie an deren Stelle einen WSDL-Import für die echte Schnittstellen-WSDL hinzu. **Hinweis:** Beachten Sie, dass die Schemendefinitionen in bestimmten Fällen geändert wurden, als der IBM Web-Service-Implementierungscode von WebSphere Studio Application Developer Integration Edition generiert wurde. Dies führt unter Umständen zu Inkonsistenzen für vorhandene Clients, die die IBM Web-Service- WSDL verwenden. Das Schemaattribut "elementFormDefault" beispielsweise wurde in dem Inline-Schema, das in der IBM Web-Service-WSDL generiert wurde, auf "qualifiziert" gesetzt, auch wenn die ursprüngliche Schemadefinition nicht qualifiziert war. Dadurch wurde der folgende Fehler während der Laufzeit generiert: WWS3047E: Fehler: Element kann nicht entrealisiert werden.
- c. Klicken Sie mit der rechten Maustaste auf diese WSDL-Datei, die Sie soeben in das Geschäftsintegrationsmodul kopiert haben, und wählen Sie **'Öffnen mit' und 'WSDL-Editor'**.
- d. Gehen Sie zur Registerkarte 'Quelle'. Löschen Sie alle in dieser Datei definierten WSDL PortTypes und Nachrichten.
- e. Daraufhin wird folgender Fehler angezeigt: Der für das '`<binding>`' Binding angegebene '`<portType>`' Porttyp ist nicht definiert. Um dies zu beheben, klicken Sie im WSDL-Editor in der Diagrammsicht mit der rechten Maustaste auf den Importabschnitt und wählen Sie **'Import hinzufügen'**.
- f. Klicken Sie in der Ansicht 'Eigenschaften' in der Registerkarte 'Allgemein' auf die Schaltfläche ... rechts neben dem Feld für die Speicherposition. Blättern Sie zur Schnittstellen-WSDL, in der sich die WSDL-Nachricht und die Porttypdefinitionen befinden und klicken Sie auf **OK**, um die Schnittstellen-WSDL in die Service/Binding-WSDL zu kopieren.
- g. Speichern Sie die WSDL-Datei.
- h. Aktualisieren/erstellen Sie das Projekt neu. Wechseln Sie in die Geschäftsintegrationsperspektive. Öffnen Sie das Assembly-Diagramm des Moduls im Assembly-Editor.
- i. Erweitern Sie das zu migrierende Modul in der Ansicht 'Projektexplorer', und erweitern Sie die logische Kategorie **Web-Service-Ports**. Der in der Binding/Service-WSDL vorhandene Port sollte nun angezeigt werden. Ziehen und übergeben Sie ihn an den Assembly-Editor.
- j. Wählen Sie die Erstellung eines **Exports mit Web-Service-Binding** und wählen Sie den entsprechenden Portnamen. Dadurch wird der Export erstellt, der das alte Binding/Service verwendet, sodass vorhandene Web-Service-Clients nicht geändert werden müssen. Wenn Sie den Export auswählen, den Sie soeben im Assembly-Editor erstellt haben und zur Ansicht 'Eigenschaften' gehen, wird in der Registerkarte 'Binding' angezeigt, dass die 5.1 Port- und Servicenamen für Sie eingetragen wurden.
- k. Speichern Sie alle Änderungen.
- l. Kurz vor der Implementierung der Anwendung können Sie die Konfiguration des generierten Web-Projekts ändern, so dass Sie der 5.1 Serviceadresse entspricht (Sie müssen diese Änderungen jedes Mal durchführen, wenn Sie Änderungen am SCA-Modul vornehmen, durch die diese Datei erneut generiert wird). Wenn Sie die Servicedefinition der IBM Web-Service-WSDL, die Sie aus 5.1 wieder verwenden, anzeigen, wird die Serviceadresse angezeigt, zu der der 5.1-Client codiert ist:
`<wsdl:soap:address location="http://localhost:9080/MyServiceWeb/services/MyServicePort"/>`
- m. Damit die generierten 6.0 Webprojekt-Artefakte mit dieser alten Serviceadresse übereinstimmen, sollten Sie den generierten Implementierungsdeskriptor des Webprojekts ändern. Öffnen Sie den Implementierungsdeskriptor in WebSphere Integration Developer und fügen Sie in der Registerkarte 'Servlets' eine zusätzliche URL-Zuordnung hinzu, die der vorhandenen URL-Zuordnung für diesen Export ähnelt, mit dem gleichen Servlet-Namen aber einem anderen URL-Muster.

- n. Wenn Sie außerdem das Kontextstammverzeichnis dieses Web-Projekts ändern müssen, so dass es dem Kontextstammverzeichnis in der ursprünglichen Serviceadresse entspricht (in diesem Beispiel ist das Kontextstammverzeichnis "MyServiceWeb"), öffnen Sie den Implementierungsdeskriptor für die J2EE-Unternehmensanwendung, in dem sich dieses Webprojekt befindet, und ändern Sie das Kontextstammverzeichnis dieses Webmoduls, so dass es dem Kontextstammverzeichnis der alten Serviceadresse entspricht. Möglicherweise wird der folgende Fehler angezeigt, den Sie ignorieren können: CHKJ3017E: Webprojekt: <WEB PROJ NAME> ist einem ungültigen Kontextstammverzeichnis zugeordnet: <NEW CONTEXT ROOT> in EAR-Projekt: <APP NAME>.

Migrationsoption 2 für das IBM Web-Service-Binding (SOAP/HTTP):

Die zweite Migrationsoption für das WebSphere Studio Application Developer Integration Edition SOAP/HTTP-Prozessbinding ist es, Geschäftsprozesse für eine nicht-SCA-Entität (beispielsweise eine JSP oder einen Java-Client) zugänglich zu machen.

Die Standalone Reference macht eine SCA-Komponente durch einen beliebigen externen Client zugänglich. So erstellen Sie eine Standalone Reference:

1. Öffnen Sie den Assembly-Editor für das vom Migrationsassistenten erstellte Modul.
2. Erstellen Sie eine Standalone Reference für jede Schnittstelle, für die ein IBM Web Service (SOAP/HTTP)-Binding in WebSphere Studio Application Developer Integration Edition generiert wurde:
 - a. Wählen Sie das Element '**Standalone References**' in der Symbolleiste.
 - b. Klicken Sie auf den Erstellungsbereich des Assembly-Editors, um eine Standalone References SCA-Entität zu erstellen.
 - c. Wählen Sie das Element '**Verbinden**' in der Symbolleiste.
 - d. Klicken Sie auf die Entität '**Standalone References**', um sie als Quelle für die Verbindung auszuwählen.
 - e. Klicken Sie auf die Komponente **SCA**, um diese als Ziel der Verbindung auszuwählen.
 - f. Es erscheint die Warnung: **Am Quellenknoten wird ein übereinstimmender Verweis erstellt. Möchten Sie fortfahren?**. Klicken Sie auf **OK**.
 - g. Wählen Sie die Entität '**Standalone References**', die soeben erstellt wurde, und wählen Sie in der Ansicht 'Eigenschaften' das Inhaltsteilfenster '**Beschreibung**'.
 - h. Erweitern Sie den Link '**Verweise**' und wählen Sie die soeben erstellte Referenz. Der Name und die Beschreibung der Referenz werden aufgelistet und können auf Wunsch geändert werden.
 - i. Wenn es mehrere Schnittstellen für den Prozess gibt, wählen Sie die zu exportierende Schnittstelle(n) mit diesem Bindingtyp.
 - j. Speichern Sie das Assembly-Diagramm.

Migration des Apache Web-Service-Binding (SOAP/HTTP):

Das Apache Web-Service-Binding (SOAP/HTTP) für einen BPEL-Prozess oder einen anderen Servicetyp kann zur empfohlenen SCA-Anweisung migriert werden.

In WebSphere Studio Application Developer Integration Edition hat dieser Bindingtyp Clients die Kommunikation mit einem BPEL-Prozess oder einem anderen Servicetyp durch Aufruf eines Apache Web-Service ermöglicht.

In WebSphere Studio Application Developer Integration Edition gab es die folgenden Optionen zur Auswahl, wenn das Apache Web-Service-Binding als Implementiertyp für einen BPEL-Prozess oder einen anderen Servicetyp ausgewählt wurde:

- Für die Dokumentdarstellung war dies **RPC** (keine sonstige Option verfügbar)
- Für SOAP Aktion war dies **URN:WSDL PortType name**
- Für Adresse war dies **http://localhost:9080/Service Project NameWeb/servlet/rpcrouter**

- Für 'Codierung verwenden' war der Standard 'Ja' (Bei 'Ja' wurde die Codierungsdarstellung auf: <http://schemas.xmlsoap.org/soap/encoding/> eingestellt)

Eine WSDL-Datei, die das Apache SOAP-Binding und den Service angibt, wird im Serviceprojekt erstellt. Standardmäßig wird sie in demselben Verzeichnis wie der Service erstellt, der mit dem Namen `<Geschäftsprozessname>_<Geschäftsprozessschnittstellen-Porttypname>_SOAP.wsdl` eingeschlossen wird. Der WSDL PortType und Nachrichten der Geschäftsprozessschnittstelle werden direkt von diesem Binding und Service verwendet. Nach der Migration sollten Sie diese WSDL für *nichts* anderes außer möglicherweise für denselben Namensbereich, Port und Servicenamen in der neuen WSDL verwenden, die für Sie in Version 6 erstellt wird.

Es gibt zwei Optionen für die Migration des WebSphere Studio Application Developer Integration Edition Web Service-Prozessbinding. Es muss entschieden werden, ob der Client zum SCA-Programmiermodell migriert wird oder ob er als IBM Web-Services-Programmiermodell belassen wird. Es gibt kein Binding mehr, das dem Apache Web Service (SOAP/HTTP) -Bindingtyp im 6 SCA-Programmiermodell entspricht.

Migrieren Sie diesen Apache Web Service zur Verwendung der IBM Web-Service-Engine. Weitere Informationen zur Durchführung dieser Migration und Erstellung eines IBM Web-Service (SOAP/HTTP) finden Sie im Thema "Migration des IBM Web Service-Binding (SOAP/HTTP)".

Migration zum SCA-Programmiermodell:

Dieser Abschnitt bezieht sich auf jeden unformatierten Java-Code, der mit einem WebSphere Studio Application Developer Integration Edition-Service im Dialog steht, und veranschaulicht die Migration vom WSIF Programmiermodell zum neuen SCA Programmiermodell, wobei der Datenfluss durch die Anwendung in Eclipse Service Data Objects (SDOs) gespeichert wird. Dieser Abschnitt beschreibt außerdem, wie Sie die gängigsten Client-Typen manuell zum neuen Programmiermodell migrieren.

Im Falle von BPEL Prozessen, die Java Snippets enthalten, veranschaulicht dieser Abschnitt die Migration von der alten Java Snippet API zur neuen Java Snippet API, wobei der Datenfluss durch die Anwendung in Eclipse Service Data Objects (SDOs) gespeichert wird. Wenn möglich, werden die Snippets automatisch durch den Migrationsassistenten migriert, es gibt jedoch Snippets, die vom Migrationsassistenten nicht vollständig migriert werden können, was bedeutet, dass manuelle Schritte für den Abschluss der Migration erforderlich sind.

Im Folgenden finden Sie eine Zusammenfassung der Änderungen des Programmiermodells:

V5.1 Programmiermodell

1. WSIF- und WSDL-basiert
2. Generierte Proxys für Services
3. Beans und Formatsteuerrouniten für Typen

V6.0 Programmiermodell (Java-zentrierter)

1. SCA-Services basierend auf SDOs mit Doclet-Tags
2. Schnittstellenbindings für Services
3. SDOs und Datenbindings für Typen

Migration von WSIFMessage API-Aufrufen zu SDO APIs:

In dem folgenden Abschnitt wird die Migration vom alten WebSphere Business Integration Server Foundation Version 5.1 Programmiermodell, in dem der Datenfluss durch die Anwendung als WSIFMessage-Objekte mit einer generierten Schnittstelle, die getypt (strongly typed) war, dargestellt wird, zum neuen Programmiermodell von WebSphere Process Server Version 6.0 beschrieben, in dem die Daten als Service Data Objects (SDOs) dargestellt werden und keine getypte Schnittstelle generiert wird.

Tabelle 10. Änderungen und Lösungen der Migration von WSIFMessage API-Aufrufen zu SDO APIs

Änderung	Lösung
WSIFMessage-basierte Wrapper-Klassen werden nicht länger für WSDL-Nachrichtentypen generiert, und auch die Java-Bean Helper-Klassen werden nicht mehr für komplexe Schementypen generiert.	<p>Beim Schreiben von Code, der mit SCA-Services interagiert, müssen die generischen SDO APIs für die Manipulation der <code>commonj.sdo.DataObject</code>-Nachrichten verwendet werden, die die durch die Anwendung fließenden Daten enthalten.</p> <p>WSDL-Nachrichtendefinitionen, die über einen einzelnen einfachgetypten Abschnitt verfügen, werden nun durch einen einfachen Java-Typ dargestellt, der den Abschnitt direkt darstellt, statt die tatsächlichen Daten in einen Wrapper einzuschließen. Wenn es sich beim einzelnen Nachrichtenabschnitt um einen komplexen Typ handelt, werden die Daten als ein <code>DataObject</code> dargestellt, das die komplexe Typdefinition befolgt.</p> <p>WSDL-Nachrichtendefinitionen, die über mehrere Abschnitte verfügen, entsprechen nun einem <code>DataObject</code>, das über Eigenschaften für alle Nachrichtenabschnitte verfügt, wobei <code>complexType</code> als 'Referenztyp'-Eigenschaften des übergeordneten <code>DataObject</code> dargestellt werden, auf die über <code>getDataObject</code>- und <code>setDataObject</code>-Methoden zugegriffen werden kann.</p>
Getypte Getter-Methoden für WSIFMessage-Abschnitte und generierte Java-Beans sollten nicht verwendet werden.	Ungetypte SDO APIs sollten für den Abruf von <code>DataObject</code> -Eigenschaften verwendet werden.
Getypte Setter-Methoden für Nachrichtenabschnitte von BPEL-Variablen sind nicht länger verfügbar.	Ungetypte SDO APIs müssen für die Einrichtung der <code>DataObject</code> -Eigenschaften verwendet werden.
Ungetypte Getter-Methoden für WSIFMessage-Eigenschaften sollten nicht länger verwendet werden.	Ungetypte SDO APIs müssen für die Einrichtung der <code>DataObject</code> -Eigenschaften verwendet werden.
Ungetypte Setter-Methoden für WSIFMessage-Eigenschaften sollten nicht länger verwendet werden.	Ungetypte SDO APIs müssen für die Einrichtung der <code>DataObject</code> -Eigenschaften verwendet werden.
Alle WSIFMessage API-Aufrufe sollten wenn möglich zur SDO API migriert werden.	Migrieren Sie den Aufruf wenn möglich zu einem äquivalenten SDO API-Aufruf. Überarbeiten Sie die Logik, wenn dies nicht möglich ist.

Migration des WebSphere Business Integration Server Foundation Client-Codes:

Dieser Abschnitt beschreibt die Migration verschiedener Client-Typen, die für die WebSphere Business Integration Server Foundation 5.1-Servicetypen möglich waren.

Migration des EJB Client:

Dieses Thema erläutert die Migration von Clients, die eine EJB-Schnittstelle verwenden, um einen Service aufzurufen.

1. Ziehen und übergeben Sie den Export (mit Drag and Drop) mit SCA Binding vom migrierten Modell zum Assembly-Editor dieses neuen Moduls. Dadurch wird ein Import mit SCA Binding erstellt. Damit ein Client einen Verweis zu diesem Import abrufen kann, muss ein Standalone-Verweis erstellt werden.
2. Wählen Sie auf der Palette den Standalone-Verweis. Klicken Sie einmal auf den Assembly Editor-Erstellungsbereich, um einen Standalone-Verweis für dieses neue Modul zu erstellen.
3. Wählen Sie das Verbindungstool, klicken Sie auf den Serviceverweis und daraufhin auf 'Importieren'.

4. Klicken Sie auf **OK**, wenn Sie gewarnt werden, dass ein übereinstimmender Verweis am Querknoten erstellt wird.
5. Sie werden gefragt: **‘Es ist für einen Java-Client einfacher, eine Java-Schnittstelle mit diesem Verweis zu verwenden – möchten Sie den WSDL-Verweis in einen kompatiblen Java-Verweis umwandeln?’**:
 - a. Beantworten Sie diese Frage mit **‘Ja’**, wenn dieser Client diesen Service abrufen soll und ihn als Java-Klasse umsetzen soll, um ihn mit einer Java-Schnittstelle aufzurufen. Diese neue Java-Schnittstelle übernimmt ihren Namen vom WSDL PortType, wobei das Paket der Schnittstelle vom Namensbereich des WSDL PortType hergeleitet wird. Für jede auf dem WSDL PortType definierte Operation gibt es eine definierte Methode, und jeder WSDL-Nachrichtenabschnitt wird als Argument zu den Schnittstellenmethoden dargestellt.
 - b. Beantworten Sie diese Frage mit **‘Nein’**, wenn der Client diesen Service abrufen und die generische `com.ibm.websphere.sca.Service-Schnittstelle` verwenden soll, um ihn mit der Aufrufoperation als einen generischen SCA-Service aufzurufen.
6. Benennen Sie den Standalone-Verweis gegebenenfalls in einen aussagekräftigeren Namen um, indem Sie die Standalone-Verweiskomponente im Assembly-Editor auswählen. Wählen Sie in der Sicht ‘Eigenschaften’ die Registerkarte ‘Details’, suchen Sie den Verweis, der soeben erstellt wurde, und ändern Sie den Namen. Notieren Sie sich den gewählten Namen für diesen Verweis, da der Client diesen Namen für den Aufruf der `locateService`-Methode der `com.ibm.websphere.sca.ServiceManager`-Instanz verwenden muss.
7. Klicken Sie auf **‘Speichern’**, um das Assemblydiagramm zu speichern.

Dieses neue Modul muss auf dem lokalen Klassenpfad des Clients vorhanden sein, damit er auf das migrierte EJB-Modul, das auf dem Server ausgeführt wird, zugreifen kann.

Im Folgenden finden Sie ein Beispiel eines Client-Codes für einen Service des Typs “CustomerInfo”:

```
// Create a new ServiceManager
ServiceManager serviceManager = ServiceManager.INSTANCE;
// Locate the CustomerInfo service
CustomerInfo customerInfoService = (CustomerInfo) serviceManager.locateService ("<name-of-standalone-reference-from-prev
// Invoke the CustomerInfo service
System.out.println(" [getMyValue] getting customer info...");
DataObject customer = customerInfoService.getCustomerInfo(customerID);
```

Der Client muss die Konstruktion der Nachricht ändern. Die Nachrichten basierten zuvor auf der WSIF-Message-Klasse, sollten nun jedoch auf der `commonj.sdo.DataObject`-Klasse basieren.

Migration des EJB-Prozessbinding-Client:

Dieses Thema erläutert die Migration von Clients, die das WSIF-EJB-Prozessbinding verwenden, um auf einen BPEL-Service zuzugreifen.

Clients, die das EJB-Prozessbinding für den Aufruf eines Geschäftsprozesses verwendet haben, sollten nun entweder die SCA API für den Aufruf des Services (der migrierte Geschäftsprozess muss über einen Export mit SCA-Binding verfügen) oder die IBM Web Service Client API verwenden, um den Service aufzurufen (der migrierte Geschäftsprozess muss über einen Export mit Web-Service-Binding verfügen).

Weitere Informationen zur Generierung solcher Clients finden Sie in den Themen “Migration des EJB-Clients”, “IBM Web-Service-Client (SOAP/JMS)” oder “IBM Web-Service-Client (SOAP/HTTP)”.

Migration des IBM Web Service (SOAP/JMS) Client:

Dieses Thema erläutert die Migration von Clients, die Web-Service-APIs (SOAP/JMS) verwenden, um einen Service aufzurufen.

Für vorhandene Clients ist während der Migration keine Migration erforderlich. Beachten Sie, dass Sie das generierte Webprojekt manuell ändern müssen (erstellen Sie eine neue servlet-Zuordnung) und manchmal das Kontextstammverzeichnis des Webprojekts im Implementierungsdeskriptor der Unternehmensanwendung, um den Service mit der gleichen Adresse zu veröffentlichen, unter der er in WebSphere Business Integration Server Foundation veröffentlicht war. Weitere Informationen enthält das Thema "Migration des IBM Web-Service-Binding (SOAP/JMS)".

Beachten Sie, dass im Vergleich zu 5.1, in der ein WSIF- oder RPC-Client generiert werden konnte, die Tools in 6.0 nur RPC Client-Erzeugung unterstützen, da RPC die bevorzugte API in 6.0 gegenüber der WSIF API ist.

Hinweis: Zur Erzeugung einer neuen Client Proxy aus WebSphere Integration Developer muss WebSphere Process Server oder WebSphere Application Server installiert sein.

1. Vergewissern Sie sich, dass WebSphere Process Server oder WebSphere Application Server installiert ist.
2. Suchen Sie in der Perspektive 'Ressourcen' oder Java nach der WSDL-Datei, die dem **Export mit Web-Service-Binding** entspricht, klicken Sie mit der rechten Maustaste und wählen Sie '**Web Services**' → '**Client generieren**' (Beachten Sie, dass dieser Assistent dem Assistenten in 5.1 sehr ähnlich ist).
3. Wählen Sie für den Client-Proxytyp '**Java-Proxy**' und klicken Sie auf '**Weiter**'.
4. Die Position der WSDL sollte angegeben sein. Klicken Sie auf '**Weiter**'.
5. Als nächstes müssen Sie die entsprechenden Optionen auswählen, um die Konfiguration Ihrer Client-Umgebung anzugeben, einschließlich der Laufzeit des Web-Services, Server,J2EE Version, Client-Typ (Java, EJB, Web, Anwendungs-Client). Klicken Sie auf '**Weiter**'.
6. Führen Sie die verbleibenden Schritte durch, um den Client-Proxy zu generieren.

Migration des IBM Web Service (SOAP/HTTP) Client:

Dieses Thema erläutert die Migration von Clients, die Web-Service-APIs (SOAP/HTTP) verwenden, um einen Service aufzurufen.

Für vorhandene Clients ist während der Migration keine Migration erforderlich. Beachten Sie, dass Sie das generierte Webprojekt manuell ändern müssen (erstellen Sie eine neue servlet-Zuordnung) und manchmal das Kontextstammverzeichnis des Webprojekts im Implementierungsdeskriptor der Unternehmensanwendung, um den Service mit der gleichen Adresse zu veröffentlichen, unter der er in WebSphere Business Integration Server Foundation veröffentlicht war. Weitere Informationen enthält das Thema "Migration des IBM Web-Service-Binding (SOAP/HTTP)".

Wenn Änderungen von Konstruktionen aufgetreten sind und Sie einen neuen Client-Proxy generieren möchten, befolgen Sie die folgenden Schritte. Beachten Sie, dass im Vergleich zu 5.1, in der ein WSIF- oder RPC-Client generiert werden konnte, die Tools in 6.0 nur RPC Client-Erzeugung unterstützen, da RPC die bevorzugte API in 6.0 gegenüber der WSIF API ist.

Hinweis: Zur Erzeugung einer neuen Client Proxy aus WebSphere Integration Developer muss WebSphere Process Server oder WebSphere Application Server installiert sein.

1. Vergewissern Sie sich, dass WebSphere Process Server oder WebSphere Application Server installiert ist.
2. Wählen Sie die WSDL-Datei, die dem **Export mit Web-Service-Binding** entspricht, klicken Sie mit der rechten Maustaste und wählen Sie '**Web Services**' → '**Client generieren**' (Beachten Sie, dass dieser Assistent dem Assistenten in 5.1 sehr ähnlich ist).
3. Wählen Sie für den Client-Proxytyp '**Java-Proxy**' und klicken Sie auf '**Weiter**'.
4. Die Position der WSDL sollte angegeben sein. Klicken Sie auf '**Weiter**'.

5. Als nächstes müssen Sie die entsprechenden Optionen auswählen, um die Konfiguration Ihrer Client-Umgebung anzugeben, einschließlich der Laufzeit des Web-Services, Server,J2EE Version, Client-Typ (Java, EJB, Web, Anwendungs-Client). Klicken Sie auf **'Weiter'**.
6. Führen Sie die verbleibenden Schritte durch, um den Client-Proxy zu generieren.

Migration des Apache Web Service (SOAP/HTTP) Client:

Die Apache Web Service Client APIs sind nicht für den Aufruf eines WebSphere Integration Developer Service geeignet. Client-Code muss für die Verwendung der IBM Web Service (SOAP/HTTP) Client APIs migriert werden.

Weitere Informationen finden Sie im Abschnitt "IBM Web-Service-Client (SOAP/HTTP)".

Wenn ein Client-Proxy in 5.1 automatisch generiert wurde, hat dieser Proxy zur Interaktion mit dem Service WSIF APIs verwendet. In 6.0 unterstützen die Tools nur RPC Client- Erzeugung, da RPC die bevorzugte API in 6.0 gegenüber der WSIF API ist.

Hinweis: Zur Erzeugung einer neuen Client Proxy aus WebSphere Integration Developer muss WebSphere Process Server oder WebSphere Application Server installiert sein.

1. Vergewissern Sie sich, dass WebSphere Process Server oder WebSphere Application Server installiert ist.
2. Wählen Sie die WSDL-Datei, die dem **Export mit Web-Service-Binding** entspricht, klicken Sie mit der rechten Maustaste und wählen Sie **'Web Services'** → **'Client generieren'** (Beachten Sie, dass dieser Assistent dem Assistenten in 5.1 sehr ähnlich ist).
3. Wählen Sie für den Client-Proxytyp **'Java-Proxy'** und klicken Sie auf **'Weiter'**.
4. Die Position der WSDL sollte angegeben sein. Klicken Sie auf **'Weiter'**.
5. Als nächstes müssen Sie die entsprechenden Optionen auswählen, um die Konfiguration Ihrer Client-Umgebung anzugeben, einschließlich der Laufzeit des Web-Services, Server,J2EE Version, Client-Typ (Java, EJB, Web, Anwendungs-Client). Klicken Sie auf **'Weiter'**.
6. Führen Sie die verbleibenden Schritte durch, um den Client-Proxy zu generieren.

Migration des JMS Client:

Clients, die mit einem 5.1-Service über die JMS-API (durch das Senden einer JMS-Nachricht an eine Warteschlange) kommunizieren, müssen unter Umständen manuell migriert werden. Dieses Thema erläutert die Migration von Clients, die JMS-APIs verwenden (eine JMS-Nachricht an eine Warteschlange senden), um einen Service aufzurufen.

Sie müssen sicherstellen, dass der in einem früheren Schritt erstellte **Export mit JMS-Binding** diese Text- oder Objektnachricht unverändert akzeptieren kann. Möglicherweise müssen Sie zu diesem Zweck ein angepasstes Datenbinding schreiben. Weitere Informationen hierzu finden Sie unter "Migration von JMS und JMS-Prozessbindings".

Der Client muss die Konstruktion der Nachricht ändern. Die Nachrichten basierten zuvor auf der WSIF-Message-Klasse, sollten nun jedoch auf der commonj.sdo.DataObject-Klasse basieren. Weitere Informationen zur Durchführung dieser manuellen Migration finden Sie im Abschnitt "Migration von WSIFMessage API-Aufrufen zu SDO APIs".

Migration des generischen Geschäftsprozess-Choreographer EJB API Client:

Dieses Thema erläutert die Migration von Clients, die die generische EJB-API des Geschäftsprozess-Choreographers der Version 5.1 verwenden, um einen BPEL-Service aufzurufen.

Es gibt eine neue Version der generischen EJB API, die DataObjects als Nachrichtenformat verwendet. Der Client muss die Konstruktion der Nachricht ändern. Die Nachrichten basierten zuvor auf der WSIF-Message-Klasse, sollten nun jedoch auf der commonj.sdo.DataObject-Klasse basieren. Beachten Sie, dass die generische EJB API sich nicht erheblich geändert hat, da ClientObjectWrapper nach wie vor eine Nachrichtenumlaufprogramm für das entsprechende Nachrichtenformat bereitstellt.

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

Der JNDI-Name der alten generischen EJB, die WSIFMessage-Objekte akzeptiert, ist:

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

Es gibt zwei generische EJBs in 6.0, da die Human Task-Operationen nun als separate EJB verfügbar sind. Die 6.0 JNDI-Namen dieser generischen EJBs lauten:

```
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
```

Migration des generischen Geschäftsprozess-Choreographer Messaging API Client und des JMS-Prozessbinding-Client:

In WebSphere Process Server 6.0 gibt es keine generische Nachrichtenübertragungs-API. Anhand der Informationen unter "Migration von JMS und JMS-Prozessbinding" können Sie eine andere Methode auswählen, um den Geschäftsprozess für Kunden zugänglich zu machen und den Client gemäß dem ausgewählten Binding erneut zu schreiben.

Migration des Geschäftsprozess-Choreographer-Web-Client:

Dieses Thema erläutert, wie Sie die Web-Client-Einstellungen und angepassten JSPs des Prozess-Choreographers aus Version 5.1 migrieren.

Der Migrationsassistent behält die Web-Client-Einstellungen aus 5.1 bei. Diese Einstellungen können im Human Task-Editor nicht bearbeitet werden. Sie sollten mit WebSphere Integration Developer 6.0 neue Web-Client-Einstellungen und JSPs erstellen.

Migration von Web-Client-Änderungen

In 5.1 konnten Sie Darstellung und Funktionsweise des Struts-basierten Web-Clients durch Änderung seiner JSP-Datei **Header.jsp** und des Style-Sheets **dwc.css** ändern.

Da der 6.0 Web-Client (umbenannt in **Business Process Choreographer Explorer**) auf Java Server Faces (JSF) anstelle von Struts basiert, ist die automatische Migration von Web-Client-Änderungen nicht möglich. Detaillierte Informationen zur Anpassung der Version 6.0 dieser Anwendung finden Sie in der "Business Process Choreographer Explorer"-Dokumentation.

Benutzerdefinierte JSPs können für Geschäftsprozesse und Mitarbeiteraktivitäten definiert werden. Der Web-Client verwendet diese JSPs zur Anzeige von Eingabe- und Ausgabenachrichten für den Prozess und Aktivitäten.

Diese JSPs sind insbesondere in folgenden Fällen hilfreich:

1. Nachrichten verfügen über nicht-primitive Abschnitte, um die Benutzerfreundlichkeit der Datenstruktur der Nachrichten zu verbessern.
2. Sie möchten die Funktionalität des Web-Clients erweitern.

Es stehen mehrere unterschiedliche Optionen für die Angabe von Web-Client-Einstellungen für einen 6.0 Prozess zur Verfügung, daher müssen Sie WebSphere Integration Developer verwenden, um die Web-Client-Einstellungen für migrierte Prozesse und Aktivitäten zu überarbeiten:

1. Wählen Sie den Prozesserstellungsbereich oder eine Aktivität im Prozess.
2. Wählen Sie in der Ansicht 'Eigenschaften' die Registerkarte '**Client**', um die Web-Client-Einstellungen zu überarbeiten.
3. Migrieren Sie alle benutzerdefinierten JSPs manuell:
 - a. Weitere Informationen zu Programmiermodelländerungen finden Sie im Abschnitt "Migration zum SCA-Programmiermodell".
 - b. Der Web-Client verwendet die generischen APIs für die Interaktion mit Geschäftsprozessen. In diesen Abschnitten finden Sie Informationen zur Migration von Aufrufen zu diesen generischen APIs.
4. Geben Sie den Namen der neuen JSP in den 6.0 Web-Client-Einstellungen für den Prozess an

Anmerkung: Die Zuordnung von JSPs ist mit dem 6.0 Business Process Choreographer Explorer nicht notwendig, da DataObjects keine benutzerdefinierte Zuordnung erfordern.

Migration von WebSphere Business Integration Server Foundation BPEL Java-Snippets:

Im Falle von BPEL Prozessen, die Java Snippets enthalten, veranschaulicht dieser Abschnitt die Migration von der alten Java Snippet API zur neuen Java Snippet API, wobei der Datenfluss durch die Anwendung als Eclipse Service Data Objects (SDOs) gespeichert wird.

Informationen zur Durchführung von Migrationsschritten, die spezifisch sind für den Übergang von WSIFMessage zu SDO finden Sie im Abschnitt "Migration von WSIFMessage API-Aufrufen zu SDO APIs".

Wenn möglich, werden die Snippets automatisch vom Migrationsassistenten migriert, es gibt jedoch Snippets, die vom Migrationsassistenten nicht vollständig migriert werden können. Hierfür werden zusätzliche manuelle Schritte für den Abschluss des Migrationsvorgangs benötigt. Informationen zu den Java-Snippet-Typen, die manuell migriert werden müssen, finden Sie im Thema 'Einschränkungen'. Wann immer eines dieser Snippets angetroffen wird, gibt der Migrationsassistent eine Erklärung dazu ab, weshalb es nicht automatisch migriert werden kann und gibt eine Warnung oder Fehlermeldung aus.

In der folgenden Tabelle werden die Änderungen des BPEL Java-Snippet-Programmiermodells und der API von Process Choreographer Version 5.1 zu 6.0 aufgelistet:

Tabelle 11. Änderungen und Lösungen für die Migration von WebSphere Business Integration Server Foundation BPEL Java-Snippets

Änderung	Lösung
<p>WSIFMessage-basierte Wrapper-Klassen werden nicht länger für WSDL-Nachrichtentypen generiert, und auch die Java-Bean Helper-Klassen werden nicht mehr für komplexe Schementypen generiert.</p>	<p>Auf BPEL-Variablen kann direkt über den Namen zugegriffen werden. Beachten Sie, dass BPEL-Variablen, deren WSDL-Nachrichtendefinition über einen einzelnen Abschnitt verfügt, diesen Abschnitt nun direkt darstellen, statt mit den tatsächlichen Daten in einen Wrapper eingeschlossen. Bei Variablen, deren Nachrichtentyp über mehrere Abschnitte verfügt, sind die Abschnitte in einen DataObject-Wrapper eingeschlossen (während es sich beim Wrapper in WebSphere Application Developer Integration Edition um eine WSIFMessage gehandelt hat).</p> <p>Da BPEL-Variablen direkt in 6.0-Snippets verwendet werden können, besteht weniger Bedarf an lokalen Variablen als in 5.1.</p> <p>Die getypten (strongly-typed) Getter für die BPEL-Variablen haben den Einschluss der Nachrichtenabschnitte mit dem Wrapper-Objekt implizit initialisiert. Es gibt kein 'Wrapper'-Objekt für BPEL-Variablen, deren WSDL-Nachrichtendefinition nur über einen einzelnen Abschnitt verfügt: in diesem Fall stellen die BPEL-Variablen den Abschnitt direkt dar (In dem Fall, in dem es sich beim einzelnen Abschnitt um einen einfachen XSD-Typ handelt, wird die BPEL-Variable als Java-Objekt-Wrapper-Typ wie beispielsweise <code>java.lang.String</code>, <code>java.lang.Integer</code>, etc. dargestellt). BPEL-Variablen mit WSDL-Nachrichtendefinitionen mit mehreren Abschnitten werden anders gehandhabt: die Abschnitte sind nach wie vor in einen Wrapper eingeschlossen und dieser DataObject-Wrapper muss explizit im 6.0 Java-Snippetcode initialisiert werden, wenn er noch nicht durch eine vorherige Operation eingerichtet wurde.</p> <p>Wenn irgendwelche der lokalen Variablen aus den 5.1-Snippets über denselben Namen wie die BPEL-Variable verfügen, kann es zu Konflikten kommen, versuchen Sie daher, diese Situation wenn möglich zu lösen.</p>
<p>WSIFMessage-Objekte werden nicht länger für die Darstellung von BPEL-Variablen verwendet.</p>	<p>Wenn irgendwelche der benutzerdefinierten Java-Klassen, die aus den Java-Snippets aufgerufen werden, über einen WSIFMessage-Parameter verfügen, muss dieser migriert werden, so dass er ein DataObject akzeptiert/ablehnt.</p>
<p>Getypte Getter-Methoden für BPEL-Variablen sind nicht länger verfügbar.</p>	<p>Auf die Variablen kann direkt über den Namen zugegriffen werden. Beachten Sie, dass BPEL-Variablen, deren WSDL-Nachrichtendefinition über einen einzelnen Abschnitt verfügt, diesen Abschnitt nun direkt darstellen, statt mit den tatsächlichen Daten in einen Wrapper eingeschlossen. Bei Variablen, deren Nachrichtentyp über mehrere Abschnitte verfügt, sind die Abschnitte in einen DataObject-Wrapper eingeschlossen (während es sich beim Wrapper in WebSphere Application Developer Integration Edition um eine WSIFMessage gehandelt hat).</p>

Tabelle 11. Änderungen und Lösungen für die Migration von WebSphere Business Integration Server Foundation BPEL Java-Snippets (Forts.)

Änderung	Lösung
Gettype Setter-Methoden für BPEL-Variablen sind nicht länger verfügbar.	Auf die Variablen kann direkt über den Namen zugegriffen werden. Beachten Sie, dass BPEL-Variablen, deren WSDL-Nachrichtendefinition über einen einzelnen Abschnitt verfügt, diesen Abschnitt nun direkt darstellen, statt mit den tatsächlichen Daten in einen Wrapper eingeschlossen. Bei Variablen, deren Nachrichtentyp über mehrere Abschnitte verfügt, sind die Abschnitte in einen DataObject-Wrapper eingeschlossen (während es sich beim Wrapper in WebSphere Application Developer Integration Edition um eine WSIFMessage gehandelt hat).
Ungettype Getter-Methoden für BPEL-Variablen, die eine WSIFMessage zurückgeben, sind nicht länger verfügbar.	<p>Auf die Variablen kann direkt über den Namen zugegriffen werden. Beachten Sie, dass BPEL-Variablen, deren WSDL-Nachrichtendefinition über einen einzelnen Abschnitt verfügt, diesen Abschnitt nun direkt darstellen, statt mit den tatsächlichen Daten in einen Wrapper eingeschlossen. Bei Variablen, deren Nachrichtentyp über mehrere Abschnitte verfügt, sind die Abschnitte in einen DataObject-Wrapper eingeschlossen (während es sich beim Wrapper in WebSphere Application Developer Integration Edition um eine WSIFMessage gehandelt hat).</p> <p>Beachten Sie, dass es zwei Variationen der <code>getVariableAsWSIFMessage</code>-Methode gab: <code>getVariableAsWSIFMessage(String variableName)</code> <code>getVariableAsWSIFMessage(String variableName, boolean readOnly)</code></p> <p>Der Standardzugriff auf eine Java-Snippet-Aktivität ist der Schreib-/Lesezugriff. Diese Standardeinstellung können Sie in einen Lesezugriff ändern, indem Sie <code>@bpe.readOnlyVariables</code> mit der Liste der Variablennamen in einem Kommentar im Snippet angeben. Beispiel: Eine Festlegung der Variablen B und D als Lesezugriff könnte folgendermaßen aussehen:</p> <pre>variableB.setString("/x/y/z", variableA.getString("/a/b/c")); // @bpe.readOnlyVariables variableD.setInt("/x/y/z", variableC.getInt("/a/b/c")); // @bpe.readOnlyVariables</pre> <p>Falls Sie ein Java-Snippet in einer Bedingung verwenden, sind Variablen standardmäßig schreibgeschützt. Sie können dies jedoch in einen Schreib-/Lesezugriff ändern, indem Sie <code>@bpe.readWriteVariables...</code> angeben.</p>
Ungettype Setter-Methoden für BPEL-Variablen sind nicht länger verfügbar.	Auf die Variablen kann direkt über den Namen zugegriffen werden. Beachten Sie, dass BPEL-Variablen, deren WSDL-Nachrichtendefinition über einen einzelnen Abschnitt verfügt, diesen Abschnitt nun direkt darstellen, statt mit den tatsächlichen Daten in einen Wrapper eingeschlossen. Bei Variablen, deren Nachrichtentyp über mehrere Abschnitte verfügt, sind die Abschnitte in einen DataObject-Wrapper eingeschlossen (während es sich beim Wrapper in WebSphere Application Developer Integration Edition um eine WSIFMessage gehandelt hat).

Tabelle 11. Änderungen und Lösungen für die Migration von WebSphere Business Integration Server Foundation BPEL Java-Snippets (Forts.)

Änderung	Lösung
Ungetypte Getter-Methoden für Nachrichtenabschnitte von BPEL-Variablen sind nicht geeignet für Nachrichten mit einem einzelnen Abschnitt und haben sich für Nachrichten mit mehreren Abschnitten geändert.	<p>Migrieren Sie zu der ungetypten Getter-Methode für die Eigenschaften von BPEL- Variablen (DataObjects).</p> <p>Beachten Sie, dass BPEL-Variablen, deren WSDL-Nachrichtendefinition über einen einzelnen Abschnitt verfügt, diesen Abschnitt direkt darstellen, und auf die Variable sollte direkt ohne Verwendung einer Getter-Methode zugegriffen werden.</p> <p>Es gab zwei Variationen der getVariablePartAsObject-Methode:</p> <pre>getVariablePartAsObject(String variableName, String partName) getVariablePartAsObject(String variableName, String partName,boolean</pre> <p>Bei Nachrichten mit mehreren Abschnitten wird entsprechende Funktionalität durch diese Methode in 6.0 geboten:</p> <pre>getVariableProperty(String variableName, QName propertyName);</pre> <p>In 6.0 gibt es keinen Vorgang für die Verwendung einer Variablen für schreibgeschützten Zugriff (dies war der Fall für die erste Methode oben sowie die zweite Methode mit forUpdate='false'). Die Variable wird direkt im 6.0 Snippet verwendet und kann jederzeit aktualisiert werden.</p>
Ungetypte Setter-Methoden für Nachrichtenabschnitte von BPEL-Variablen sind nicht geeignet für Nachrichten mit einem einzelnen Abschnitt und haben sich für Nachrichten mit mehreren Abschnitten geändert.	<p>Migrieren Sie zu der ungetypten Setter-Methode für die Eigenschaften von BPEL- Variablen (DataObjects).</p> <p>Beachten Sie, dass BPEL-Variablen, deren WSDL-Nachrichtendefinition über einen einzelnen Abschnitt verfügt, diesen Abschnitt direkt darstellen, und auf die Variable sollte direkt ohne Verwendung einer Setter-Methode zugegriffen werden.</p> <p>Aufrufe der folgenden Methode müssen migriert werden:</p> <pre>setVariableObjectPart(String variableName, String partName, Object da</pre> <p>Bei Nachrichten mit mehreren Abschnitten wird entsprechende Funktionalität durch diese Methode in 6.0 geboten:</p> <pre>setVariableProperty(String variableName, QName propertyName,Serializa</pre>
Getypte Getter-Methoden für BPEL-Partnerverbindungen sind nicht länger verfügbar.	Migrieren Sie zu den ungetypten Getter-Methoden für BPEL-Partnerverbindungen.
Getypte Setter-Methoden für BPEL-Partnerverbindungen sind nicht länger verfügbar.	Migrieren Sie zu den ungetypten Setter-Methoden für BPEL-Partnerverbindungen.
Getypte Getter-Methoden für BPEL-Korrelationsgruppen sind nicht länger verfügbar.	<p>V5.1-Snippet:</p> <pre>String corrSetPropStr = getCorrelationSetCorrSetAPropertyCust int corrSetPropInt = getCorrelationSetCorrSetBPropertyCustome</pre> <p>V6.0-Snippet:</p> <pre>String corrSetPropStr = (String) getCorrelationSetProperty("C int corrSetPropInt = ((Integer) getCorrelationSetProperty ("C</pre>

Tabelle 11. Änderungen und Lösungen für die Migration von WebSphere Business Integration Server Foundation BPEL Java-Snippets (Forts.)

Änderung	Lösung
Zusätzlicher benötigter Parameter für die ungetypten Getter-Methoden für benutzerdefinierte BPEL-Aktivitätseigenschaften.	V5.1-Snippet: String val = getActivityCustomProperty("propName"); V6.0-Snippet: String val = getActivityCustomProperty("name-of-current-activity");
Zusätzlicher benötigter Parameter für die ungetypten Setter-Methoden für benutzerdefinierte BPEL-Aktivitätseigenschaften.	V5.1-Snippet: String newVal = "new value"; setActivityCustomProperty("propName", newVal); V6.0-Snippet: String newVal = "new value"; setActivityCustomProperty("name-of-current-activity", "propName", newVal);
Die Methode raiseFault(QName faultQName, Serializable message) ist nicht mehr verfügbar.	Migrieren Sie wenn möglich zu raiseFault(QName faultQName, String variableName); ansonsten migrieren Sie zur Methode raiseFault(QName faultQName) oder erstellen Sie eine neue BPEL-Variable für das serialisierbare Objekt.

Interaktionen mit WebSphere Business Integration-Adaptern migrieren:

Wenn es sich beim JMS-Client um einen Adapter von WebSphere Business Integration handelt, müssen Sie die Tools von Enterprise Service Discovery verwenden, um den Import mit JMS-Binding zu erstellen. Dieser Import verwendet ein spezielles Datenbinding zur Serialisierung der SDO in das genaue Format, das vom Adapter von WebSphere Business Integration erwartet wird.

So greifen Sie auf die Tools von Enterprise Service Discovery zu:

1. Gehen Sie zu **'Datei' → 'Neu' → 'Sonstige' → 'Geschäftsintegration'** und wählen Sie **'Enterprise Service Discovery'**. Klicken Sie auf **'Weiter'**.
2. Wählen Sie **WebSphere Business Integration Adapter Artifact Importer**. Klicken Sie auf **'Weiter'**.
3. Geben Sie den Pfad der Konfigurationsdatei (.cfg)-Datei für den Adapter von WebSphere Business Integration ein und das Verzeichnis, das das XML-Schema des Geschäftsobjekts enthält, das vom Adapter verwendet wird. Klicken Sie auf **'Weiter'**.
4. Überprüfen Sie die für Sie generierte Abfrage, und wenn diese korrekt ist, klicken Sie auf **'Abfrage ausführen'**. Wählen Sie in der Liste **'Nach Abfrage erkannte Objekte'** die Objekte, die Sie hinzufügen möchten (eins nach dem anderen) und klicken Sie auf die Schaltfläche **'>> Hinzufügen'**.
5. Akzeptieren Sie die Konfigurationsparameter für das Geschäftsobjekt und klicken Sie auf **OK**.
6. Wiederholen Sie diesen Vorgang für jedes Geschäftsobjekt.
7. Klicken Sie auf **'Weiter'**.
8. Wählen Sie für das **'Laufzeitformat für Geschäftsobjekte'** die Option **SDO**. Wählen Sie für das **'Zielprojekt'** das Modul, das Sie gerade migriert haben. Lassen Sie das Feld **'Ordner'** leer.
9. Klicken Sie auf **'Fertig stellen'**.

Dieses Tool migriert die alten XSDs in das Format, das von dem speziellen Datenbinding erwartet wird, entfernen Sie daher die alten XSDs für den Adapter von WebSphere Business Integration aus dem Modul und verwenden Sie die neuen XSDs. Wenn das Modul keine Nachrichten vom Adapter empfängt, löschen

Sie die mit diesem Tool generierten Exporte. Wenn das Modul keine Nachrichten an den Adapter sendet, löschen Sie den Import. Weitere Informationen zu dieser Funktion finden Sie im Information Center.

Migration von WSDL-Schnittstellen, die über SOAP-verschlüsselte Feldgruppentypen verfügen:

Dieser Abschnitt beschreibt die Migration oder Handhabung von Schemas, die über SOAP-verschlüsselte Feldgruppentypen verfügen.

Soap-verschlüsselte Feldgruppentypen mit RPC-Stil werden als unbegrenzte Sequenzen eines konkreten Typen in 6.0 behandelt. Eine Erstellung von XSD-Typen, die in irgend einer Weise auf die Typen soapend:Array verweisen, wird nicht empfohlen, da das Programmiermodell sich in Richtung dem eingeschlossenen Dokument/Literal-Stil anstelle des RPC-Stils bewegt.

Es wird Fälle geben, in denen eine SCA-Anwendung einen externen Service aufrufen muss, der den Typ soapend:Array verwendet. Dies kann in einigen Fällen nicht vermieden werden, daher wird im Folgenden beschrieben, wie Sie diese Situation handhaben können:

WSDL-Beispielcode:

```
<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>
```

Beispielcode für einen Client dieses Web-Services:

```
// Locate the vendor service and find the doFindVendor operation
Service findVendor=(Service)ServiceManager.INSTANCE.locateService("vendorSearch");
OperationType doFindVendorOperationType=findVendor.getReference().getOperationType("doGoogleSearch");
// Create the input DataObject
DataObject doFindVendor=DataFactory.INSTANCE.create(doFindVendorOperationType.getInputType());
doFindVendor.setString("productId", "12345");
doFindVendor.setString("productName", "Refrigerator");
// Invoke the FindVendor service
DataObject findVendorResult = (DataObject)findVendor.invoke(doFindVendorOperationType, doFindVendor);
// Display the results
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

```

Im Folgenden finden Sie ein weiteres Beispiel, in dem es sich beim Roottyp um ein soapenc:Array handelt. Achten Sie darauf, wie das sampleElements DataObject mit dem zweiten oben aufgelisteten Schema erstellt wird. Zunächst wird der Typ des DataObject ermittelt und anschließend die Eigenschaft für sampleStructElement. Hierbei handelt es sich in Wirklichkeit um eine Platzhalter Eigenschaft und diese wird nur verwendet, um eine gültige Eigenschaft beim Hinzufügen der DataObjects zur Sequenz zu erhalten. In Ihrem Szenario kann das folgende Muster verwendet werden:

WSDL-Beispielcode:

```

<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>

```

Beispielcode für einen Client dieses Web-Services:

```

// Create the input DataObject and get the SDO sequence for the any
// element
DataFactory dataFactory=DataFactory.INSTANCE;
DataObject arrayOfStruct = dataFactory.create("http://soapinterop.org/xsd","ArrayOfSOAPStruct");
Sequence sequence=arrayOfStruct.getSequence("any");
// Get the SDO property for the sample element that we want to use
// here to populate the sequence
// We have defined this element in an XSD file, see SampleElements.xsd
DataObject sampleElements=dataFactory.create("http://sample/elements",
"DocumentRoot");
Property property = sampleElements.getType().getProperty("sampleStructElement");
// Add the elements to the sequence
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);
// Invoke the echoStructArray operation
System.out.println("[client] invoking echoStructArray operation");
DataObject echoArrayOfStruct = (DataObject)interopTest.invoke("echoStructArray", arrayOfStruct);
// Display the results
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"));

```

Migration des WebSphere Business Integration EJB-Projekts:

In WebSphere Studio Application Developer Integration Edition können EJB-Projekte über spezielle WebSphere-Geschäftsintegrationsfunktionen wie erweiterte Nachrichtenübertragung (Extended Messaging, CMM) und CMP/A komponentenbasierte Persistenz überall (Component-managed persistence anywhere) verfügen. Die Implementierungsdeskriptoren für solche Projekte müssen migriert werden, und dieser Abschnitt beschreibt, wie diese Migration durchgeführt wird.

Um diese Migration durchzuführen, führen Sie die folgenden Schritte durch:

1. Kopieren Sie das WebSphere Business Integration EJB-Projekt in den neuen 6.0 Arbeitsbereich und importieren Sie es aus WebSphere Integration Developer mithilfe des Assistenten unter **'Datei' → 'Importieren' → 'Vorhandenes Projekt in Arbeitsbereich'**. Optional können Sie auch den J2EE-Migrationsassistenten ausführen.
2. Schließen Sie alle laufenden Instanzen von WebSphere Integration Developer im 6.0 Arbeitsbereich.
3. Führen Sie das folgende Script aus, das die WebSphere Business Integration-Implementierungsdeskriptoren im EJB-Projekt migriert:

Unter Windows:

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

Unter Linux:

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

Die folgenden Parameter werden unterstützt, wobei der Arbeitsbereich und Projektname verbindlich sind:

```

Syntax:      WSADIEEJBProjectMigration.bat
[-e eclipse-folder] -d workspace -p project
eclipse-folder: Die Speicherposition Ihres Eclipse-Ordners -- normalerweise lautet diese 'eclipse'
                 befindet sich in Ihrem Produktinstallationsordner.
workspace:      Der Arbeitsbereich, der das zu migrierende WSADIE EJB-Projekt enthält.
project:        Der Name des zu migrierenden Projekts.

```

Beispielsweise

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

4. Beim Öffnen von WebSphere Integration Developer müssen Sie das EJB-Projekt aktualisieren, um die aktualisierten Dateien zu erhalten.
5. Suchen Sie nach der Datei **ibm-web-ext.xmi** im EJB-Projekt. Wenn eine gefunden wurde, vergewissern Sie sich, dass die folgende Zeile in der Datei unter Element enthalten ist:

```

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

```

6. Entfernen Sie den alten Implementierungscode, der in 5.1 erzeugt wurde. Generieren Sie den Implementierungscode neu, indem Sie die WebSphere Application Server-Richtlinien zu diesem Thema befolgen.

Namensbereichskollisionen manuell korrigieren:

In WebSphere Studio Application Developer Integration Edition 5.1 wurde die Definition von zwei unterschiedlichen XSD- oder WSDL-Typen mit dem gleichen Namen und demselben Zielnamensbereich unterstützt. In WebSphere Integration Developer 6.0 wird dies nicht unterstützt. Falls Sie nach der Erstellung der migrierten Projekte Fehler aufgrund von doppelt vorhandenen Definitionen feststellen, ist eine manuelle Migration erforderlich.

So können Sie dieses Problem lösen:

1. Falls die Definitionen identisch sind, löschen Sie eine der Definitionen. Anschließend müssen Sie das Projekt bereinigen und erneut erstellen. Korrigieren Sie alle hierdurch möglicherweise verursachten Fehler, indem Sie in vorhandenen XSDL-/XSD-Dateien auf die Datei mit der Definition verweisen, die Sie *nicht* gelöscht haben.
2. Falls die Definitionen *nicht* identisch sind und Sie beide Definitionen im migrierten Service verwenden müssen, benennen Sie den Definitionsnamen oder den Zielnamensbereich um. Falls lediglich wenige Definitionen in der gesamten Datei doppelt vorhanden sind, empfiehlt es sich, ihre Namen zu ändern. Sind alle Definitionen in der Datei doppelt vorhanden, wird die Änderung des Zielnamensbereichs für alle Definitionen empfohlen. Bereinigen Sie das Projekt, und erstellen Sie es erneut. Achten Sie hierbei darauf, dass die Artefakte, die Sie in den geänderten Definitionen verwenden wollen, auf den neuen Definitionsnamen oder Namensbereich verweisen.
3. Falls zwei Importanweisungen für denselben Namensbereich in einer WSDL-Datei vorhanden sind, können Sie dieses Problem lösen, indem Sie die Importe so verketteten, dass eine der WSDL-Dateien die andere Datei importiert, die wiederum die nächste Datei importiert usw. Auf diese Weise findet nur ein Import für diesen Namensbereich pro WSDL-Datei statt. Anschließend müssen Sie das Projekt bereinigen und erneut erstellen.

5.1-WSIF-Definitionen manuell löschen:

Nachdem Sie die Migration der Quellenartefakte abgeschlossen haben, sollten Sie alle WSDL-Definitionen für WSIF-Bindings und -Services (WSIF = Web Services Invocation Framework) der Version 5.1, die nicht mehr verwendet werden, aus Ihren Projekten der Version 6.0 löschen. Das Auslastungsszenario für die Servicemigration ist der einzige Fall, bei dem weiterhin ein WSIF-Binding oder -Service verwendet wird.

Die folgenden WSDL-Namensbereiche geben an, dass es sich bei einer Binding- oder Servicedefinition um einen 5.1-WSIF-Service handelt, der gelöscht werden kann, wenn er nicht mehr verwendet wird:

EJB-WSIF-Namensbereich:

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

Java-WSIF-Namensbereich:

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

JMS-WSIF-Namensbereich:

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

WSIF-Namensbereich für Geschäftsprozess:

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

WSIF-Namensbereich für Umsetzungsprogramm:

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

IMS-WSIF-Namensbereich:

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

CICS-ECI-WSIF-Namensbereich:

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

CICS-EPI-WSIF-Namensbereich:

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

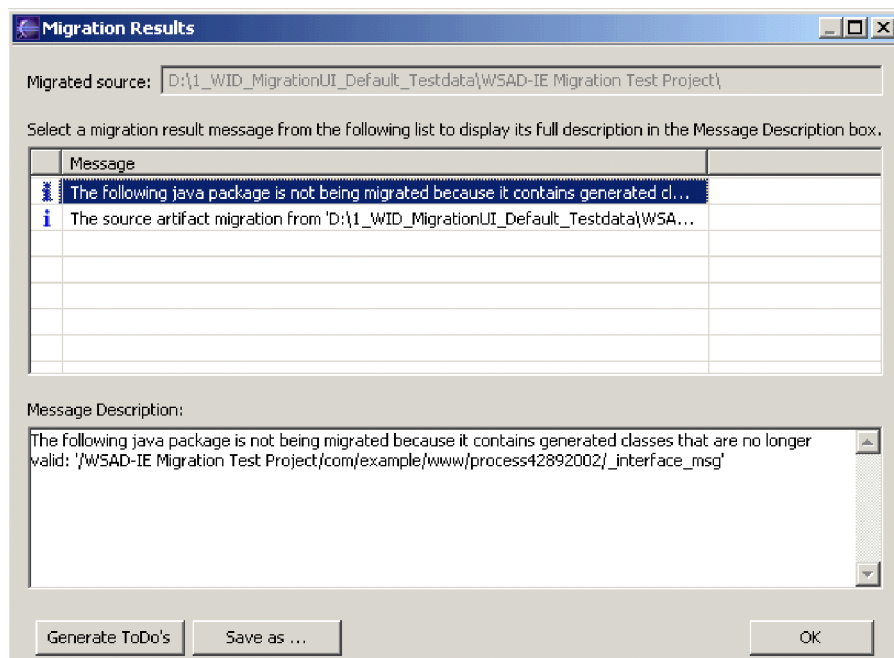
HOD-WSIF-Namensbereich:

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

Die Migration der Quellenartefakte überprüfen

Wenn der Migrationsassistent erfolgreich abgeschlossen wurde, wird eine Liste mit Fehlermeldungen, Warnungen und/oder Informationsnachrichten angezeigt. Sie können diese Nachrichten zur Überprüfung der Quellenartefaktmigration verwenden.

Die folgende Seite wird angezeigt, nachdem der Migrationsassistent beendet wurde:



Überprüfen Sie jede Nachricht, um zu sehen, ob eventuell sofortige Maßnahmen ergriffen werden müssen, um ein Artefakt, das nicht vollständig migriert werden konnte, zu korrigieren.

Um zu überprüfen, ob ein Teil einer Migration abgeschlossen ist, wechseln Sie zur Geschäftsintegrationsperspektive und stellen Sie sicher, dass alle Prozesse und WSDL-Schnittstellen des alten Serviceprojekts im neuen Modul angezeigt werden. Erstellen Sie das Projekt und korrigieren Sie alle Fehler, die die Erstellung des Projekts verhindern.

Nach Durchführen der manuellen Migrationsschritte, die für den Abschluss der Migration der Geschäftsintegrationsanwendung notwendig sind, exportieren Sie die Anwendung als eine EAR-Datei und installieren Sie sie auf einem WebSphere Process Server, und konfigurieren Sie die entsprechenden Ressourcen.

Führen Sie die manuellen Migrationsschritte aus, die für die Migration des Client-Codes notwendig sind, oder generieren Sie neuen Client-Code mithilfe von WebSphere Integration Developer. Stellen Sie sicher, dass der Client auf die Anwendung zugreifen kann und dass die Anwendung sich genauso verhält wie in der vorherigen Laufzeitumgebung.

Mit Migrationsfehlern von Quellenartefakten arbeiten

Wenn Ihre Quellenartefaktmigration von WebSphere Studio Application Developer Integration Edition fehlschlägt, gibt es eine Reihe von Möglichkeiten, mit diesen Fehlschlägen umzugehen.

Im Folgenden werden einige der möglichen Quellenartefakt-Migrationsfehler aufgeführt:

- Wenn Sie die folgende Nachricht erhalten:

"Migrationsfehlernachricht"
Grund: Fataler Migrationsfehler
Nachricht: Wenden Sie sich an Ihren IBM Ansprechpartner

Prüfen Sie die Protokolldatei von WebSphere Integration Developer im Ordner .metadata des neuen Arbeitsbereichs, um die genauen Informationen des Fehlers anzuzeigen. Beheben Sie wenn möglich die Ursache des Fehlers, löschen Sie das im neuen Arbeitsbereich erstellte Modul und führen Sie die Migration erneut durch.

Wenn der Migrationsassistent ohne diese Nachricht beendet wird, wird eine Liste mit Informationen, Warnungen und Fehlernachrichten angezeigt. Diese bedeuten, dass ein Teil des Serviceprojekts nicht automatisch migriert werden konnte, und dass manuelle Änderungen vorgenommen werden müssen, um die Migration abzuschließen.

Bewährte Verfahren für den Quellenartefakt-Migrationsprozess

Es gibt eine Reihe von bewährten Verfahren für den Quellenartefakt-Migrationsprozess von WebSphere Studio Application Developer Integration Edition.

Die folgenden Verfahren illustrieren das Design von WebSphere Studio Application Developer Integration Edition Services, um deren erfolgreiche Migration zum neuen Programmiermodell sicherzustellen:

- Versuchen Sie, die **Assign**-Aktivität so häufig wie möglich zu verwenden (im Gegensatz zum Umsetzungsservice, der nur benötigt wird, wenn eine erweiterte Umsetzung erforderlich ist. Sie sollten dieses Verfahren verwenden, da temporäre Komponenten erstellt werden müssen, so dass das SCA-Modul einen Umsetzungsservice aufrufen kann. Zusätzlich gibt es keine spezielle Toolunterstützung in WebSphere Integration Developer für die Umsetzungsservices, die in 5.1 erstellt wurden (Sie müssen den WSDL- oder XML-Editor zur Bearbeitung der in der WSDL-Datei eingebetteten XSLT, wenn Sie das Verhalten des Umsetzungsservices ändern müssen).
- Geben Sie einen Teil pro WSDL-Nachricht an, gemäß der Web-Services Interoperabilitäts- (WS-I)-Spezifikation und dem 6.0 bevorzugten Stil.
- Verwenden Sie den WSDL doc-literal-Stil, da dies der bevorzugte Stil in 6.0 ist.
- Stellen Sie sicher, dass alle komplexen Typen mit einem Namen versehen wurden und dass jeder komplexe Typ anhand seines Zielnamensbereichs und Namens eindeutig erkannt werden kann. Der folgende Code zeigt das empfohlene Verfahren für die Definition von komplexen Typen und von Elementen dieses Typs (Definition des komplexen Typs gefolgt von einer Elementdefinition, die den Typ verwendet):

```
<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>
```

Das folgende Beispiel ist ein anonymer komplexer Typ, der vermieden werden sollte, weil er bei der Serialisierung eines SDO in XML Probleme verursachen kann (ein Element enthält eine Definition für einen anonymen komplexen Typ):

```

<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>

```

- Wenn Sie einen Service für externe Kunden veröffentlichen, generieren Sie Service-Implementierungscode mithilfe von IBM Web-Services (anstelle von Apache SOAP/HTTP), da IBM Web-Services in 6.0 direkt unterstützt werden, Apache Web-Services jedoch nicht.
- Es gibt zwei Arten, auf die Sie WSDL- und XSD-Dateien in 5.1 organisieren können, um den von Ihnen durchzuführenden Reorganisationsaufwand während der Migration zu reduzieren. In 6.0 müssen gemeinsam genutzte Artefakte wie WSDL- und XSD-Dateien in BI-Projekten (Geschäftsintegrations-Module und Bibliotheken) abgelegt sein, damit auf sie von einem BI-Service verwiesen werden kann:
 - Speichern Sie alle WSDL-Dateien, die vom mehr als einem Serviceprojekt gemeinsam genutzt werden, in einem Java-Projekt, auf das die Serviceprojekte verweisen können. Während der Migration zu 6.0 erstellen Sie eine neue Geschäftsintegrationsbibliothek mit demselben Namen wie für das gemeinsam verwendete Java-Projekt 5.1. Kopieren Sie alle Artefakte aus dem gemeinsam genutzten Java-Projekt 5.1. zur Bibliothek, so dass der Migrationsassistent die Artefakte auflösen kann, wenn die Serviceprojekte, die diese Artefakte verwenden, migriert werden.
 - Speichern Sie eine lokale Kopie aller WSDL/XSD-Dateien, auf die ein Serviceprojekt verweist, im Serviceprojekt selbst. WebSphere Studio Application Developer Integration Edition-Serviceprojekte werden in ein Geschäftsintegrationsmodul in WebSphere Integration Developer migriert und ein Modul kann keine Abhängigkeiten von anderen Modulen haben (ein Serviceprojekt mit Abhängigkeiten von anderen Serviceprojekten für die gemeinsame Nutzung von WSDL- oder XSD-Dateien kann nicht problemlos migriert werden).
- Vermeiden Sie die Verwendung der generischen Nachrichtenübertragungs-API (generische MDBs) des Geschäftsprozess-Choreographen, da diese in 6.0 nicht enthalten ist. Eine MDB-Schnittstelle, die ein spätes Binding bietet, steht in 6.0 *nicht* zur Verfügung.
- Verwenden Sie die EJB-API des generischen Geschäftsprozess-Choreographen, statt die generierten Session-Beans aufzurufen, die für eine bestimmte Version eines Prozesses spezifisch sind. Diese Session-Beans werden in V6.0.0.0 nicht generiert.
- Wenn Sie über einen Geschäftsprozess mit mehreren Antworten für dieselbe Operation verfügen, stellen Sie im Falle von Client-Einstellungen sicher, dass alle Antworten für diese Operation über dieselben Client-Einstellungen verfügen, da in 6.0 nur ein Satz mit Client-Einstellungen pro Operationsantwort unterstützt wird.
- Entwerfen Sie BPEL Java-Snippets gemäß den folgenden Richtlinien:
 - Vermeiden Sie das Senden von WSIFMessage-Parametern zu benutzerdefinierten Java-Klassen - versuchen Sie, wenn möglich nicht vom WSIFMessage-Datenformat abzuhängen.
 - Vermeiden Sie wenn möglich die Verwendung von WSIF-Metadaten-APIs.
- Vermeiden Sie die Deklaration von lokalen Variablen in BPEL Java-Snippets, die denselben Namen wie die BPEL-Variablen haben. In 6.0 sind BPEL-Variablen direkt über die Snippets zugänglich, sodass lokale Variablen mit demselben Namen einen Konflikt verursachen können.
- Vermeiden Sie wenn möglich die Erstellung von Top-down EJBs oder Java-Services, da das Java/EJB-Gerüst, das aus den WSDL-Porttypen/Nachrichten generiert wird, von WSIF-Klassen abhängt (z. B. WSIFFormatPartImpl). Erstellen Sie stattdessen die Java/EJB-Schnittstellen zuerst und generieren Sie einen Service um die Java-Klasse/EJB herum (Bottom-up-Methode).

- Vermeiden Sie die Erstellung oder Verwendung von WSDL-Schnittstellen, die auf den Typ `soapenc:Array` verweisen, da dieser Schnittstellentyp nicht naturgemäß im SCA-Programmiermodell unterstützt wird
- Vermeiden Sie die Erstellung von Nachrichtentypen, bei deren problemorientierten Nachrichtentypen es sich um einen Feldgruppentyp handelt (`maxOccurs`-Attribut ist größer als eins), da dieser Schnittstellentyp nicht naturgemäß im SCA-Programmiermodell unterstützt wird.
- Definieren Sie Ihre WSDL-Schnittstellen genau - vermeiden Sie wenn möglich XSD `complexType`s, die auf den Typ `xsd:anyType` verweisen.
- Achten Sie bei jeder WSDL- und XSD-Definition, die Sie aus einer EJB- oder Java-Bean generieren, darauf, dass der Zielnamensbereich eindeutig ist (der Java-Klassenname und -Paketname werden durch den Zielnamensbereich dargestellt), um bei der Migration auf WebSphere Process Server V6 Kollisionen zu verhindern. In WebSphere Process Server V6 sind zwei unterschiedliche WSDL/XSD-Definitionen mit einem identischen Namen und Zielnamensbereich nicht zulässig. Diese Situation tritt häufig auf, wenn der Web-Service-Assistent oder der Befehl 'Java2WSDL' ohne explizite Angabe des Zielnamensbereichs verwendet wird (der Zielnamensbereich ist zwar für den Paketnamen der EJB- oder Java-Bean eindeutig, nicht jedoch für die Klasse selbst, so dass bei der Generierung eines Web-Services für zwei oder mehr EJB- oder Java-Beans im gleichen Paket Probleme auftreten). Die Lösung besteht darin, im Web-Service-Assistenten eine angepasste Zuordnung von Paket und Namensbereich anzugeben oder die Befehlszeilenoption '**-namespace**' für den Befehl 'Java2WSDL' zu verwenden, um sicherzustellen, dass der Namensbereich der generierten Dateien für die jeweilige Klasse eindeutig ist.
- Verwenden Sie nach Möglichkeit für jede WSDL-Datei eindeutige Namensbereiche. In der WSDL 1.1-Spezifikation gibt es Einschränkungen hinsichtlich des Imports von zwei unterschiedlichen WSDL-Dateien mit demselben Namensbereich. Diese Einschränkungen werden in WebSphere Integration Developer 6.0 genauestens umgesetzt.

Einschränkungen des Migrationsprozesses (für Quellenartefaktmigration)

Es gibt bestimmte Einschränkungen für den Quellenartefakt-Migrationsprozess von WebSphere Studio Application Developer Integration Edition.

In den folgenden Listen sind einige der Einschränkungen des Migrationsprozesses für die Quellenartefaktmigration im Detail beschrieben:

Allgemeine Einschränkungen

- Dieser Migrationsassistent kann keine vollständigen Arbeitsbereiche von WebSphere Studio Application Developer Integration Edition verarbeiten. Er ist dazu gedacht, ein WebSphere Studio Application Developer Integration Edition-Serviceprojekt zur Zeit zu migrieren.
- Der Migrationsassistent migriert keine Anwendungs-Binaries - er migriert nur Quellenartefakte, die in einem WebSphere Studio Application Developer Integration Edition-Serviceprojekt enthalten sind.
- Business Rule Beans werden in WebSphere Process Server 6.0 nicht mehr unterstützt, es gibt während der Installation von WebSphere Process Server jedoch eine Option, die Unterstützung für die nicht mehr unterstützten Business Rule Beans zu installieren, damit diese auf einem Server von WebSphere Process Server 6.0 "ohne Wartung (auf AS-IS-Basis)" ausgeführt werden können. Es gibt jedoch keine Toolunterstützung für die alten Business Rule Beans, und wenn die alten Business Rule Bean-Artefakte in den Tools kompiliert werden sollen, müssen Sie den Anweisungen in der Dokumentation von WebSphere Integration Developer für die Installation dieser nicht länger unterstützten Funktionen über den eingebetteten Testserver von WebSphere Process Server 6.0 folgen, und die nicht länger unterstützten JAR-Dateien zum Klassenpfad des Projekts als externe JARs hinzufügen. Sie sollten die neuen Geschäftsregeltools, die in WebSphere Integration Developer zur Verfügung stehen, verwenden, um ihre Geschäftsregeln gemäß der 6.0-Spezifikation neu zu erstellen.
- Die erweiterte Nachrichtenübertragung wird in WebSphere Process Server 6.0 nicht mehr unterstützt, es gibt während der Installation von WebSphere Process Server jedoch eine Option, die Unterstützung für die nicht mehr unterstützten Funktionen der erweiterten Nachrichtenübertragung zu installieren, damit diese auf einem Server von WebSphere Process Server 6.0 "ohne Wartung (auf AS-IS-Basis)" ausgeführt werden können. Es gibt jedoch keine Unterstützung für die alten Funktionen der erweiterten

Nachrichtenübertragung, und wenn die alten Artefakte der erweiterten Nachrichtenübertragung in den Tools kompiliert werden sollen, müssen Sie den Anweisungen in der Dokumentation von WebSphere Integration Developer für die Installation dieser nicht länger unterstützten Funktionen über den eingebetteten Testserver von WebSphere Process Server 6.0 folgen, und die nicht länger unterstützten JAR-Dateien zum Klassenpfad des Projekts als "externe JARs" hinzufügen.

- Das bereitgestellte JMS-Standarddatenbinding ermöglicht keinen Zugriff auf die angepassten JMS-Headereigenschaften. Damit die SCA-Services auf alle angepassten JMS-Headereigenschaften zugreifen können, muss ein angepasstes Datenbinding geschrieben werden.

Einschränkungen für das SCA-Programmiermodell

- Die SDO Version 1 Spezifikations-API bietet keinen Zugang zur COBOL- oder C-Bytefeldgruppe – dies betrifft diejenigen, die mit IMS-Multisegmenten arbeiten.
- Die SDO Version 1 Spezifikation für serielle Verarbeitung unterstützt keine COBOL-Neudefinitionen oder C-Datentypvariablen.
- Beachten Sie bei der Überarbeitung Ihrer Quellenartefakte gemäß dem SCA-Programmiermodell, dass der WSDL-Stil für den Einschluss eines Dokuments/Literals (dies ist der Standardstil für neue mit WebSphere Integration Developer-Tools erstellte Artefakte) die Methodenüberlastung nicht unterstützt. Die anderen WSDL-Stile werden nach wie vor unterstützt, daher wird empfohlen, einen anderen WSDL-Stil/Verschlüsselung als Einschluss eines Dokuments/Literals für diese Fälle zu verwenden.
- Die native Unterstützung für Feldgruppen ist beschränkt. Um einen externen Service aufzurufen, der eine WSDL-Schnittstelle mit Feldgruppentypen 'soapenc:Array' zugänglich macht, müssen Sie eine WSDL-Schnittstelle erstellen, die ein Element definiert, dessen Attribut "maxOccurs" größer als 1 ist (dies ist der empfohlene Vorgang für den Entwurf eines Feldgruppentyps).

Technische Einschränkungen für den BPEL-Migrationsprozess

- **Mehrere Antworten pro BPEL Operation** - In WebSphere Business Integration Server Foundation konnte ein Geschäftsprozess über eine Empfangsaktivität und mehrere Antwortaktivitäten für dieselbe Operation verfügen.
- **Einschränkungen der BPEL Java Snippet-Migration** - Das Programmiermodell hat sich von WebSphere Studio Application Developer Integration Edition zu WebSphere Integration Developer erheblich geändert und nicht alle unterstützten WebSphere Studio Application Developer Integration Edition APIs können direkt zu entsprechenden WebSphere Integration Developer APIs migriert werden. Die Java-Logik kann in den BPEL Java Snippets gefunden werden, sodass das automatische Migrationstool möglicherweise nicht jedes Java Snippet in das neue Programmiermodell konvertieren kann. Die meisten der Standard Snippet API-Aufrufe werden automatisch von der Version 5.1 des Java Snippet-Programmiermodells zur Version 6.0 des Java Snippet-Programmiermodells migriert. WSIF API-Aufrufe werden wenn möglich zu DataObject API-Aufrufen migriert. Alle benutzerdefinierten Java-Klassen, die WSIFMessage-Objekte übernehmen, erfordern manuelle Migration, sodass sie stattdessen `commonj.sdo.DataObject`-Objekte übernehmen und zurückgeben:
 - **WSIFMessage Metadaten-APIs** - Jede WSIF API-Syntax sollte wenn möglich eliminiert werden. Der Migrationsassistent kann WSIFMessage Metadaten und andere WSIF APIs nicht automatisch zur SDO Entsprechung migrieren, daher ist eine manuelle Migration erforderlich.
 - **EndpointReference/EndpointReferenceType APIs** - Diese Klassen werden nicht automatisch migriert. Manuelle Migration ist erforderlich, da die Partnerlink Getter/Setter-Methoden mit `commonj.sdo.DataObject`-Objekten statt mit `com.ibm.websphere.srm.bpel.wsaddressing.EndpointReferenceType`-Objekten aus 5.1 arbeiten.
 - **Komplexe Typen mit doppelten Namen** - Wenn eine Anwendung komplexe Typen (in WSDLs oder XSDs) mit identischen Namensbereichen und lokalen Namen deklariert, oder unterschiedliche Namensbereiche mit identischen lokalen Namen, können Java Snippets, die diese Typen verwenden, nicht korrekt migriert werden. Überprüfen Sie die Snippets auf ihre Korrektheit, wenn der Migrationsassistent abgeschlossen ist.

- **Komplexe Typen mit lokalen Namen, die mit den Java-Klassen im java.lang-Paket identisch sind** - Wenn eine Anwendung komplexe Typen (in WSDLs oder XSDs) mit lokalen Namen deklariert, die mit Klassen im java.lang-Paket von J2SE 1.4.2 übereinstimmen, können Java Snippets, die die entsprechende java.lang-Klassen verwenden, nicht korrekt migriert werden. Überprüfen Sie die Snippets auf ihre Korrektheit, wenn der Migrationsassistent abgeschlossen ist.
- **BPEL-Variablen mit demselben Namen wie lokale Variablen im BPEL Java-Snippet** - Wenn Sie lokale Variablen mit demselben Namen wie die BPEL-Variablen in Ihren BPEL Java-Snippets definiert haben, müssen Sie dies manuell korrigieren, da die BPEL-Variablen nun über ihren Namen in den Java-Snippets zugänglich sind und ein Konflikt vorhanden sein kann.
- **BPEL-Variablen mit Lesezugriff und mit Schreib-/Lesezugriff** - In 5.1 konnte in jedem Java-Snippet-Code eine BPEL-Variable "mit Lesezugriff" definiert werden, was bedeutet, dass sich keine an diesem Objekt vorgenommenen Änderungen auf den Wert der BPEL-Variablen auswirken. Außerdem konnte eine BPEL-Variable mit "Schreib-/Lesezugriff" definiert werden. Dies bedeutet, dass alle am Objekt vorgenommenen Änderungen für die BPEL-Variable selbst wiedergegeben werden. Mit den folgenden vier Methoden könnte ein "Lesezugriff" auf ein Java-Snippet in einem beliebigen 5.1-BPEL-Java-Snippet erfolgen:

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

Mit den beiden folgenden Methoden könnte ein "Schreib-/Lesezugriff" auf eine BPEL-Variable in einem beliebigen 5.1-BPEL-Java-Snippet erfolgen:

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

In Version 6.0 wird der Lesezugriff und der Schreib-/Lesezugriff auf BPEL-Variablen für "jedes Snippet separat" verwaltet. Dies bedeutet, dass Sie einen besonderen Kommentar zum BPEL-Java-Snippet hinzufügen und auf diese Weise angeben können, ob Aktualisierungen der BPEL-Variablen nach Abschluss der Snippetausführung gelöscht werden oder erhalten bleiben sollen. Standardzugriffseinstellungen für die BPEL-Java-Snippettypen der Version 6.0:

```
BPEL-Java-Snippet-Aktivität
Standardzugriff: Schreib-/Lesezugriff
Standardzugriff überschreiben mit Kommentar:
@bpe.readOnlyVariables names="variableA,variableB"
BPEL-Java-Snippet-Ausdruck (wird in Zeitlimit, Bedingung usw. verwendet)
Standardzugriff: Lesezugriff
Standardzugriff überschreiben mit Kommentar:
@bpe.readWriteVariables names="variableA,variableB"
```

Während der Migration werden diese Kommentare automatisch erstellt, sobald der Zugriff auf eine Variable in einer Weise erfolgt, die in Version 6.0 nicht als Standard definiert ist. Falls es zu einem Konflikt kommt (es also im gleichen Snippet einen "Lesezugriff" und einen "Schreib-/Lesezugriff" auf die BPEL-Variable gab), wird eine Warnung ausgegeben, und der Zugriff wird auf "Schreib-/Lesezugriff" gesetzt. Wenn Sie solche Warnungen empfangen, stellen Sie sicher, dass die Einstellung des "Schreib-/Lesezugriffs" für die BPEL-Variable in Ihrer Situation korrekt ist. Andernfalls müssen Sie die Einstellung mit dem BPEL-Editor von WebSphere Integration Developer manuell korrigieren.

- **Mehrwertige Eigenschaften in komplexen Typen** - In 5.1 werden mehrwertige Eigenschaften durch Feldgruppen des Eigenschaftentyps dargestellt. Daher verwenden Aufrufe für den Abruf und die Einstellung der Eigenschaft Feldgruppen. In 6.0 wird java.util.List für diese Darstellung verwendet. Die automatische Migration verarbeitet alle Vorgänge, bei denen es sich bei der mehrwertigen Eigenschaft um einen beliebigen Typ eines Java-Objekts handelt, wenn es sich beim Eigenschaftentyp jedoch um Java-Basiselemente handelt (int, long, short, byte, char, float, double und boolean), werden Aufrufe für den Abruf und die Einstellung der gesamten Feldgruppe nicht konvertiert. Die manuelle Migration macht in einem solchen Fall möglicherweise das Hinzufügen einer Schleife für

den Einschluss / das Auspacken der Basiselemente in/aus ihrer entsprechenden Java Wrapper-Klasse (Integer, Long, Short, Byte, Character, Float, Double und Boolean) für die Verwendung im verbleibenden Snippet erforderlich.

- **Instanzerstellung von generierten Klassen, die komplexe Typen darstellen** - In 5.1 können in einer Anwendung generierte Klassen komplexer Typen leicht in einem Java Snippet mit dem Standard No-Argument-Konstruktor instanziiert werden. Es folgt ein Beispiel hierfür:

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

In 6.0 muss eine spezielle Factory-Class verwendet werden, um diese Typen zu Instanzieren, oder es kann eine Instanz des übergeordneten Typs zur Erstellung des Subtyps verwendet werden. Wenn eine BPEL Prozessvariable InputVariable als Typ InputMessage definiert wurde, würde die 6.0 Version des vorhergehenden Snippets folgendermaßen lauten:

```
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");
```

Der Snippet Converter versucht, diese Änderung durchzuführen, wenn jedoch die Reihenfolge, in der die ursprünglichen Instanzerstellungen auftreten, nicht dem Hierarchiemuster (übergeordnet gefolgt von untergeordnet) folgt, ist eine manuelle Migration erforderlich (d.h. der Converter versucht nicht, die Instanzerstellungsanweisungen im Snippet intelligent neu anzuordnen).

- In WebSphere Business Integration Server Foundation 5.1 wurden dynamische Verweise als WSDL-Nachrichtenteile des Typs EndpointReferenceType oder Elements EndpointReference aus dem folgenden Namensbereich dargestellt:

<http://wsaddressing.bpel.srm.websphere.ibm.com>

Solche Verweise werden zum Standard-Elementtyp service-ref aus dem Standardnamensbereich des Geschäftsprozesses migriert:

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

<http://schemas.xmlsoap.org/ws/2004/08/addressing>

Anweisungen für den manuellen Import dieser Schemadefinitionen in Ihr Projekt, so dass alle Verweise ordnungsgemäß aufgelöst werden können, finden Sie in der BPEL-Editor-Dokumentation.

- **Nachrichtentyp für BPEL-Variable** - Für alle BPEL-Variablen, die in Java-Snippets verwendet werden, muss ein WSDL-Nachrichtentyp angegeben werden. Java-Snippets, die ohne angegebenes Attribut "messageType" auf BPEL-Variablen zugreifen, können nicht migriert werden.

Bemerkungen

Die in diesem IBM Produkt eingeschlossene XDoclet-Dokumentation unterliegt folgendem Urheberrecht: Copyright (c) 2000-2004, XDoclet Team. Alle Rechte vorbehalten.

Teile basieren auf *Design Patterns: Elements of Reusable Object-Oriented Software*, von Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides, Copyright (c) 1995 bei Addison-Wesley Publishing Company, Inc. Alle Rechte vorbehalten.

Einschränkungen für US-Beamte und -Angestellte - Nutzung, Kopieren und Weitergabe eingeschränkt durch GSA ADP Schedule Contract mit der IBM Corp.

Möglicherweise bietet IBM die in dieser Dokumentation beschriebenen Produkte, Services oder Funktionen in anderen Ländern nicht an. Informationen über die gegenwärtig im jeweiligen Land verfügbaren Produkte und Services sind beim IBM Ansprechpartner erhältlich. Hinweise auf IBM Lizenzprogramme oder andere IBM Produkte bedeuten nicht, dass nur Programme, Produkte oder Services von IBM verwendet werden können. An Stelle der Produkte, Programme oder Services können auch andere ihnen äquivalente Produkte, Programme oder Services verwendet werden, solange diese keine gewerblichen oder andere Schutzrechte der IBM verletzen. Die Verantwortung für den Betrieb von Fremdprodukten, Fremdprogrammen und Fremdservices liegt beim Kunden.

Für in diesem Handbuch beschriebene Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieses Handbuchs ist keine Lizenzierung dieser Patente verbunden. Lizenzanforderungen sind schriftlich an folgende Adresse zu richten (Anfragen an diese Adresse müssen auf Englisch formuliert werden):

IBM Director of Licensing
IBM Europe,Middle East & Africa
Tour Descartes
2,avenue Gambetta
92066 Paris La Defense
France

Trotz sorgfältiger Bearbeitung können technische Ungenauigkeiten oder Druckfehler in dieser Veröffentlichung nicht ausgeschlossen werden. Die Angaben in diesem Handbuch werden in regelmäßigen Zeitabständen aktualisiert. Die Änderungen werden in Überarbeitungen oder in Technical News Letters (TNLs) bekannt gegeben. IBM kann ohne weitere Mitteilung jederzeit Verbesserungen und/oder Änderungen an den in dieser Veröffentlichung beschriebenen Produkten und/oder Programmen vornehmen.

Verweise in diesen Informationen auf Websites anderer Anbieter dienen lediglich als Benutzerinformationen und stellen keinerlei Billigung des Inhalts dieser Websites dar. Das über diese Websites verfügbare Material ist nicht Bestandteil des Materials für dieses IBM Produkt. Die Verwendung dieser Websites geschieht auf eigene Verantwortung.

Werden an IBM Informationen eingesandt, können diese beliebig verwendet werden, ohne dass eine Verpflichtung gegenüber dem Einsender entsteht.

Lizenznehmer des Programms, die Informationen zu diesem Produkt wünschen mit der Zielsetzung: (i) den Austausch von Informationen zwischen unabhängigen, erstellten Programmen und anderen Programmen (einschließlich des vorliegenden Programms) sowie (ii) die gemeinsame Nutzung der ausgetauschten Informationen zu ermöglichen, wenden sich an folgende Adresse:

*Intellectual Property Dept. für Rational Software
IBM Corporation
20 Maguire Road
Lexington, Massachusetts 02421-3112
U.S.A.*

Die Bereitstellung dieser Informationen kann unter Umständen von bestimmten Bedingungen - in einigen Fällen auch von der Zahlung einer Gebühr - abhängig sein.

Die Lieferung des in dieser Dokumentation aufgeführten Lizenzprogramms sowie des zugehörigen Lizenzmaterials erfolgt auf der Basis der IBM Rahmenvereinbarung sowie der Allgemeinen Geschäftsbedingungen der IBM Customer Agreement, IBM, der IBM Internationalen Nutzungsbedingungen für Programmpaket oder einer äquivalenten Vereinbarung.

Alle in diesem Dokument enthaltenen Leistungsdaten stammen aus einer gesteuerten Umgebung. Die Ergebnisse, die in anderen Betriebsumgebungen erzielt werden, können daher erheblich von den hier erzielten Ergebnissen abweichen. Einige Daten stammen möglicherweise von Systemen, deren Entwicklung noch nicht abgeschlossen ist. Eine Gewährleistung, dass diese Daten auch in allgemein verfügbaren Systemen erzielt werden, kann nicht gegeben werden. Darüber hinaus wurden einige Daten unter Umständen durch Extrapolation berechnet. Die tatsächlichen Ergebnisse können abweichen. Benutzer dieses Dokuments sollten die entsprechenden Daten in ihrer spezifischen Umgebung prüfen.

Alle Informationen zu Produkten anderer Anbieter stammen von den Anbietern der aufgeführten Produkte, deren veröffentlichten Ankündigungen oder anderen allgemein verfügbaren Quellen. IBM hat diese Produkte nicht getestet und kann daher keine Aussagen zu Leistung, Kompatibilität oder anderen Merkmalen der Fremdprodukte machen. Fragen zu den Leistungsmerkmalen von Produkten anderer Anbieter sind an den jeweiligen Anbieter zu richten.

Die oben genannten Erklärungen bezüglich der Produktstrategien und Absichtserklärungen von IBM stellen die gegenwärtige Absicht der IBM dar, unterliegen Änderungen oder können zurückgenommen werden, und repräsentieren nur die Ziele der IBM.

Diese Veröffentlichung enthält Beispiele für Daten und Berichte des alltäglichen Geschäftsablaufes. Sie sollen nur die Funktionen des Lizenzprogrammes illustrieren; sie können Namen von Personen, Firmen, Marken oder Produkten enthalten. Alle diese Namen sind frei erfunden; Ähnlichkeiten mit tatsächlichen Namen und Adressen sind rein zufällig.

COPYRIGHTLIZENZ:

Diese Veröffentlichung enthält Musteranwendungsprogramme, die in Quellsprache geschrieben sind. Sie dürfen diese Musterprogramme kostenlos kopieren, ändern und verteilen, wenn dies zu dem Zweck geschieht, Anwendungsprogramme zu entwickeln, verwenden, vermarkten oder zu verteilen, die mit der Anwendungsprogrammierschnittstelle konform sind, für die diese Musterprogramme geschrieben werden. Diese Beispiele wurden nicht unter allen denkbaren Bedingungen getestet. Daher kann IBM die Zuverlässigkeit, Wartungsfreundlichkeit oder Funktion dieser Programme weder zusagen noch gewährleisten. Sie dürfen diese Musterprogramme kostenlos kopieren, ändern und verteilen, wenn dies zu dem Zweck geschieht, Anwendungsprogramme zu entwickeln, verwenden, vermarkten oder zu verteilen, die mit der Anwendungsprogrammierschnittstelle von IBM konform sind.

Kopien oder Teile der Musterprogramme bzw. daraus abgeleiteter Code müssen folgenden Copyrightvermerk beinhalten:

(C) (Name Ihrer Firma) (Jahr). Teile dieses Codes sind von IBM Musterprogrammen abgeleitet. (C) Copyright IBM Corp. 2000, 2005. Alle Rechte vorbehalten.

Wird dieses Buch als Softcopy (Book) angezeigt, erscheinen keine Fotografien oder Farabbildungen.

Informationen zur Programmierschnittstelle

Die Informationen zur Programmierschnittstelle sollen Sie bei der Erstellung von Anwendungssoftware mithilfe dieses Programms unterstützen.

Allgemeine Programmierschnittstellen ermöglichen es Ihnen, Anwendungssoftware zu schreiben, die die Services der Tools dieses Programms erhält.

Diese Informationen können ferner Informationen zu Diagnose, Änderung und Optimierung enthalten. Informationen zu Diagnose, Änderung und Optimierung werden zur Verfügung gestellt, um Sie beim Debugging Ihrer Anwendungssoftware zu unterstützen.

Achtung: Verwenden Sie diese Informationen zu Diagnose, Änderung und Optimierung nicht als Programmierschnittstelle, es sie geändert werden können.

Marken und Servicemarken

Siehe <http://www.ibm.com/legal/copytrade.shtml>.



Gedruckt in Deutschland

SC12-3569-00

