



Guide de migration



Guide de migration

Important

Avant d'utiliser le présent document et le produit associé, prenez connaissance des informations générales figurant à la section Remarques à la fin du manuel.

Remarque

Les captures d'écrans de ce manuel ne sont pas disponibles en français à la date d'impression.

Deuxième édition - décembre 2005

LE PRESENT DOCUMENT EST LIVRE EN L'ETAT SANS AUCUNE GARANTIE EXPLICITE OU IMPLICITE. IBM DECLINE NOTAMMENT TOUTE RESPONSABILITE RELATIVE A CES INFORMATIONS EN CAS DE CONTREFACON AINSI QU'EN CAS DE DEFAUT D'APTITUDE A L'EXECUTION D'UN TRAVAIL DONNE.

Ce document est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. Les informations qui y sont fournies sont susceptibles d'être modifiées avant que les produits décrits ne deviennent eux-mêmes disponibles. En outre, il peut contenir des informations ou des références concernant certains produits, logiciels ou services non annoncés dans ce pays. Cela ne signifie cependant pas qu'ils y seront annoncés.

Pour plus de détails, pour toute demande d'ordre technique, ou pour obtenir des exemplaires de documents IBM, référez-vous aux documents d'annonce disponibles dans votre pays, ou adressez-vous à votre partenaire commercial.

Vous pouvez également consulter les serveurs Internet suivants :

- <http://www.fr.ibm.com> (serveur IBM en France)
- <http://www.can.ibm.com> (serveur IBM au Canada)
- <http://www.ibm.com> (serveur IBM aux Etats-Unis)

*Compagnie IBM France
Direction Qualité
Tour Descartes
92066 Paris-La Défense Cedex 50*

© Copyright IBM France 2005. Tous droits réservés.

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

Table des matières

Migration d'applications à l'aide de WebSphere Integration Developer . . .	1
Version PDF	1
Tutoriel : Migration vers WebSphere Integration Developer	1
Migration vers WebSphere Integration Developer	1
Migration vers WebSphere Process Server à partir de WebSphere InterChange Server	1

Migration vers WebSphere Integration Developer à partir de WebSphere MQ Workflow.	31
Migration d'artefacts source vers WebSphere Integration Developer à partir de WebSphere Studio Application Developer Integration Edition.	38
Remarques	103

Migration d'applications à l'aide de WebSphere Integration Developer

WebSphere Integration Developer version 6.0 fournit les outils nécessaires pour migrer votre environnement existant.

Les rubriques suivantes décrivent le concept, la référence et les instructions pas à pas pour migrer vers WebSphere Integration Developer :

Version PDF

Ces informations de migration sont également disponibles au format PDF.

Ce document est également disponible sous forme de fichier PDF.

Vous aurez besoin d'Adobe Acrobat pour le visualiser. Une version gratuite de ce logiciel est disponible sur le site Web www.adobe.com.

Tutoriel : Migration vers WebSphere Integration Developer

Ce tutoriel indique comment utiliser WebSphere Integration Developer pour migrer votre projet de service existant vers un projet de module.

Ce tutoriel contient une leçon qui se présente sous la forme d'un film :

- Leçon 1 : Migration d'un projet de service vers un projet de module.

Cliquez sur le lien suivant pour lancer le tutoriel :

Remarque : Ce lien ne fonctionne pas sur ce site Web. Pour qu'il fonctionne correctement, WebSphere Integration Developer doit être installé sur le système utilisé.

Tutoriel : Migration vers WebSphere Integration Developer

Migration vers WebSphere Integration Developer

WebSphere Integration Developer version 6.0 fournit les outils nécessaires pour migrer votre environnement existant.

Les rubriques suivantes décrivent le concept, la référence et les instructions pas à pas pour migrer vers WebSphere Integration Developer :

Migration vers WebSphere Process Server à partir de WebSphere InterChange Server

La migration à partir de WebSphere InterChange Server vers WebSphere Process Server est prise en charge par les fonctions suivantes :

Remarque : Reportez-vous aux Notes sur l'édition pour plus d'informations sur les restrictions liées à la migration dans cette édition de WebSphere Process Server.

- Migration automatique des artefacts source via les outils de migration pouvant être appelés à partir des éléments suivants :
 - Menu **Fichier** → **Importer** de WebSphere Integration Developer

- Ecran d'accueil de WebSphere Integration Developer
- Assistant de migration de WebSphere Process Server - Premières étapes
- Utilitaire de ligne de commande **reposMigrate**
- Prise en charge native dans l'environnement d'exécution de plusieurs API de WebSphere InterChange Server
- Prise en charge de la technologie WebSphere Business Integration Adapter actuelle, assurant la compatibilité des adaptateurs existants avec WebSphere Process Server

Bien que la migration d'artefacts source soit prise en charge, il est recommandé d'effectuer une analyse et des tests exhaustifs pour déterminer si les applications résultantes fonctionneront comme prévu sous WebSphere Process Server ou si elles devront être à nouveau modifiées après la migration. Cette recommandation est basée sur les restrictions suivantes liées à la parité fonctionnelle entre WebSphere InterChange Server et cette version de WebSphere Process Server. La version actuelle de WebSphere Process Server n'offre aucun support équivalant aux fonctions WebSphere InterChange Server suivantes :

- Access Interface
- Adapter Initiated Synchronous Request Response
- Synchronous Send Service Call With Time-out
- Async_in Service Call
- Isolation
- Event Sequencing
- Hot Deployment/Dynamic Update
- Security - Auditing
- Security - Fine Grain RBAC
- Group Support
- Scheduler - Pause Operation
- Security Descriptors are not migrated
- Custom XML Snippet transformations are not supported during Map and Collaboration Template migration

Chemins de migration pris en charge pour WebSphere InterChange Server

Les outils de migration de WebSphere Process Server prennent en charge la migration à partir de WebSphere InterChange Server versions 4.2.2, 4.2.3 et 4.3.

Toute version de WebSphere InterChange Server antérieure à 4.2.2 devra d'abord migrer vers la version 4.2.2, 4.2.3 ou 4.3 avant de migrer vers WebSphere Process Server.

Préparation de la migration depuis WebSphere InterChange Server

Avant de migrer vers WebSphere Process Server depuis WebSphere InterChange Server, vous devez vérifier que vous avez bien préparé votre environnement. WebSphere Process Server fournit les outils nécessaires pour migrer depuis WebSphere InterChange Server.

Ces outils de migration peuvent être appelés depuis :

- Le menu **Fichier** → **Importer** de WebSphere Integration Developer ;
- L'écran d'accueil de WebSphere Integration Developer ;
- L'Assistant de migration dans WebSphere Process Server - Premières étapes.

Les données en entrée qui sont utilisées par les outils de migration se trouvent dans un fichier jar de référentiel exporté à partir de WebSphere InterChange Server. Avant d'accéder à ces outils de migration avec ces options, vous devez d'abord :

1. vérifier que vous exécutez une version de WebSphere InterChange Server qui peut être migrée vers WebSphere Process Server. Voir la rubrique Chemins de migration pris en charge pour WebSphere InterChange Server.
2. exporter vos artefacts source depuis WebSphere InterChange Server dans un fichier de référentiel .jar à l'aide de la commande de WebSphere InterChange Server **repos_copy** (voir la documentation de WebSphere InterChange Server). Ce fichier .jar contiendra les entrées des outils de migration.

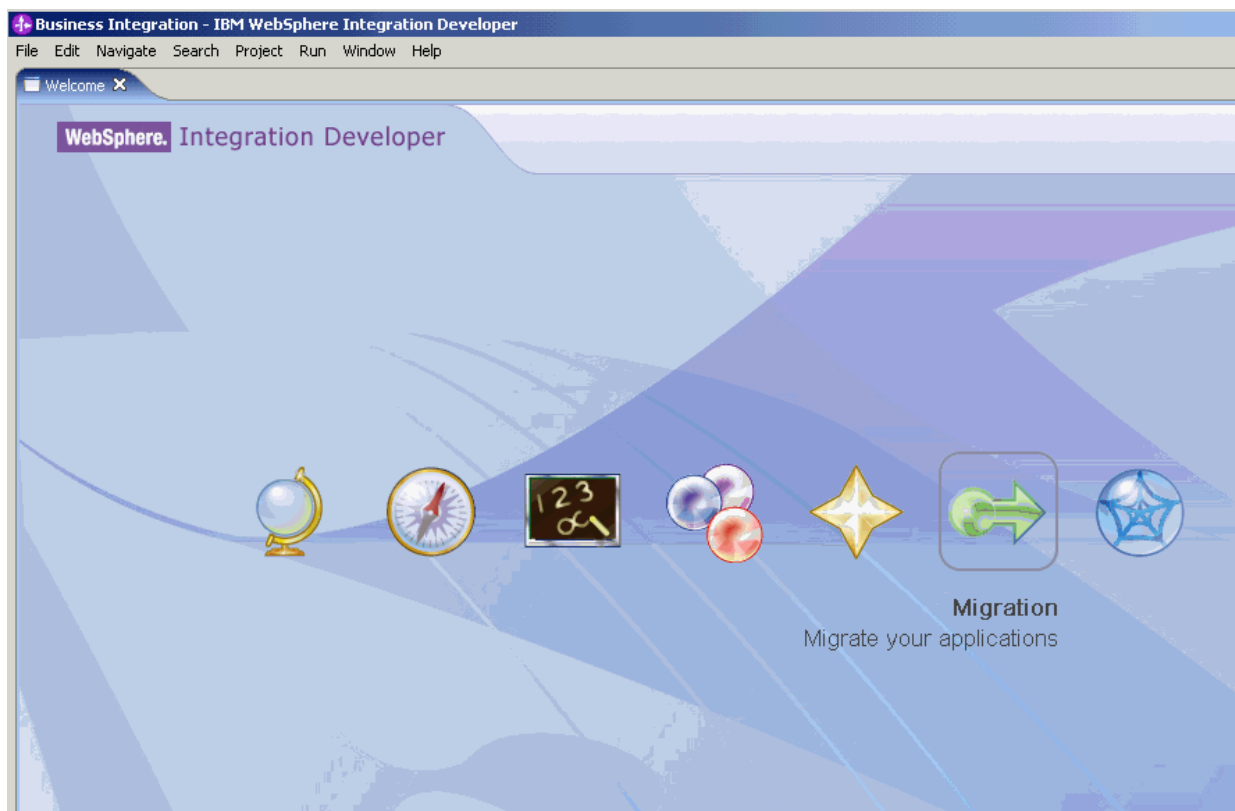
Migration de WebSphere InterChange Server avec l'Assistant de migration

Vous pouvez utiliser l'Assistant de migration de WebSphere Integration Developer pour migrer vos artefacts WebSphere InterChange Server.

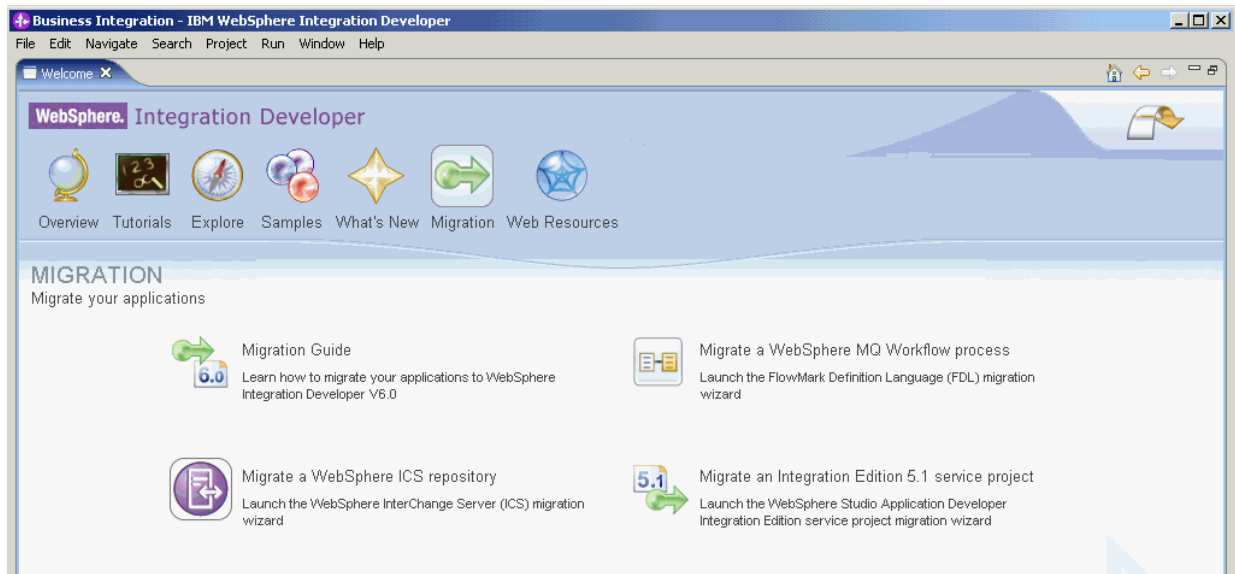
Suivez les étapes suivantes pour utiliser l'Assistant de migration et migrer vos artefacts WebSphere InterChange Server :



1. Dans la page d'accueil, cliquez sur l'icône pour ouvrir la page Migration.



2. Dans la page Migration, sélectionnez l'option Migrer un référentiel WebSphere ICS :

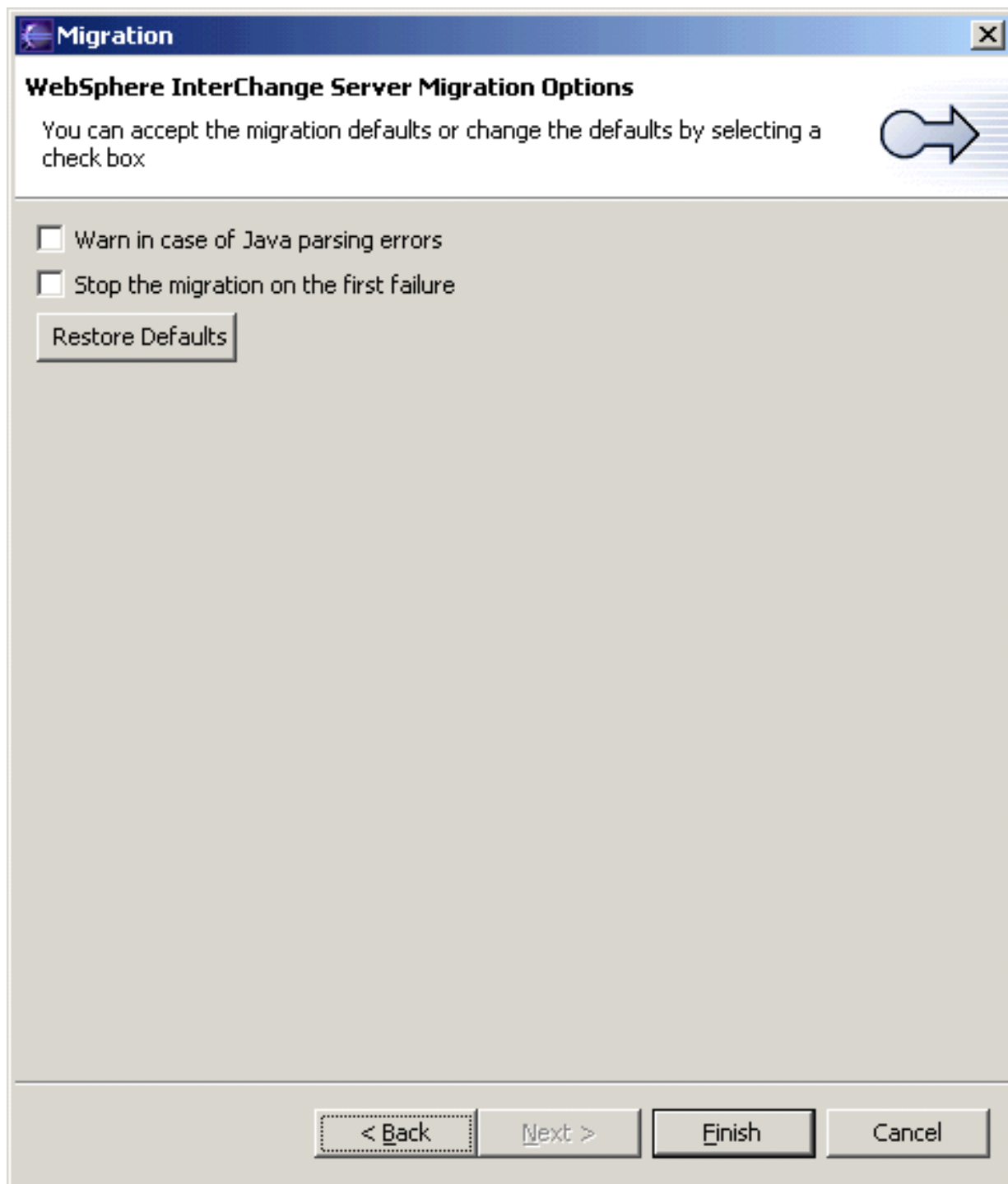


. Vous pouvez également lancer l'Assistant de migration à partir du menu **Fichier** → **Importer**, en sélectionnant **WebSphere InterChange Server - Fichier JAR** puis en cliquant sur **Suivant**.

3. L'Assistant de migration apparaît. Pour entrer le nom du fichier source dans le champ **Sélection de la source**, cliquez sur le bouton **Parcourir** et recherchez le fichier. Entrez le nom du module dans le champ approprié. Cliquez sur **Suivant**.



4. La fenêtre Options de migration apparaît. Vous pouvez accepter les options de migration par défaut ou sélectionner une case à cocher pour modifier une option.



Cliquez sur **Terminer**.

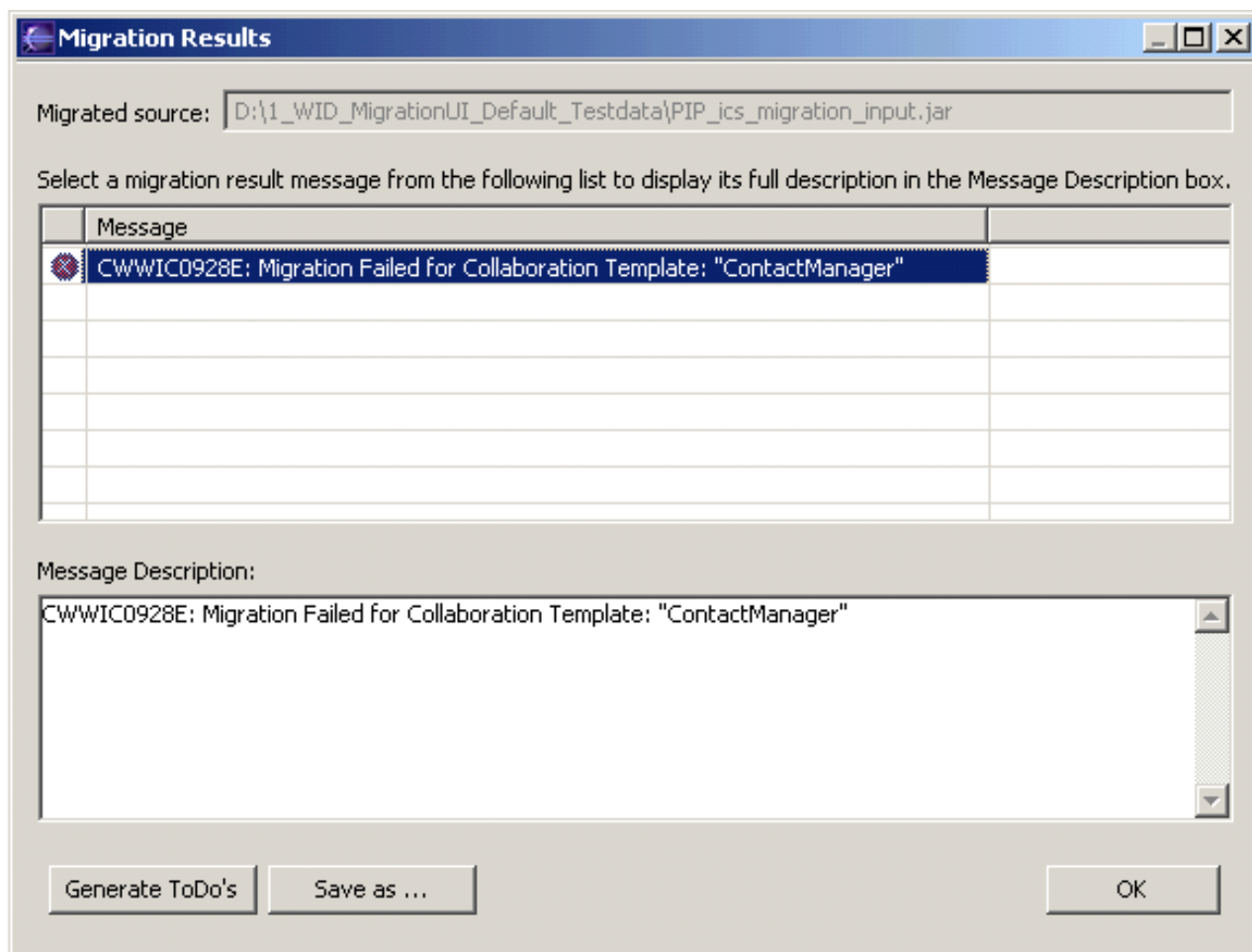
Vérification de la migration de WebSphere InterChange Server

Si aucune erreur n'est rapportée lors de la migration du fichier JAR de WebSphere InterChange Server, cela signifie que les artefacts ont été correctement migrés. Lorsque l'exécution de l'assistant d'installation échoue, une liste répertoriant les erreurs, les avertissements et/ou les messages informatifs s'affiche. Vous pouvez utiliser ces messages pour vérifier la migration de WebSphere InterChange Server.

Remarque : En raison de la complexité d'une migration de WebSphere InterChange Server vers WebSphere Process Server, il est fortement recommandé d'effectuer un test approfondi des

applications générées qui s'exécutent dans WebSphere Process Server afin de vous assurer de leur bon fonctionnement avant de les mettre en production.

La page suivante apparaît lorsque l'exécution de l'Assistant de migration est terminée :



La fenêtre Résultats de la migration contient une liste des messages de migration. Cliquez sur un message pour afficher une description complète des détails de la migration. A partir de cette liste de messages, vous pouvez créer, le cas échéant, une liste de points à corriger en cliquant sur le bouton **Générer Actions à accomplir**.

Traitement des problèmes de migration dans WebSphere InterChange Server

Si la migration depuis WebSphere InterChange Server échoue, vous pouvez régler ce problème de deux manières.

Remarque : Vous préférerez peut-être la première méthode car, au début, vous connaissez mieux WebSphere InterChange Server. Toutefois, à mesure que vous connaîtrez mieux WebSphere Process Server et ses nouveaux artefacts, vous choisirez peut-être de réparer les artefacts migrés dans WebSphere Integration Developer.

1. Si la nature de l'erreur le permet, vous pouvez modifier les artefacts de WebSphere InterChange Server à l'aide des outils de WebSphere InterChange Server puis exporter le fichier .jar et tenter une nouvelle migration.
2. Vous pouvez corriger toutes les erreurs dans les nouveaux artefacts de WebSphere Process Server en les éditant dans WebSphere Integration Developer.

Artefacts WebSphere InterChange Server gérés par les outils de migration

Les outils de migration permettent d'effectuer automatiquement la migration des artefacts WebSphere InterChange Server.

Les artefacts pouvant faire l'objet d'une migration sont les suivants :

- Les **objets métier** deviennent des objets métier WebSphere Process Server.
- Les **mappes** deviennent des mappes WebSphere Process Server.
- Les **relations** deviennent des relations et des rôles WebSphere Process Server.
- Les **modèles de collaboration** deviennent des éléments BPEL et WSDL WebSphere Process Server.
- Les **objets de collaboration** deviennent des artefacts SCA WebSphere Process Server.
- Les **définitions de connecteurs** deviennent des artefacts SCA WebSphere Process Server.

Les resfacts source et artefacts WebSphere InterChange Server pouvant être configurés automatiquement par les outils de migration sous WebSphere Process Server sont les suivants :

- Pools DBConnection
- Bases de données de relations
- Entrées du programmeur

Les artefacts WebSphere InterChange Server qui ne sont pas traités par les outils de migration sont les suivants :

- Artefacts de tests de performances

API WebSphere InterChange Server prises en charge

Outre les outils de migration des artefacts source fournis par WebSphere Process Server et WebSphere Integration Developer, de nombreuses API WebSphere InterChange Server sont prises en charge. Les outils de migration fonctionnent avec les API WebSphere InterChange Server en conservant au maximum le code des fragments personnalisés lors de la migration.

Remarque : Ces API sont fournies uniquement pour assurer la prise en charge des applications WebSphere InterChange Server migrées, jusqu'à ce que ces dernières puissent être modifiées pour utiliser les nouvelles API de Process Server. Elles sont obsolètes et seront supprimées dans une édition ultérieure.

Les API WebSphere InterChange Server prises en charge dans Process Server sont répertoriées ci-après. Les fonctions remplies par ces API sous Process Server sont similaires à celles qui étaient offertes sous WebSphere InterChange Server. Consultez la documentation de WebSphere InterChange Server pour obtenir la description des fonctions de ces API.

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)
- executeStoredProcure(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)
- executeStoredProcedure(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, int, String, 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

Mappage de l'objet DataObject de WebSphere Process Server à partir d'un fichier XML WebSphere InterChange Server

Si vous utilisez les adaptateurs existants pour vous connecter à WebSphere Process Server, l'algorithme suivant vous permettra de mieux comprendre comment l'objet DataObject de WebSphere Process Server a été créé à partir d'un fichier XML WebSphere InterChange Server. Ces informations indiquent les valeurs choisies pour remplacer celles utilisées dans WebSphere InterChange Server, ainsi que leur emplacement.

Généralités

- Pour la définition de l'instruction de ChangeSummary, tous les paramétrages sont effectués via les interfaces API **markCreate/Update/Delete**.
- Pour la définition de l'instruction de ChangeSummary/EventSummary, les instructions **Create**, **Update** et **Delete** sont définies dans ChangeSummary et toutes les autres dans EventSummary.
- Pour obtenir l'instruction à partir de l'élément ChangeSummary :
 - Si isDelete a la valeur true, l'instruction utilisée est **Delete**.

Remarque : Si isDelete a la valeur true, la valeur de isCreate n'est pas prise en compte. Ce cas peut se présenter si la création a été marquée avant l'entrée de DataObject dans le concentrateur (et sa suppression), ou si la création et la suppression se sont toutes deux produites dans le concentrateur.

- Si isDelete a la valeur false et isCreate la valeur true, l'instruction utilisée est **Create**.
- Si isDelete et isCreate ont la valeur false, l'instruction utilisée est **Update**.
- Pour éviter qu'un objet de données soit associé à l'instruction **Create** au lieu de l'instruction **Update**, vous devez procéder comme suit (si la journalisation est activée) :
 - Interrompez la journalisation pendant la création de l'objet de données.
 - Réactivez la journalisation pour la mise à jour de l'objet de données (ou utilisez l'API **markUpdated**).

Chargement

Charge un fichier d'exécution XML WebSphere InterChange Server dans une instance WebSphere Business Integration BusinessGraph AfterImage.

- Une instance de l'élément BusinessGraph approprié est créée.
- La journalisation est activée pour l'élément ChangeSummary. Ainsi, les entrées ne seront pas supprimées si vous l'activez ultérieurement.

- La journalisation liée à l'élément `ChangeSummary` est interrompue pour éviter que des informations indésirables ne soient consignées dans le résumé des modifications.
- Les attributs de l'objet métier principal sont créés dans l'objet de données (voir la section "Traitement des attributs" ci-dessous).
- Si l'objet métier principal possède des objets enfants, ceux-ci font l'objet d'un traitement récursif.
- Les attributs de ces objets métier enfants sont créés dans l'objet de données (voir la section "Traitement des attributs" ci-dessous).
- L'instruction de l'objet métier principal est définie dans l'instruction principale de l'élément `BusinessGraph` et dans les récapitulatifs.
- L'instruction des objets métier enfants est définie dans les récapitulatifs.

Enregistrement

Enregistre une instance WebSphere Business Integration `BusinessGraph AfterImage` dans un fichier d'exécution XML WebSphere InterChange Server. Une exception est générée si l'élément `BusinessGraph` entrant n'est pas au format `AfterImage`.

- Une instance du document XML WebSphere InterChange Server est créée.
- Tous les objets de données inclus dans l'élément `ChangeSummary` sont traités comme s'ils se trouvaient à leur emplacement d'origine.

Remarque : Ce processus de restauration ne conserve pas l'ordre de la liste.

- L'instruction de l'objet métier principal est définie dans l'instruction principale de l'élément `BusinessGraph`.
- Les attributs de l'objet métier principal sont définis à partir de l'objet de données (voir la section "Traitement des attributs" ci-dessous).
- Si l'objet de données possède des objets de données enfants, ceux-ci font l'objet d'un traitement récursif.
- L'instruction des objets de données enfants est traitée de la manière suivante :
 - Si aucune instruction des éléments `EventSummary` ou `ChangeSummary` ne correspond à l'objet de données enfant, l'instruction prend la valeur "" (chaîne vide).
 - Si seul `EventSummary` contient une instruction, celle-ci est utilisée.
 - Si seul `ChangeSummary` contient une instruction, celle-ci est utilisée.
 - Si `ChangeSummary` et `EventSummary` contiennent chacun une instruction, la valeur indiquée dans `ChangeSummary` est prioritaire.
- Les attributs des objets de données enfants sont définis dans l'objet métier (voir la section "Traitement des attributs" ci-dessous).

Traitement des attributs

- Toutes les valeurs non mentionnées ci-dessous sont chargées et enregistrées en l'état.
- `ObjectEventId` est chargé dans/enregistré à partir de `EventSummary`.
- Pour **CxBlank** et **CxIgnore** :
 - Durant la phase de conversion de `BusinessObject` effectuée sous WebSphere Business Integration, **CxBlank** et **CxIgnore** sont définis/identifiés de la manière suivante :
 - **CxIgnore** : Non défini ou défini avec la valeur `null` Java ;
 - **CxBlank** : Valeur dépendant du type (voir tableau ci-dessous).
 - Durant la phase de conversion XML effectuée sous WebSphere InterChange Server, **CxBlank** et **CxIgnore** sont définis/identifiés de la manière suivante :

Tableau 1. Définition de CxBlank et de CxIgnore

Type	CxIgnore	CxBlank
Entier	Integer.MIN_VALUE	Integer.MAX_VALUE
Variable flottante	Float.MIN_VALUE	Float.MAX_VALUE
Double	Double.MIN_VALUE	Double.MAX_VALUE
Chaîne/date/texte long	"CxIgnore"	""
Objets métier enfants	(élément vide)	N/A

Méthodes recommandées pour le processus de migration de WebSphere InterChange Server

Les instructions suivantes sont fournies pour vous assister lors du développement des artefacts d'intégration pour WebSphere InterChange Server. En vous y conformant, vous pourrez effectuer la migration d'artefacts WebSphere InterChange Server vers WebSphere Process Server avec beaucoup plus de facilité.

Ces recommandations sont fournies uniquement pour vous guider lors du développement de nouvelles solutions d'intégration. Il est possible qu'elles ne s'appliquent pas au contenu existant ou doivent être adaptées dans certaines situations. Dans ce cas, essayez de dévier le moins possible des procédures décrites, afin d'alléger les modifications requises lors de la migration des artefacts. Notez que ces instructions ne sont pas nécessairement les méthodes conseillées pour le développement des artefacts WebSphere InterChange Server en général. Elles se rapportent uniquement aux facteurs pouvant affecter le déroulement des migrations d'artefacts effectuées ultérieurement.

Développement général

Certaines considérations sont valides pour la plupart des artefacts d'intégration. En règle générale, les artefacts utilisant les fonctionnalités offertes par les outils et conformes aux modèles de métadonnées appliqués par ce dernier sont les plus simples à migrer. En revanche, la migration des artefacts possédant des extensions et des dépendances externes importantes peut nécessiter une intervention manuelle plus conséquente.

La liste suivante présente les méthodes recommandées facilitant les opérations de migration dans le cadre du développement général de solutions basées sur WebSphere InterChange Server.

- Utilisez WebSphere InterChange Server pour la création de solutions permettant l'intégration automatique de processus en temps réel.
- Documentez la conception du système et des composants.
- Utilisez les outils de développement pour modifier les artefacts d'intégration.
- Faites appel aux méthodes recommandées pour définir des règles à l'aide des outils et des fragments Java.

Aussi évident que cela puisse paraître, les solutions d'intégration doivent respecter l'architecture et le modèle de programmation fournis par WebSphere InterChange Server. Ce modèle est le plus approprié aux solutions permettant l'intégration automatique de processus en temps réel. Par ailleurs, chaque composant d'intégration de WebSphere InterChange Server joue un rôle bien défini dans l'architecture. Plus vous déviez de ce modèle, plus la migration des contenus vers les artefacts WebSphere Process Server appropriés deviendra complexe.

Le fait de documenter la conception du système est également un facteur crucial pour la réussite des projets de migration futurs. Vous devez faire en sorte de décrire la conception et l'architecture d'intégration, y compris les besoins en termes de fonctionnalités et de qualité de service, les interdépendances entre des artefacts partagés par plusieurs projets et les décisions de conception prises

lors de la phase de déploiement. Ces informations vous permettront de mieux analyser le système et d'alléger les tâches devant être effectuées après la migration.

Il est essentiel d'utiliser uniquement les outils de développement fournis pour créer, configurer et modifier des définitions d'artefacts. Evitez tout traitement manuel des métadonnées (modification directe des fichiers XML, etc.) qui risquerait d'altérer les artefacts devant faire l'objet d'une migration.

Lorsque vous développez du code Java pour les modèles de collaboration, les mappes, les utilitaires de partage de code et d'autres composants, tenez compte des points suivants :

- Utilisez uniquement les interfaces API publiées.
- Utilisez l'éditeur d'activités.
- Utilisez des adaptateurs pour accéder aux systèmes EIS.
- Evitez d'inclure des dépendances externes dans les fragments de code Java.
- Respectez les normes de développement J2EE, de manière à assurer la portabilité du code.
- Ne générez pas d'unités d'exécution. N'utilisez pas de primitives de synchronisation pour les unités d'exécution. Si vous y êtes obligé, vous devrez les convertir lors de la migration pour permettre l'utilisation de beans asynchrones.
- N'effectuez aucune opération d'entrée-sortie sur le disque à l'aide de java.io.* Utilisez JDBC pour stocker toutes les données.
- N'effectuez aucune fonction pouvant être réservée à un conteneur EJB (E-S de port, chargement de classes, chargement de bibliothèques natives, etc.). Si vous y êtes obligé, vous devrez convertir ces fragments manuellement lors de la migration pour permettre l'utilisation des fonctions du conteneur EJB.

Utilisez uniquement les interfaces API mentionnées dans la documentation du produit concernant les artefacts. Celles-ci sont présentées en détail dans les guides de développement de WebSphere InterChange Server. Bien que des API de compatibilité soient souvent fournies dans WebSphere Process Server, seules les API mentionnées seront incluses. WebSphere InterChange Server comprend un grand nombre d'interfaces internes pouvant intéresser les développeurs, mais rien ne garantit leur prise en charge ultérieure.

Lors de la conception de la logique métier et des règles de transformation dans les mappes et les modèles de collaboration, utilisez le plus possible l'éditeur d'activités. Cet outil permet de s'assurer que la logique est décrite via des métadonnées pouvant être converties plus facilement vers les nouveaux artefacts. Faites appel à la fonction "My Collections" de l'éditeur d'activités pour toutes les opérations que vous souhaitez réutiliser dans les outils. Evitez d'inclure des fichiers .jar correspondant à des bibliothèques de code développées sur site dans le chemin de classe de WebSphere InterChange Server ; vous seriez contraint de les faire migrer manuellement.

Pour tout fragment Java pouvant nécessiter un développement, il est recommandé de limiter le code utilisé aux instructions de base définissant l'ordre des scripts (évaluations, opérations et calculs de base, formatage des données, conversions de types, etc.). Si une logique d'application plus complexe est requise, il est préférable de l'encapsuler dans des EJB fonctionnant sous WebSphere Application Server et d'utiliser des appels de service Web pour l'appeler à partir de WebSphere InterChange Server. Utilisez les bibliothèques JDK standard plutôt que des bibliothèques tierces ou externes devant être migrées séparément. Regroupez tous les éléments connexes dans un fragment de code unique et évitez d'utiliser une logique basée sur des contextes de connexion et de transaction répartis sur plusieurs fragments. Par exemple, en ce qui concerne les bases de données, le code permettant d'obtenir une connexion, de démarrer et d'arrêter une transaction ou de fermer la connexion doit être regroupé dans un seul fragment.

D'une manière générale, vérifiez que le code conçu pour fournir l'interface avec un système EIS est intégré à des adaptateurs et non à des mappes ou à des modèles de collaboration. Cette méthode est recommandée en matière de conception d'architecture. Elle permet également d'éviter que les contraintes

liées à la configuration requise pour des bibliothèques tierces et d'autres considérations ne doivent être intégrées au niveau du code (gestion des connexions et implémentations JNI (Java Native Interface) possibles.

Utilisez un traitement des exceptions rendant le code aussi sécurisé que possible. Faites en sorte que le code puisse fonctionner dans un environnement de serveur d'applications J2EE, même s'il fonctionne actuellement dans un environnement J2SE. Conformez-vous aux pratiques de développement J2EE : évitez les variables statiques, la génération d'unités d'exécution et les E-S sur disque. Ces pratiques ont fait leurs preuves et s'avèrent particulièrement efficaces en matière de portabilité.

Utilitaires de code

Comme indiqué précédemment, le développement de bibliothèques de code destinées à être partagées par plusieurs artefacts d'intégration de l'environnement WebSphere InterChange Server n'est pas recommandé. Si des fragments de code doivent être réutilisés dans plusieurs artefacts d'intégration, il est préférable de faire appel à la fonction "My Collections" de l'éditeur d'activités. Pensez également à encapsuler la logique dans des EJB fonctionnant sous WebSphere Application Server et exécutés sous WebSphere InterChange Server au moyen d'appels de service. Bien que certaines bibliothèques de code courant puissent s'exécuter correctement sous WebSphere Process Server, l'utilisateur est responsable de leur migration.

Pools de connexion aux bases de données

Les pools de connexion aux bases de données définis par l'utilisateur sont très utiles dans les mappes et les modèles de collaboration, notamment pour les recherches de données simples et la gestion des états portant sur plusieurs instances de processus. Sous WebSphere InterChange Server, un pool de connexion aux bases de données se présente comme une ressource JDBC standard de WebSphere Process Server. Le fonctionnement de base est identique, mais il peut exister des différences dans la façon dont les connexions et les transactions sont gérées.

Pour garantir une portabilité optimale, évitez les situations impliquant des transactions de bases de données actives sur plusieurs noeuds de fragments Java dans un modèle de collaboration ou une mappe. Par exemple, le code permettant d'obtenir une connexion, de démarrer et d'arrêter une transaction ou de fermer la connexion doit être regroupé dans un seul fragment.

Objets métier

Les points essentiels devant être pris en compte lors du développement d'objets métier sont les suivants : utilisation exclusive des outils fournis pour la configuration des artefacts, définition explicite du type et de la longueur des attributs de données, utilisation exclusive des API documentées.

Les objets métier de WebSphere Process Server sont basés sur des objets SDO (Service Data Objects) dont les attributs sont définis de façon très spécifique. Les objets métier sous WebSphere InterChange Server et les adaptateurs ne présentant pas les mêmes exigences de précision, il arrive que les utilisateurs associent des données de type chaîne à des attributs d'une autre nature. Pour éviter tout incident sous WebSphere Process Server, soyez le plus explicite possible dans la spécification des types de données.

Etant donné que les objets métier de WebSphere Process Server peuvent être sérialisés en cours d'exécution lorsqu'ils sont transférés d'un composant à l'autre, il est important de définir explicitement la longueur des attributs afin de minimiser l'utilisation des resfacs source système. Pour cette raison, évitez, par exemple, d'utiliser la longueur maximum d'attribut de chaîne (255 caractères). Ne définissez pas non plus une valeur de 0 ; la valeur par défaut de 255 caractères serait alors prise en compte. Indiquez la longueur exacte requise pour les attributs.

Etant donné que les règles XSD NCName s'appliquent aux noms des attributs d'objet métier sous WebSphere Process Server, n'utilisez pas les espaces ni les deux points (:) dans ces noms. Ces caractères

ne sont pas reconnus par WebSphere Process Server pour les noms d'attributs des objets métier. Renommez les attributs des objets métier avant la migration.

Si un objet métier contient un tableau, notez qu'il est possible que l'ordre d'origine ne soit pas respecté lors de son indexation dans les mappes et/ou les relations. Il n'est pas garanti que l'ordre d'indexation soit conservé lorsque de la migration sous WebSphere Process Server, particulièrement si des entrées sont supprimées.

Comme indiqué précédemment, il est primordial de modifier les définitions des objets métier exclusivement à l'aide de l'outil fourni, et d'utiliser uniquement les API publiées pour les objets métier des artefacts d'intégration.

Modèles de collaboration

La plupart des instructions décrites plus haut s'appliquent également au développement de modèles de collaboration.

Pour vous assurer que les processus sont décrits de façon appropriée par les métadonnées, utilisez toujours l'outil de conception de processus fourni pour créer et modifier les modèles de collaboration, et évitez d'éditer directement les fichiers de métadonnées. Utilisez le plus possible l'éditeur d'activités pour optimiser l'utilisation des métadonnées décrivant la logique requise.

Pour alléger les tâches manuelles nécessaires après la migration, utilisez uniquement les API documentées dans les modèles de collaboration. Evitez l'utilisation de variables statiques. Utilisez plutôt des variables non statiques et des propriétés de collaboration pour gérer les conditions requises par la logique métier. Evitez l'utilisation de qualificatifs Java finaux, transitoires et natifs dans les fragments Java. Ceux-ci ne peuvent pas être appliqués dans les fragments Java BPEL résultant de la migration des modèles de collaboration.

Pour assurer une portabilité du code optimale, évitez d'utiliser des appels de fin de connexion explicites et la mise entre parenthèses de transactions (validations et annulations) explicites pour les pools de connexion aux bases de données définis par l'utilisateur. Utilisez plutôt les fonctions de nettoyage implicite et de mise entre parenthèses implicite des transactions gérées par le conteneur. Evitez les situations impliquant des transactions et des connexions systèmes actives sur plusieurs noeuds de fragments Java dans un modèle de collaboration. Cela s'applique à toutes les connexions à un système externe ainsi qu'aux pools de connexion aux bases de données définis par l'utilisateur. Comme indiqué précédemment, les opérations impliquant un service EIS externe doivent être gérées au sein d'un adaptateur, et le code lié aux opérations portant sur les bases de données doit être contenu dans un fragment de code unique. Cela peut s'avérer nécessaire dans une collaboration qui, lorsqu'elle est présentée sous la forme d'un composant de processus métier BPEL, peut être déployée sélectivement en tant que flux interruptible. Dans ce cas, le processus peut comporter plusieurs transactions distinctes ; seules les informations sur les variables d'état et les variables globales sont alors transmises entre les activités. Le contexte de toutes les connexions système ou transactions associées impliquées par ces transactions de processus serait perdu.

N'utilisez pas de caractères spéciaux dans les noms de propriétés des modèles de collaboration. Une fois ces noms migrés en noms de propriétés BPEL, ces caractères ne seront pas reconnus. Renommez les propriétés de manière à supprimer ces caractères avant de procéder à la migration. Vous éviterez ainsi la création d'erreurs de syntaxe BPEL lors de la migration.

Ne faites pas référence à des variables en utilisant le terme "this". Par exemple, au lieu d'utiliser "this.inputBusObj", utilisez "inputBusObj".

Définissez la portée des variables au niveau des classes et non au niveau des scénarios. Une portée limitée à un scénario n'est pas conservée lors de la migration.

Initialisez toutes les variables déclarées dans les fragments Java avec une valeur par défaut. Exemple : "Object myObject = null;". Assurez-vous que toutes les variables sont initialisées lors de la déclaration, avant la migration.

Vérifiez qu'aucune instruction d'importation Java ne se trouve dans les sections modifiables par l'utilisateur des modèles de collaboration. Dans la définition de ces derniers, utilisez les zones d'importation pour indiquer les packages Java à importer.

Mappes

La plupart des instructions décrites pour les modèles de collaboration s'appliquent également aux mappes.

Pour vous assurer que les mappes sont décrites de façon appropriée par les métadonnées, utilisez toujours l'outil de conception fourni pour créer et modifier les mappes, et évitez d'éditer directement les fichiers de métadonnées. Utilisez le plus possible l'éditeur d'activités pour optimiser l'utilisation des métadonnées décrivant la logique requise.

Utilisez une sous-mappe lorsque vous faites référence à des objets métier enfant dans une mappe.

Évitez d'utiliser du code Java comme valeur d'une instruction SET : cette opération n'est pas valide sous WebSphere Process Server. Utilisez plutôt des constantes. Par exemple, la valeur définie est "xml version=" + "1.0" + " encoding=" + "UTF-8" n'est pas reconnue par WebSphere Process Server. Indiquez "xml version=1.0 encoding=UTF-8" avant d'effectuer la migration.

Pour alléger les tâches manuelles nécessaires après la migration, utilisez uniquement les API documentées dans les modèles de collaboration. Évitez l'utilisation de variables statiques. Utilisez plutôt des variables non statiques et des propriétés de collaboration pour gérer les conditions requises par la logique métier. Évitez l'utilisation de qualificatifs Java finaux, transitoires et natifs dans les fragments Java.

Si un objet métier contient un tableau, notez qu'il est possible que l'ordre d'origine ne soit pas respecté lors de son indexation dans les mappes. Il n'est pas garanti que l'ordre d'indexation soit conservé lorsque de la migration sous WebSphere Process Server, particulièrement si des entrées sont supprimées.

Pour assurer une portabilité du code optimale, évitez d'utiliser des appels de fin de connexion explicites et la mise entre parenthèses de transactions (validations et annulations) explicites pour les pools de connexion aux bases de données définis par l'utilisateur. Utilisez plutôt les fonctions de nettoyage implicite et de mise entre parenthèses implicite des transactions gérées par le conteneur. Évitez les situations impliquant des transactions et des connexions systèmes actives sur des fragments Java répartis sur plusieurs noeuds de transformation. Cela s'applique à toutes les connexions à un système externe ainsi qu'aux pools de connexion aux bases de données définis par l'utilisateur. Comme indiqué précédemment, les opérations impliquant un service EIS externe doivent être gérées au sein d'un adaptateur, et le code lié aux opérations portant sur les bases de données doit être contenu dans un fragment de code unique.

Relations

Bien que les définitions de relations puissent être migrées pour être utilisées sous WebSphere Process Server, n'oubliez pas que le schéma et les données d'instances des tables de relations peuvent être réutilisés par ce même programme, tout comme ils peuvent être partagés par WebSphere InterChange Server et WebSphere Process Server.

En ce qui concerne les relations, les points cruciaux à prendre en compte sont les suivants : vous devez faire appel uniquement à l'outil fourni pour configurer les composants associés, et vous ne devez utiliser que les API publiées pour les relations incluses dans les artefacts d'intégration.

Les définitions de relations ne doivent être modifiées qu'à l'aide de l'outil de conception de relations fourni. En outre, seul WebSphere InterChange Server doit être utilisé pour configurer le schéma des relations, qui est généré automatiquement lors du déploiement des définitions de relations. Ne modifiez pas le schéma des tables de relations directement à l'aide d'outils de base de données ou de scripts SQL.

En outre, si vous devez modifier manuellement les données d'instances des relations faisant partie de ce schéma, assurez-vous d'utiliser les fonctions de l'outil de conception approprié.

Comme indiqué précédemment, vous ne devez utiliser que les API publiées pour les relations incluses dans les artefacts d'intégration.

Clients Access Framework

Ne développez aucun nouveau client utilisant les API IDL CORBA. Celles-ci ne sont pas prises en charge par WebSphere Process Server.

Migration vers WebSphere Integration Developer à partir de WebSphere MQ Workflow

WebSphere Integration Developer fournit les outils nécessaires pour migrer à partir de WebSphere MQ Workflow.

L'Assistant de migration vous permet de convertir les définitions FDL des processus métier que vous avez exportés depuis le composant Buildtime de WebSphere MQ Workflow vers les artefacts correspondants de Business Process Choreographer (BPC). Les artefacts BPC générés comprennent les définitions XMLSchema des objets métier, les définitions WSDL, BPEL, et les définitions TEL.

L'outil de conversion nécessite une définition FDL sémantiquement complète d'un modèle de processus que vous exportez de WebSphere MQ Workflow Buildtime avec l'option **export deep**. Cette option garantit que toutes les spécifications de données, programme et sous-processus nécessaires sont comprises. Assurez-vous également que chaque définition de serveur d'exécution défini par l'utilisateur (UPES) référencée dans votre modèle de processus WebSphere MQ Workflow est aussi sélectionnée lorsque vous exportez FDL à partir de WebSphere MQ Workflow Buildtime.

Remarque : L'Assistant de migration ne permet pas de migrer les composants suivants :

- des instances d'exécution de WebSphere MQ Workflow,
- des applications de programme appelées par un agent d'exécution de programme (PEA) WebSphere MQ Workflow ou un serveur d'exécution de processus (PES pour z/OS) WebSphere MQ Workflow.

Préparation de la migration dans WebSphere MQ Workflow

Avant de migrer WebSphere Integration Developer depuis WebSphere MQ Workflow, vous devez d'abord vérifier que vous avez bien préparé votre environnement.

L'étendue des mappages dépend de la manière dont vous suivez les recommandations suivantes pour la migration :

- Vérifiez que les activités des programmes FDL sont associées à un serveur d'exécution de processus (UPES) défini par l'utilisateur, s'il ne s'agit pas d'activités de **personnel** pures.
- Vérifiez que les affectations de personnel pour les activités des programmes WebSphere MQ Workflow sont compatibles avec les **instructions de personnel** par défaut de TEL.
- Utilisez des noms courts et simples pour améliorer la lisibilité des modèles de processus migrés. Notez que les noms FDL peuvent ne pas être acceptés comme noms BPEL. L'Assistant de migration vous aide à convertir automatiquement les noms FDL en noms BPEL valides.

L'Assistant de migration produit des constructions BPE dont la syntaxe est correcte même pour les structures WebSphere MQ Workflow qui ne peuvent pas être migrées (activités des programmes PEA ou PES, certaines affectations de personnel dynamiques, etc) et qui doivent être adaptées manuellement aux artefacts BPE exécutables.

Le tableau suivant détaille les règles de correspondance applicables :

Tableau 2. Règles de correspondance

WebSphere MQ Workflow	Business Process Choreographer
Processus	<i>Processus avec mode d'exécution longRunning; Liens partenaires pour les interfaces entrantes et sortantes du processus</i>
Source et Récepteur	<i>Variables pour l'entrée et la sortie des processus ; activité Réception et activité Réponse</i>
Activité des programmes	<i>Activité Appeler</i>
Activité des processus	<i>Activité Appeler</i>
Activité Vider	<i>Activité Vider</i>
Bloc	<i>Etendue avec activités BPEL imbriquées</i>
Condition de sortie de l'activité	<i>Activité Pendant (qui contient l'activité proprement dite)</i>
Condition de démarrage de l'activité	<i>Condition Jointure de l'activité</i>
Affectation de personnel de l'activité	<i>Activité Tâche manuelle</i>
Conteneur d'entrée et conteneur de sortie de l'activité	<i>Variables utilisées pour spécifier l'entrée et la sortie de l'activité Appeler</i>
Connecteur de contrôle ; Condition de transition	<i>Lier; Condition de transition</i>
Connecteur de données	<i>Activité Affecter</i>
Conteneur de données globales	<i>Variable</i>

Si possible, commencez le processus de migration avec des petits projets. L'Assistant de migration va simplifier la conversion de vos modèles de processus WebSphere MQ Workflow en modèles de processus Business Process Editor mais sachez que les processus ne peuvent pas être mappés deux par deux quand vous créez un nouveau modèle de programmation. Les syntaxes des langages de spécification des processus sous-jacents (FDL et BPEL) sont parfois communes mais pas totalement identique. Business Process Editor n'aurait aucune utilité sinon. Les services Web constituent une nouvelle technologie prometteuse appelée à remplacer des technologies dépassées.

D'une manière générale, vous devez toujours examiner et modifier, si possible, les artefacts créés. Un effort supplémentaire doit être fait pour réussir la migration ou finir la tâche de migration.

Migration de définitions WebSphere MQ Workflow avec l'Assistant de migration

L'Assistant de migration vous permet de convertir les définitions FDL des processus métier que vous avez exportés depuis le composant Buildtime de WebSphere MQ Workflow vers les artefacts correspondants de Business Process Choreographer (BPC). Les artefacts BPC générés comprennent les définitions XMLSchema des objets métier, les définitions WSDL, BPEL, et les définitions TEL.

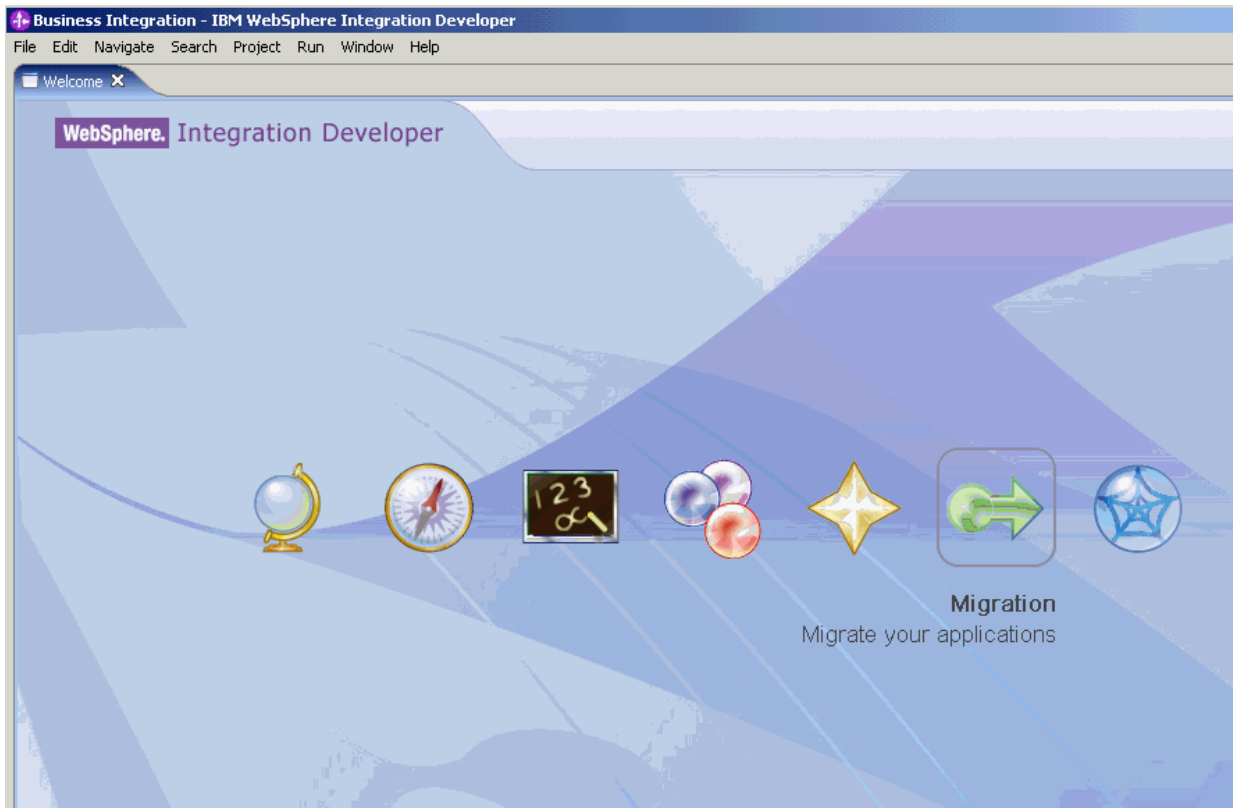
Remarque : L'Assistant de migration ne permet pas de migrer les composants suivants :

- instances d'exécution de WebSphere MQ Workflow
- Les applications appelées par un agent d'exécution de programme (PEA) de WebSphere MQ Workflow ou par un serveur d'exécution de processus de WebSphere MQ Workflow (PES pour z/OS)

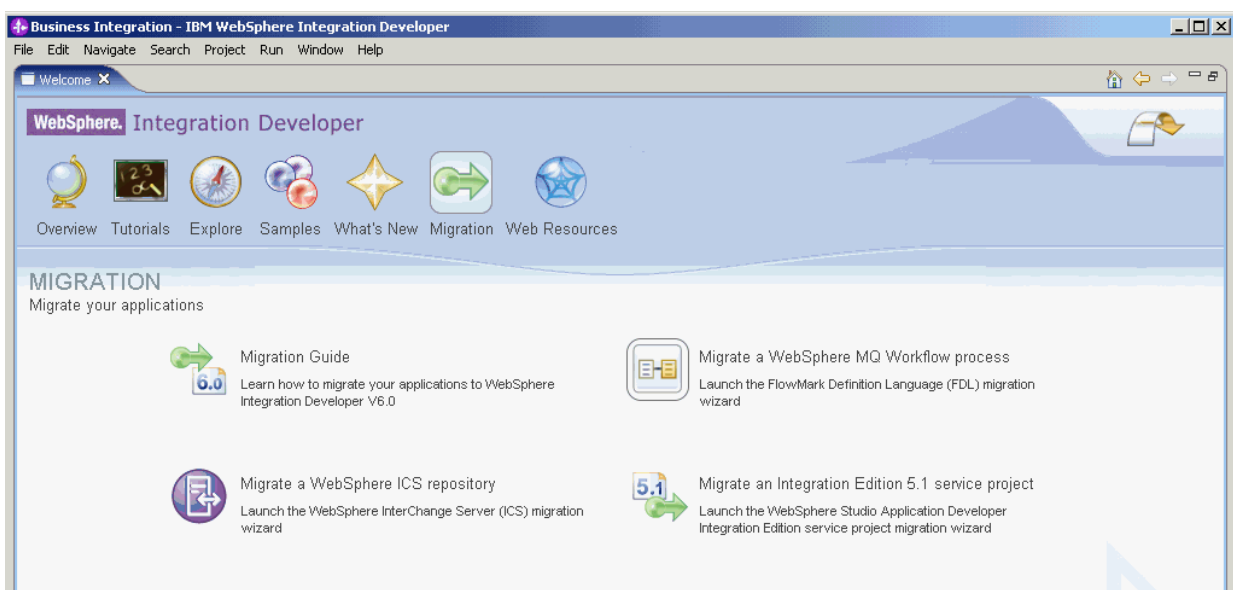
Suivez les étapes suivantes pour utiliser l'Assistant de migration et migrer vos artefacts WebSphere MQ Workflow :



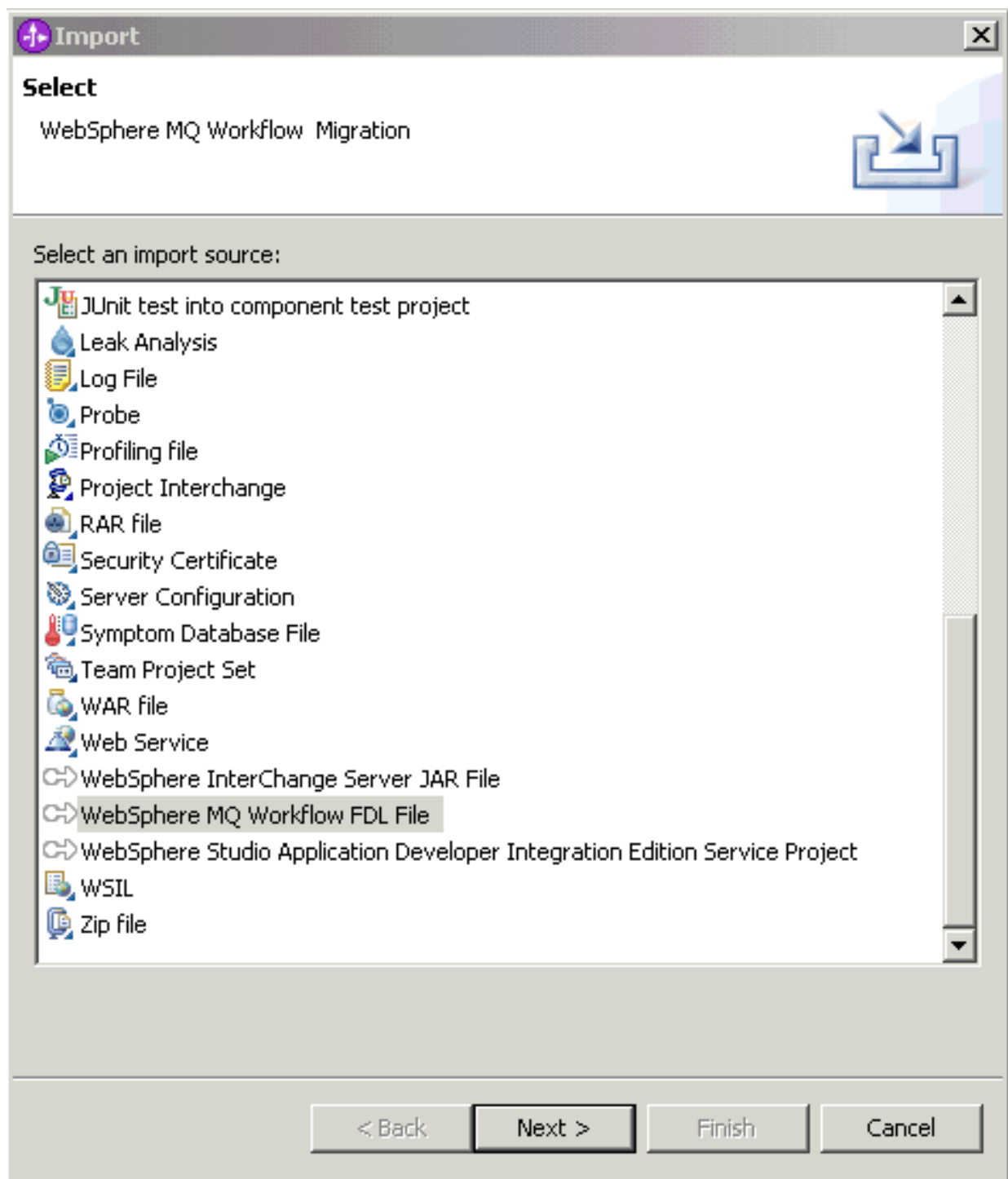
1. Dans la page d'accueil, cliquez sur l'icône pour ouvrir la page Migration.



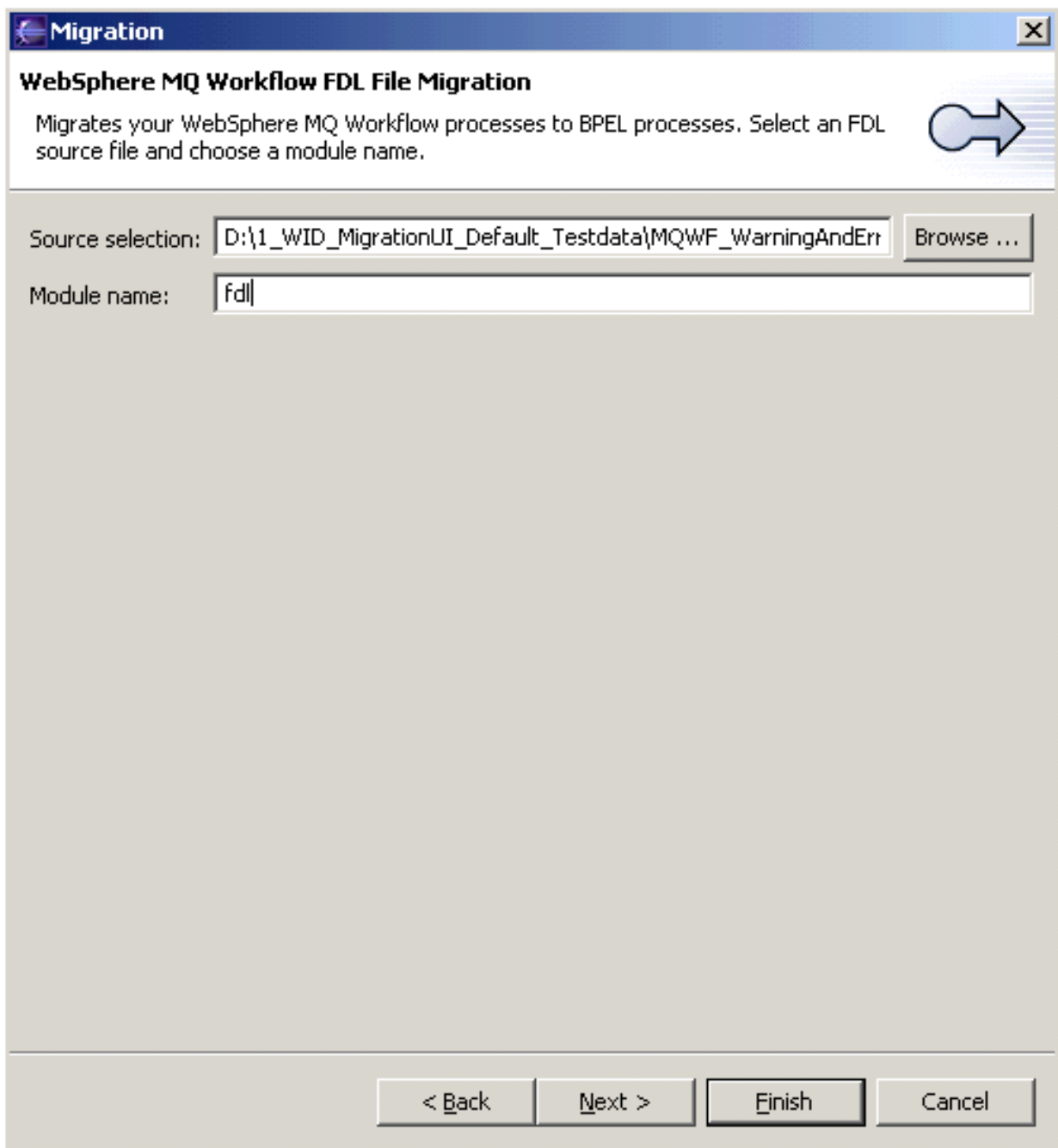
2. Dans la page Migration, sélectionnez l'option Migrer un processus WebSphere MQ Workflow.



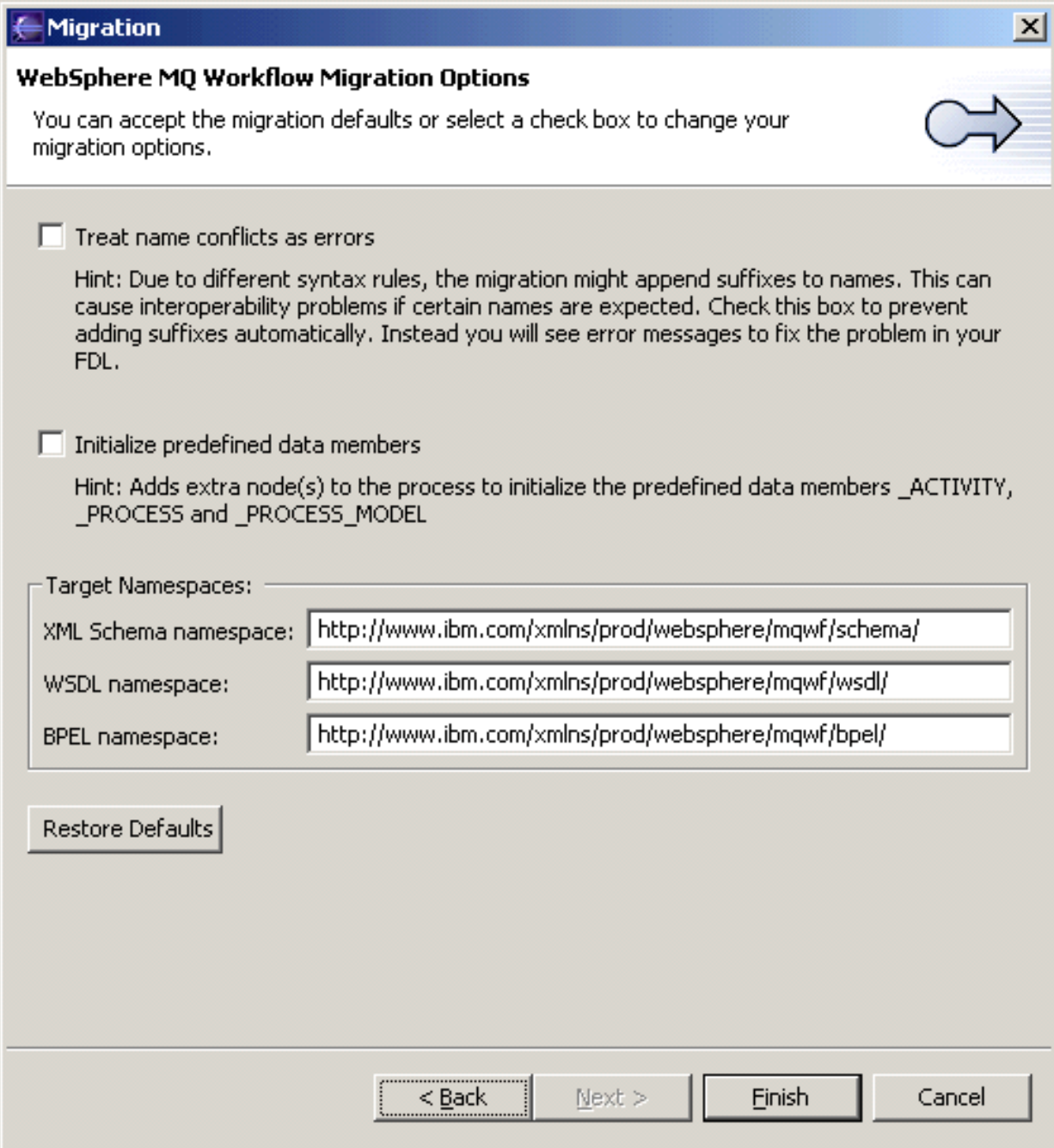
(Remarque : vous pouvez aussi ouvrir l'Assistant de migration en cliquant sur **Fichier** → **Importer un fichier** → **WebSphere MQ Workflow FDL** :



3. L'Assistant de migration apparaît. Pour entrer le nom du fichier .fdl dans le champ **Sélection de la source**, cliquez sur le bouton **Parcourir** et recherchez le fichier. Entrez le nom du module dans le champ approprié. Cliquez sur **Suivant**.



4. La page Options de migration apparaît. Vous pouvez accepter les options de migration par défaut ou sélectionner une case à cocher pour modifier une option. La sélection de la case à cocher **Traiter les conflits de noms comme des erreurs** empêche l'ajout automatique de suffixes qui peuvent entraîner des erreurs d'interopérabilité. La case à cocher **Initialiser les membres de données prédéfinis** ajoute des noeuds supplémentaires au processus afin d'initialiser les membres de données prédéfinis.



The image shows a Windows-style dialog box titled "Migration" with a close button (X) in the top right corner. The main title is "WebSphere MQ Workflow Migration Options". Below the title, there is a text block: "You can accept the migration defaults or select a check box to change your migration options." To the right of this text is a large blue arrow pointing right. There are two checkboxes: "Treat name conflicts as errors" and "Initialize predefined data members". Each checkbox has a hint text below it. The "Target Namespaces" section contains three text boxes for "XML Schema namespace:", "WSDL namespace:", and "BPDL namespace:", each with a default URL. A "Restore Defaults" button is located below these text boxes. At the bottom of the dialog are four buttons: "< Back", "Next >", "Finish", and "Cancel".

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:

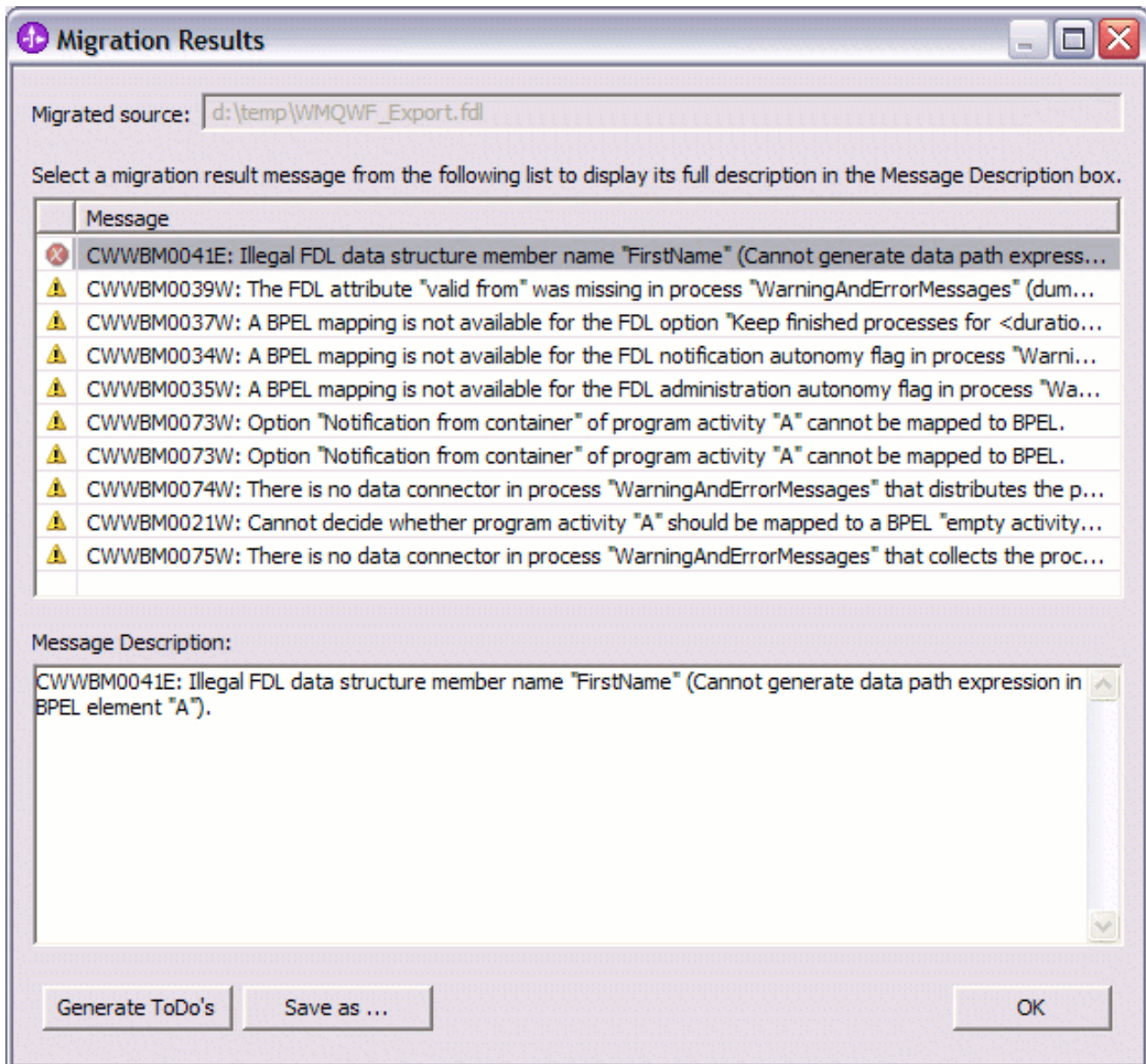
BPDL namespace:

Cliquez sur **Terminer**.

Vérification de la migration de WebSphere MQ Workflow

Lorsque l'exécution de l'assistant d'installation est terminée, une liste répertoriant les erreurs, les avertissements et/ou les messages informatifs s'affiche. Vous pouvez utiliser ces messages pour vérifier la migration de WebSphere MQ Workflow.

La page suivante apparaît lorsque l'exécution de l'Assistant de migration est terminée :



La fenêtre Résultats de la migration contient une liste des messages de migration. Cliquez sur un message pour afficher une description complète des détails de la migration. A partir de cette liste de messages, vous pouvez créer, le cas échéant, une liste de points à corriger en cliquant sur le bouton **Générer Actions à accomplir**.

Restrictions du processus de migration (à partir de WebSphere MQ Workflow)

Certaines restrictions s'appliquent au processus de migration de WebSphere MQ Workflow.

La migration de FDL générera des actions d'appel pour les activités UPES et les WSDL correspondants. Cependant, l'environnement d'exécution diffère beaucoup entre IBM WebSphere MQ Workflow et IBM WebSphere Process Server au niveau des techniques utilisées pour mettre en corrélation les messages d'appel et leurs réponses. Ainsi, le BPEL généré ne peut pas s'exécuter correctement si aucune couche de compatibilité UPES n'est disponible.

Migration d'artefacts source vers WebSphere Integration Developer à partir de WebSphere Studio Application Developer Integration Edition

Les artefacts source peuvent être migrés à partir de WebSphere Studio Application Developer Integration Edition vers WebSphere Integration Developer. La migration des artefacts source dans une application implique leur migration vers le nouveau modèle de programmation WebSphere Integration Developer afin que les nouvelles fonctionnalités puissent être utilisées. L'application peut alors être redéployée et installée sur le serveur WebSphere Integration Developer version 6.0.

Plusieurs fonctions de WebSphere Business Integration Server Foundation 5.1 ont été déplacées dans la base de WebSphere Application Server 6.0. Le site suivant donne des conseils pour migrer ces fonctions : http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.websphere.base.doc/info/aes/ae/rins_migratepme.html?resultof=%22%70%6d%65%22%20

Pour migrer complètement un projet de service WebSphere Studio Application Developer Integration Edition, réalisez deux tâches fondamentales :

1. Utilisez l'Assistant de migration pour migrer automatiquement les artefacts vers le projet de module Business Integration.
2. Utilisez WebSphere Integration Developer pour terminer la migration manuellement. Cela demande de corriger tous les codes Java ne pouvant pas être migrés automatiquement et de reconnecter les artefacts migrés.

Remarque : La migration d'exécution (chemin de mise à niveau) ne sera pas fournie dans WebSphere Process Server 6.0 et ce mode de migration des artefacts source sera la seule option pour migrer les projets de service WebSphere Studio Integration Edition dans la version 6.0.

Chemins de migration pris en charge pour la migration d'artefacts source

Avant de débiter la migration d'artefacts source à partir de WebSphere Studio Application Developer Integration Edition, vérifiez les chemins de migration pris en charge par WebSphere Integration Developer.

L'Assistant de migration permet de migrer un projet de service WebSphere Studio Application Developer Integration Edition version 5.1 (ou supérieure) à la fois. Il ne migrera *pas* un espace de travail entier.

L'Assistant de migration ne migre pas les applications binaires. Il migre uniquement les artefacts source trouvés dans un projet de service WebSphere Studio Application Developer Integration Edition.

Préparation des artefacts source pour la migration

Avant de migrer des artefacts source vers WebSphere Integration Developer depuis WebSphere Studio Application Developer Integration Edition, vous devez vérifier que vous avez bien préparé votre environnement.

Les étapes suivantes décrivent comment préparer votre environnement avant de migrer les artefacts source vers WebSphere Integration Developer depuis WebSphere Studio Application Developer Integration Edition.

1. Vérifiez que vous disposez d'une copie de sauvegarde de tout l'espace de travail de la version 5.1 avant d'effectuer la migration.
2. Relisez la section relative à la migration dans le Centre de documentation de Rational Application Developer pour déterminer la meilleure manière de migrer les projets non WBI dans votre espace de travail : <http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.rad.migration.doc/topics/tmigratefrom51x.html>
3. Relisez la section sur les services Web dans le Centre de documentation de Rational Application Developer pour obtenir des informations sur les fonctionnalités de service Web fournies par Rational

Application Developer :

<http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.webservices.rad.nav.doc/developingweb.html>

4. Vérifiez que toutes les fonctions requises de WebSphere Integration Developer sont activées. Si elles ne sont pas activées, vous risquez de ne pas voir les options de menu décrites ci-après. Pour activer les fonctions importantes :
 - Dans WebSphere Integration Developer, sélectionnez l'option **Fenêtre** puis sélectionnez **Préférences**.
 - Ouvrez l'**espace de travail** puis sélectionnez la catégorie **Fonctionnalités**.
 - Sélectionnez toutes les fonctionnalités dans les catégories suivantes :
 - Advanced J2EE
 - Enterprise Java
 - Integration Developer
 - Java Developer
 - Web Developer (standard)
 - Web Service Developer
 - XML Developer
 - Cliquez sur **OK**.
5. Utilisez un nouveau répertoire d'espace de travail pour WebSphere Integration Developer. Il est déconseillé d'ouvrir WebSphere Integration Developer dans un ancien espace de travail WebSphere Studio Application Developer Integration Edition qui contient des projets de service car ces projets doivent d'abord être migrés dans un format lisible par WebSphere Integration Developer. Les étapes à suivre pour le faire sont les suivantes :
 - a. Copiez tous les projets autres que des services depuis l'ancien espace de travail vers le nouveau. *Ne copiez pas* les projets version 5.1 EJB, Web et EAR générés lors de la création du code de déploiement pour un projet de service 5.1. Le nouveau code de déploiement 6.0 sera régénéré automatiquement quand le module BI sera compilé.
 - b. Ouvrez WebSphere Integration Developer dans l'espace de travail vide et importez tous les projets autres que des services en cliquant sur **Fichier** → **Importer** → **Projet existant dans l'espace de travail** puis sélectionnez les projets que vous avez copiés dans le nouvel espace de travail.
 - Si le projet est un projet J2EE, migrez-le vers le niveau 1.4 en utilisant l'Assistant de migration de Rational Application Developer :
 - 1) Avec le bouton droit, cliquez sur le projet et sélectionnez **Migration** → **Assistant de migration J2EE**.
 - 2) Lisez les avertissements sur la première page et cliquez sur **Suivant**.
 - 3) Vérifiez que le **projet J2EE** est coché dans la liste des projets. Laissez cochées les options **Migrer la structure de projet** et **Migrer le niveau de spécification J2EE**. Sélectionnez J2EE version **1.4** et **Serveur de processus WebSphere v6.0 du serveur cible**.
 - 4) Sélectionnez toutes les autres options appropriées pour votre projet J2EE et cliquez sur **Terminer**. Si cette étape s'accomplit normalement, un message vous indiquera que **la migration a réussi**.
 - 5) S'il existe des erreurs dans le projet J2EE après la migration, vous devrez supprimer toutes les entrées de chemin de classe qui référencent des fichiers jar v5 ou des bibliothèques et ajouter les bibliothèques **Bibliothèque système JRE** et **Cible de serveur WPS** dans le chemin de classe (voir ci-après). Ceci devrait résoudre la plupart de ces erreurs.
 - Pour les projets EJB WebSphere Business Integration contenant des données CMM (Extended Messaging) ou CMP/A (Container Managed Persistence over Anything), les fichiers de descripteurs d'extension jar EJB IBM doivent être migrés une fois le projet 5.1 importé dans l'espace de travail 6.0. Voir "Migration de projets EJB WebSphere Business Integration" pour plus d'informations.

- Modifiez le chemin de classe pour chaque projet non-service importé dans l'espace de travail. Pour ajouter les bibliothèques JRE et WebSphere Process Server dans le chemin de classe, avec le bouton droit, cliquez sur le projet importé et sélectionnez **Propriétés**. Allez à l'entrée **Chemin de compilation Java** et sélectionnez l'onglet **Bibliothèques**. Ensuite, exécutez les opérations suivantes :
 - 1) Sélectionnez **Ajouter une bibliothèque** → **Bibliothèque système JRE** → **Autre JRE - WPS Server v6.0 JRE** → **Terminer**.
 - 2) Ensuite, sélectionnez **Ajouter une bibliothèque** → **Cible de serveur WPS** → **Configurer le chemin de classe du serveur WPS** → **Terminer**.
- 6. Pour obtenir les meilleurs résultats, avant d'exécuter l'Assistant de migration, sélectionnez l'option de menu **Projet** et vérifiez que la case **Compiler automatiquement** n'est PAS cochée. WebSphere Integration Developer est différent de WebSphere Studio Application Developer Integration Edition car les options de déploiement de service y sont spécifiées au moment de la conception. Quand le projet est compilé, le code de déploiement est automatiquement mis à jour dans les projets EJB et Web générés et l'option **Générer le code de déploiement** n'existe donc plus.
- 7. Pour migrer tous les fichiers .bpel d'un projet de service, vous devez vérifier que tous les fichiers .wsdl et .xsd référencés par les fichiers .bpel peuvent être résolus dans un projet Business Integration dans le nouvel espace de travail :
 - Si les fichiers .wsdl et/ou .xsd sont dans le même projet de service que le fichier .bpel, aucune autre action n'est requise.
 - Si les fichiers .wsdl et/ou .xsd sont dans un autre projet de service que celui que vous migrez, les artefacts 5.1 doivent être réorganisés avec WebSphere Studio Application Developer Integration Edition avant la migration car les projets de module Business Integration ne partagent pas les artefacts. Vous pouvez réorganiser les artefacts 5.1 de deux manières :
 - Dans WebSphere Studio Application Developer Integration Edition, créez un nouveau projet Java contenant tous les artefacts communs. Placez tous les fichiers .wsdl et .xsd qui sont partagés par plusieurs projets de service dans ce nouveau projet Java. Ajoutez sur ce nouveau projet Java une dépendance avec tous les projets de service qui utilisent ces artefacts communs. Dans WebSphere Integration Developer, créez un nouveau projet de bibliothèque Business Integration portant le même nom que le projet Java 5.1 partagé avant de migrer les projets de service. Copiez manuellement les anciens fichiers .wsdl et .xsd depuis le projet Java 5.1 partagé vers le dossier de ce nouveau projet de bibliothèque BI. Ceci doit être fait avant de migrer les projets de service BPEL.
 - Vous pouvez aussi garder une copie locale des fichiers partagés .wsdl et .xsd dans chaque projet de service pour qu'il n'existe pas de dépendances entre les projets de service.
 - Si les fichiers .wsdl et/ou .xsd sont dans un autre type de projet (habituellement d'autres projets Java), vous devez créer un projet de bibliothèque d'intégration métier portant le même nom que le projet 5.1. Vous devez également définir le chemin de classe du nouveau projet de bibliothèque en y ajoutant les entrées du projet Java 5.1, le cas échéant. Ce type de projet est utile pour stocker des artefacts partagés.

Vous êtes maintenant prêt à commencer le processus de migration.

Migration des projets de service avec l'Assistant de migration de WebSphere Integration Developer

L'Assistant de migration de WebSphere Integration Developer permet de migrer des projets de service.

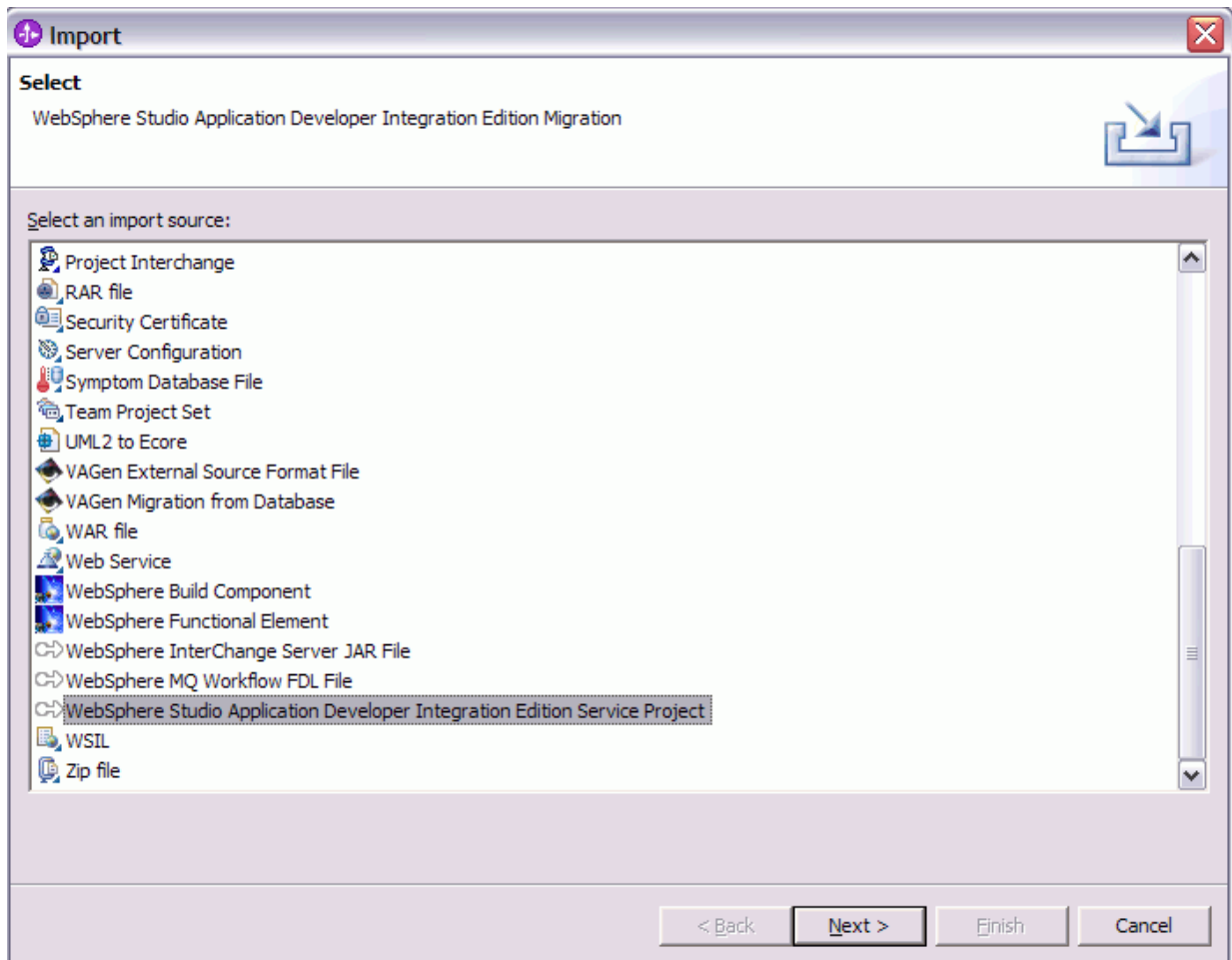
L'Assistant de migration exécute les opérations suivantes :

1. Création d'un module Business Integration (vous définissez vous-même son nom)
2. Migration des entrées de chemin de classe du projet de service vers le nouveau module
3. Copie de tous les artefacts source de WebSphere Business Integration Server Foundation depuis le projet source sélectionné vers le module
4. Migration des extensions BPEL dans des fichiers WSDL

5. Migration des processus métier (fichiers .bpel) de BPEL4WS version 1.1 vers la nouvelle version prise en charge par WebSphere Process Server, qui repose sur BPEL4WS version 1.1, avec les fonctionnalités majeures des spécifications du langage BPEL version 2.0 à venir
6. Création d'un composant SCA pour chaque .bpel
7. Création d'un fichier de contrôle .mon pour chaque processus BPEL afin de préserver le comportement de contrôle par défaut de WebSphere Studio Application Developer Integration Edition (si nécessaire)

Suivez les étapes suivantes pour migrer des projets de service avec l'Assistant de migration de WebSphere Integration Developer :

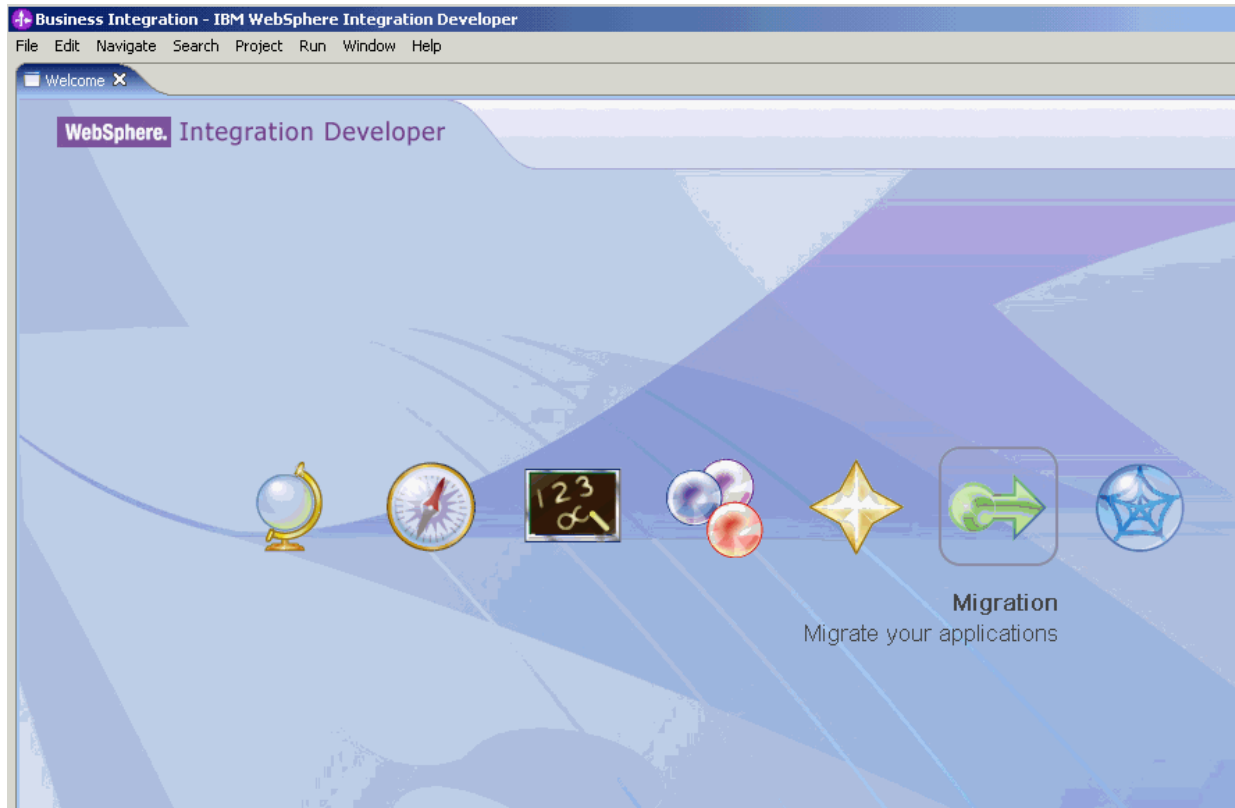
1. Appelez l'assistant en sélectionnant **Fichier** → **Importer** → **WebSphere Studio Application Developer - Projet de service d'édition d'intégration**.



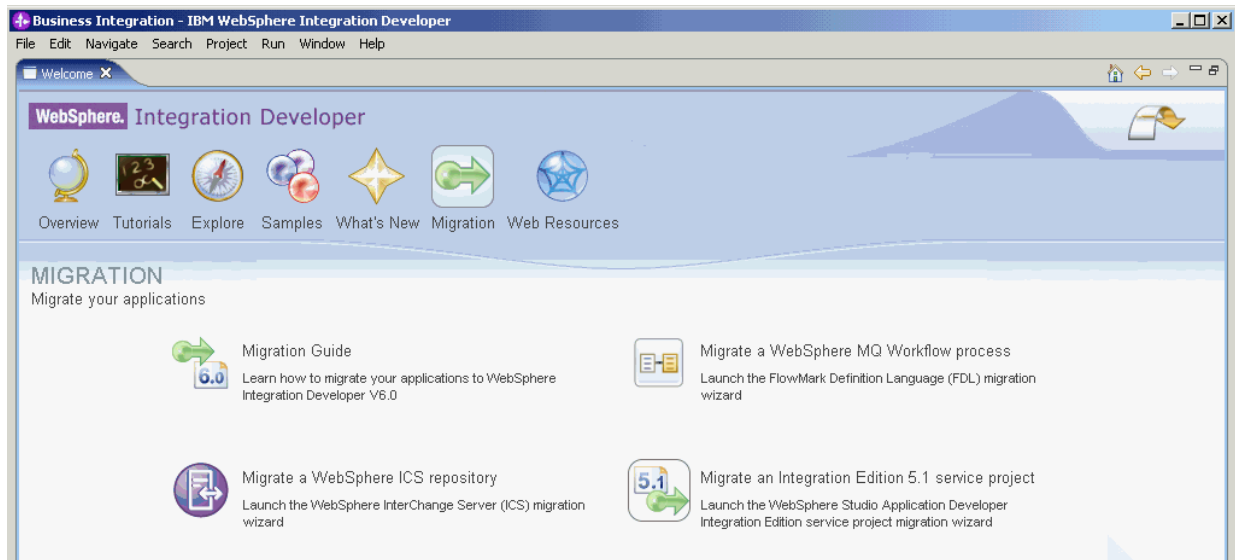
Vous pouvez également ouvrir l'Assistant de migration depuis la page d'accueil en cliquant sur



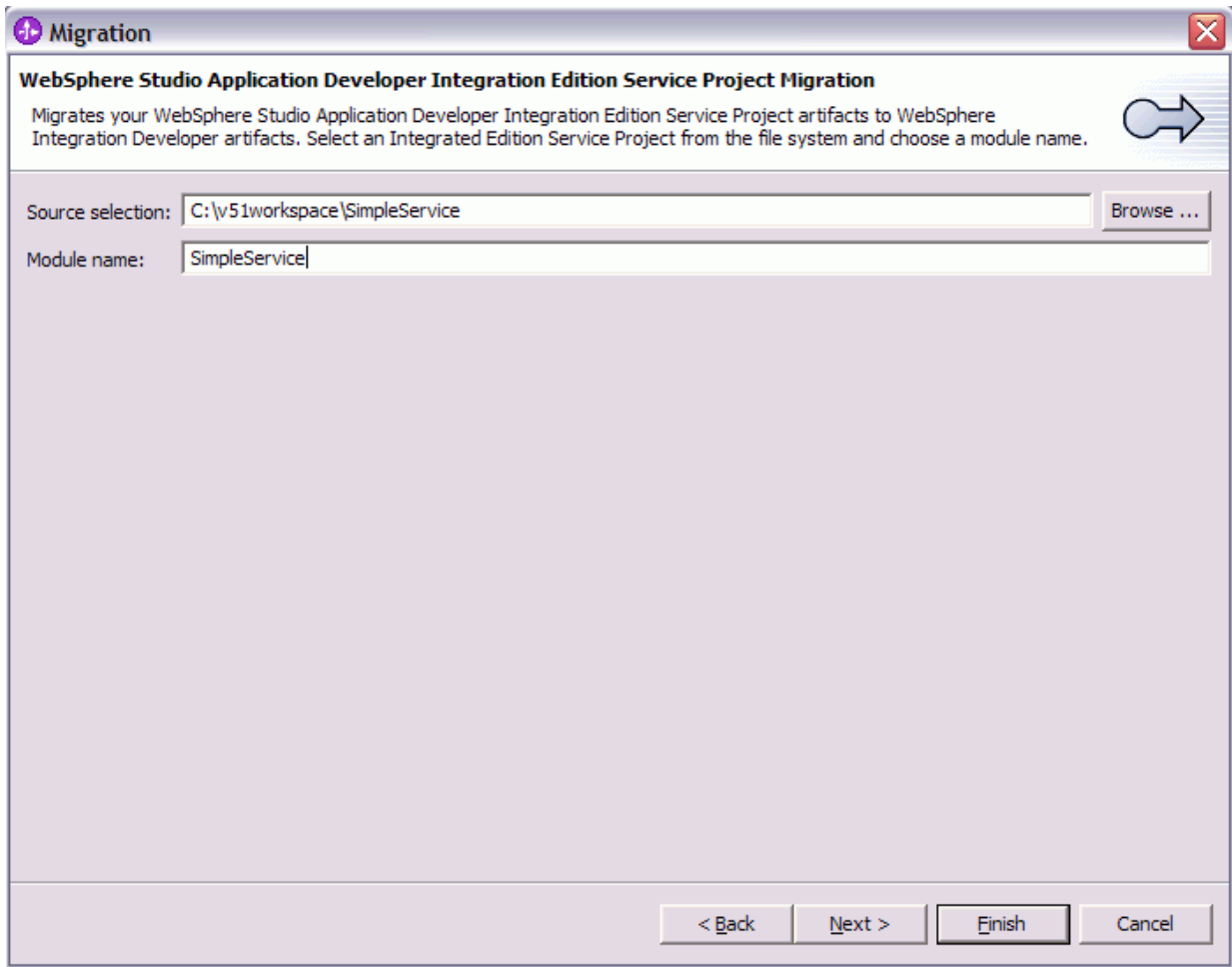
l'icône :



Dans la page Migration, sélectionnez l'option Migrer un projet de service Integration Edition 5.1 :

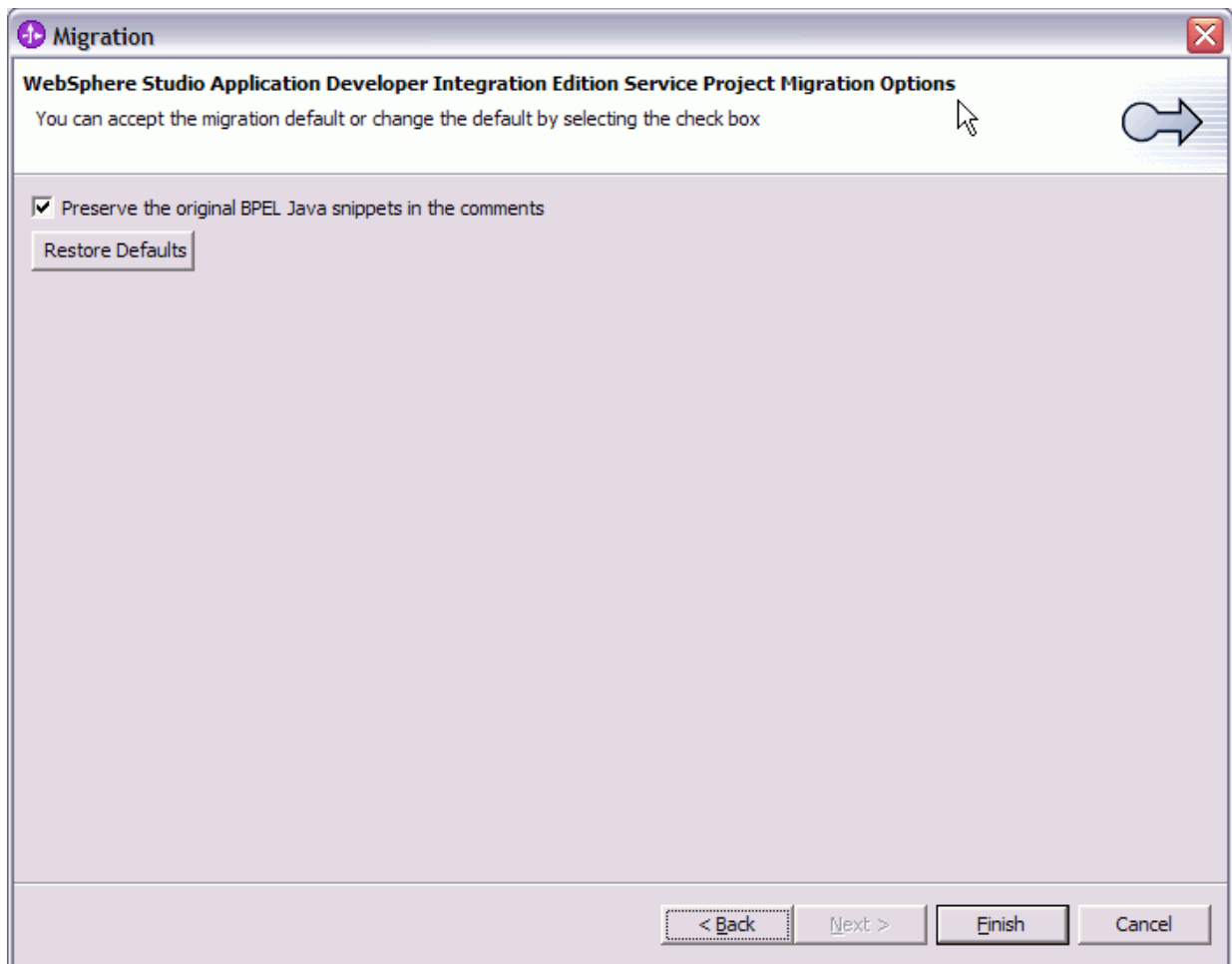


2. L'Assistant de migration apparaît. Entrez le chemin d'accès à la sélection de la source ou cliquez sur le bouton **Parcourir** pour le rechercher. Entrez également le nom de module de l'emplacement du projet de service WebSphere Studio Application Developer Integration Edition à migrer :



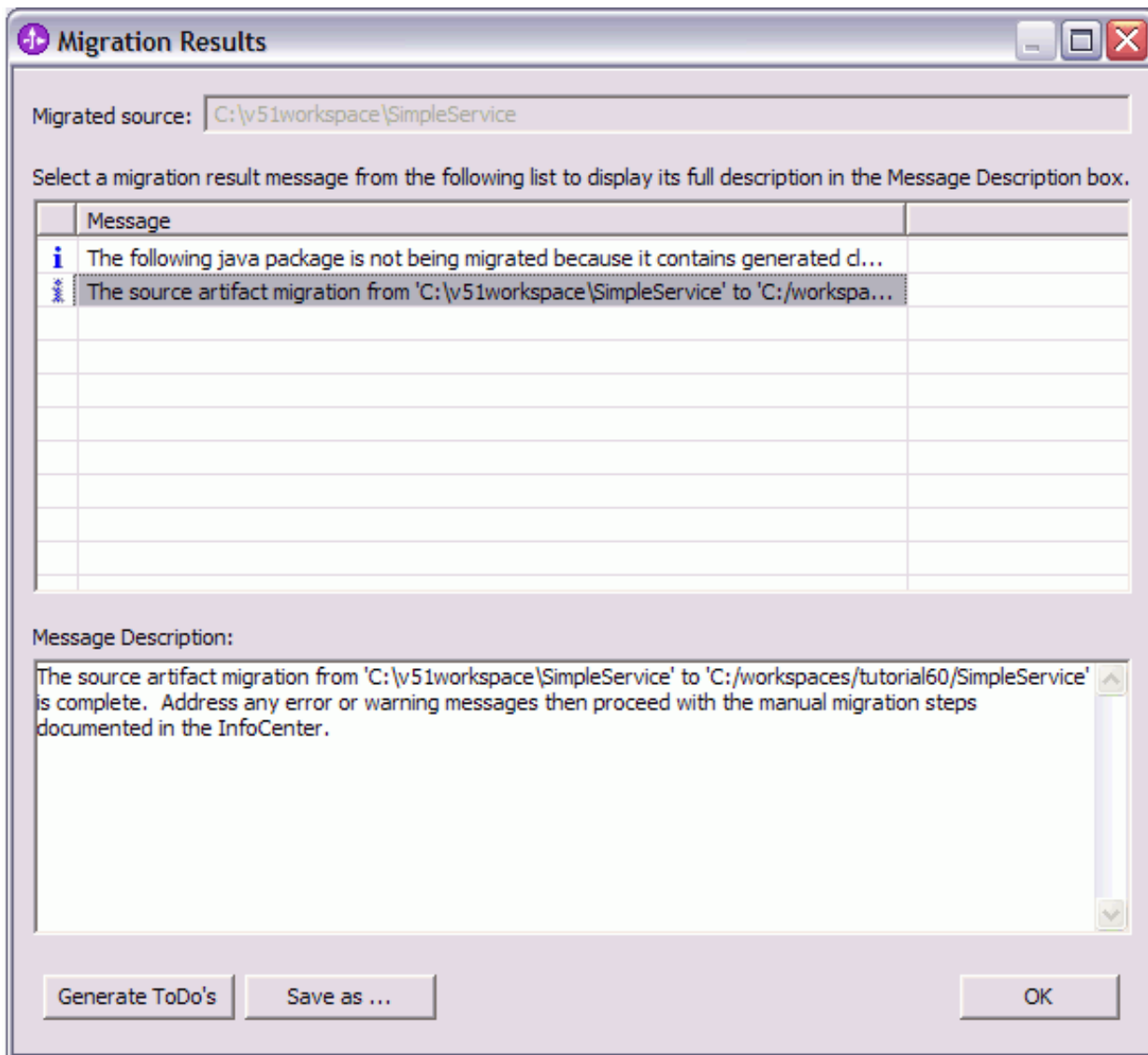
Remarque : Il est conseillé de choisir le nom du projet de service comme nom du module car, si l'espace de travail contient d'autres projets WebSphere Studio Application Developer Integration Edition qui dépendent de ce projet, vous n'aurez pas besoin de mettre à jour les chemins de classe des projets dépendants après les avoir importés dans WebSphere Integration Developer.

3. Dans les options de migration, cochez la case Conserver les fragments Java BPEL d'origine dans les commentaires :



Cliquez sur **Terminer**.

4. Une fois le processus de migration terminé, la fenêtre Résultats de la migration s'affiche :



Un fichier journal contenant ces messages de migration est automatiquement généré dans le dossier .metadata de l'espace de travail 6.0. Ce fichier journal est appelé ".log".

Quand l'Assistant de migration a terminé, compilez le module Business Integration créé et essayez de résoudre les erreurs de compilation éventuelles. Vérifiez tous les fichiers .bpel migrés. Vérifiez qu'ils sont totalement migrés et qu'ils peuvent être ouverts dans WebSphere Integration Developer BPEL Editor. Certains fragments BPEL Java ne peuvent pas être migrés automatiquement. Si vous constatez des erreurs dans les fragments Java BPEL, reportez-vous à la section "Migration vers le modèle de programmation SCA" pour connaître les étapes nécessaires à la résolution de ces erreurs. Si vous avez utilisé l'Assistant de migration pour migrer un projet de service vers un module BI, ouvrez l'éditeur de dépendances de module pour vérifier que les dépendances sont bien définies. Pour cela, passez dans Business Integration et double-cliquez sur le projet BI. Vous pouvez ensuite ajouter des dépendances sur les projets de bibliothèques d'intégration métier, les projets Java et les projets J2EE.

Migration manuelle de l'application

Lorsque l'Assistant de migration a migré les artefacts vers le nouveau module Business Integration, les artefacts doivent être liés pour créer une application conforme au modèle SCA.

1. Ouvrez WebSphere Integration Developer et passez dans Business Integration. Vous pouvez voir le ou les modules créés par l'Assistant de migration (un module par projet de service migré). Le premier artefact listé sous le projet de module est le fichier d'assemblage du module (même nom que le module).
2. Double-cliquez sur le fichier d'assemblage pour l'ouvrir dans Assembly Editor et créer des composants SCA que vous pourrez relier ensemble pour obtenir des fonctionnalités identiques à celles de l'application version 5.1. S'il existait des processus BPEL dans le projet de service WebSphere Studio Application Developer Integration Edition, l'Assistant de migration doit avoir créé des composants SCA par défaut pour chacun d'eux dans Assembly Editor.
3. Sélectionnez un composant puis ouvrez la vue Propriétés. Vous verrez les propriétés Description, Détails et Implémentation et vous pourrez les modifier.

Les informations suivantes décrivent comment relier manuellement l'application en utilisant les outils disponibles dans WebSphere Integration Developer :

Création de composants SCA et d'imports SCA pour les services dans l'application en vue d'une reconnexion :

Tous les projets demandent des reconnections après la migration afin de reconnecter les services comme ils étaient dans la version 5.1. Par exemple, tous les processus métier migrés doivent être reconnectés à leurs partenaires métier. Un composant SCA ou une importation SCA doit être créé pour tous les autres types de service. Pour les projets de service WebSphere Studio Application Developer Integration Edition qui interagissent avec des systèmes ou des entités externes au projet, une importation SCA peut être créée pour le projet migré afin d'accéder à ces entités en tant que services, en fonction du modèle de programmation SCA.

Pour les projets de service WebSphere Studio Application Developer Integration Edition qui interagissent avec des entités internes au projet (par exemple un processus métier, un service de transformateur ou une classe Java), une importation SCA peut être créée afin d'accéder à ces entités en tant que services, en fonction du modèle de programmation SCA.

Les sections suivantes fournissent d'autres détails sur l'importation SCA ou les composants SCA à créer selon le type de service à migrer.

Migration d'un service Java :

Vous pouvez migrer un service Java vers un composant Java SCA.

Si le projet de service WebSphere Studio Application Developer Integration Edition dépendait d'autres projets Java, copiez les projets existants dans le nouveau répertoire de l'espace de travail et importez-les dans WebSphere Integration Developer à l'aide des commandes **Fichier → Importer → Projet existant dans l'espace de travail**.

Dans WebSphere Studio Application Developer Integration Edition, quand vous créez un nouveau service Java depuis une classe Java existante, les options suivantes sont proposées :

- Créer des schémas XSD pour les types de données complexes :
 - Dans le fichier WSDL de l'interface
 - Comme nouveau fichier pour chaque type de données
- Prendre en charge la gestion d'erreurs :
 - Générer une erreur
 - Ne pas générer d'erreur
- Autres détails sur le service à générer comme les noms des connexions et des services.

La version 6.0 propose de nouvelles fonctions telles que le mappage de données, la médiation par interface, les machines métier, les sélecteurs, les règles métier, etc. Vous devez d'abord déterminer si certains de ces nouveaux composants peuvent remplacer un composant personnalisé Java. Si cela n'est pas possible, suivez la procédure de migration décrite ci-après.

Importez le projet de service à l'aide de l'Assistant de migration. Cela va créer un module Business Integration contenant les messages WSDL, les types de port, les connexions et les services créés dans WebSphere Studio Application Developer Integration Edition.

Dans Business Integration, développez le module pour voir son contenu. Pour ouvrir Assembly Editor, double-cliquez sur le premier élément sous le projet de module (même nom que le projet).

Vous disposez des options suivantes :

Avantages et inconvénients de chacune des options de nouvelle connexion d'un service Java :

Il existe des avantages et des inconvénients pour chacune des options de nouvelle connexion d'un service Java.

La liste suivante décrit les options ainsi que les avantages et les inconvénients de chaque option :

- La première option est censée donner de meilleures performances pendant l'exécution car l'appel d'un service Web est plus lent que l'appel d'un composant Java.
- La première option peut propager le contexte alors que l'appel d'un service Web ne propage pas le contexte de la même manière.
- La deuxième option n'implique pas la création d'un code personnalisé.
- Il est possible que la deuxième option ne soit pas disponible pour certaines définitions d'interface Java dans la mesure où la génération d'un service Java est soumise à des restrictions. Voir la documentation de Rational Application Developer à l'adresse suivante : <http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.doc/ref/rlimit.html>
- La deuxième option peut entraîner la modification de l'interface, et par conséquent, la modification du client SCA.
- La deuxième option requiert qu'un serveur WebSphere Process Server 6.0 soit installé et configuré pour fonctionner avec WebSphere Integration Developer. Pour afficher les environnements d'exécution configurés pour fonctionner avec WebSphere Integration Developer, cliquez sur **Fenêtre → Préférences → Serveur → Environnements d'exécution installés**. Sélectionnez l'entrée **WebSphere Process Server V6.0**, le cas échéant, et assurez-vous qu'elle pointe vers l'emplacement d'installation du produit. Vérifiez que cette entrée est activée si le serveur existe et désactivée si le serveur n'est pas installé. Vous pouvez également cliquer sur le bouton **Ajouter...** pour ajouter un autre serveur.
- Si le composant Java a été généré dans WebSphere Studio Application Developer Integration Edition à l'aide de l'approche descendante dans laquelle le squelette Java a été généré à partir d'un WSDL, les paramètres à l'intérieur et à l'extérieur de cette classe Java mettront probablement **WSIFFormatPartImpl** sous forme de sous-classe. Dans ce cas, choisissez la première option pour générer un nouveau squelette Java de style SCA à partir des WSDL/XSD d'origine ou la deuxième option pour générer un nouveau squelette Java générique (indépendant des API DataObject ou WSIF) à partir de l'interface WSDL d'origine.

Créer un composant personnalisé Java : option 1 :

La technique de migration recommandée est d'utiliser le type de composant WebSphere Integration Developer Java qui permet de représenter le service Java en tant que composant SCA. Au cours de la migration, vous devez écrire un code Java pour faire la conversion entre le style d'interface Java SCA et le style d'interface Java existant du composant.

Pour créer le composant Java personnalisé :

1. Sous le projet de module, développez **Interfaces** et sélectionnez l'interface WSDL créée pour cette classe Java dans WebSphere Studio Application Developer Integration.
2. Faites glisser cette interface dans l'éditeur d'assemblage. Une boîte de dialogue vous demande de sélectionner le type de composant à créer. Sélectionnez **Composant sans type d'implémentation** et cliquez sur **OK**.
3. Un composant générique apparaît dans le diagramme de l'assemblage. Sélectionnez-le et passez dans la vue **Propriétés**.
4. Dans l'onglet **Description**, vous pouvez remplacer le nom et le nom d'affichage du composant par un nom plus descriptif.
5. Dans l'onglet **Détails**, vous verrez que ce composant a une interface, celle que vous avez fait glisser dans l'éditeur d'assemblage.
6. Vérifiez que la classe Java à laquelle vous tentez d'accéder est dans le chemin de classe du projet de service, si elle n'est pas dans le projet de service lui-même.
7. Avec le bouton droit, cliquez sur le projet de module et sélectionnez **Ouvrir l'éditeur de dépendances**. Vérifiez que le projet contenant l'ancienne classe Java apparaît dans la section **Java**. Dans la négative, ajoutez-le en cliquant sur le bouton **Ajouter**.
8. De retour dans l'éditeur d'assemblage, avec le bouton droit, cliquez sur le composant que vous venez de créer et sélectionnez **Générer l'implémentation** → **Java**. Sélectionnez ensuite le module dans lequel l'implémentation Java sera générée. Ceci crée une structure de service Java compatible avec l'interface WSDL et le modèle de programmation SCA dans laquelle les types complexes sont représentés par un objet `commonj.sdo.DataObject` et les types simples sont représentés par des objets Java équivalents.

Les exemples de code suivants montrent :

1. les définitions correspondantes issues de l'interface WSDL 5.1 ;
2. les méthodes Java de WebSphere Studio Application Developer Integration Edition 5.1 qui correspondent à WSDL ;
3. les méthodes Java de WebSphere Integration Developer 6.0 pour WSDL.

Le code suivant montre les définitions correspondantes issues de l'interface WSDL 5.1 :

```
<types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
    attributeFormDefault="qualified"
    elementFormDefault="unqualified"
    targetNamespace="http://migr.practice.ibm.com/"
    xmlns:xsd1="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>

  <part name="symbol" type="xsd:string"/>
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
</message>
<message name="getStockInfoResponse">
  <part name="result" type="xsd1:StockInfo"/>
</message>

  <operation name="getStockInfo" parameterOrder="symbol">
    <input message="tns:getStockInfoRequest">
```

```

        name="getStockInfoRequest"/>
<output message="tns:getStockInfoResponse"
        name="getStockInfoResponse"/>
</operation>

```

Le code suivant montre les méthodes Java de WebSphere Studio Application Developer Integration Edition 5.1 qui correspondent à WSDL :

```

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

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

```

Le code suivant montre les méthodes Java de WebSphere Integration Developer 6.0 pour 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.
}

```

Vous devez ensuite remplir les lignes de code contenant "//TODO" dans la classe d'implémentation Java générée. Vous avez deux options :

1. Transférer la logique de la classe Java d'origine vers cette nouvelle classe en l'adaptant aux objets DataObjects.
 - C'est l'option recommandée si vous avez choisi l'approche partant du haut dans WebSphere Studio Application Developer Integration Edition et que vous voulez que votre composant Java puisse gérer les paramètres DataObject. Cette modification est nécessaire car les classes Java générées à partir des définitions WSDL dans WebSphere Studio Application Developer Integration Edition ont des dépendances WSIF qui doivent être éliminées.
2. Créez une instance privée de l'ancienne classe Java à l'intérieur de la nouvelle classe Java et écrivez un code pour :
 - a. convertir tous les paramètres de la classe d'implémentation Java générée en paramètres compatibles avec l'ancienne classe Java ;
 - b. appeler l'instance privée de l'ancienne classe Java avec les paramètres convertis ;
 - c. convertir la valeur de retour de l'ancienne classe Java dans le type de valeur de retour déclarée par la méthode d'implémentation Java générée.
 - d. Cette option est recommandée pour les scénarios de consommation dans lesquels les proxy de service WSIF doivent être utilisés par des nouveaux composants Java de type 6.0.

Quand vous avez terminé l'une des options ci-dessus, vous devez reconnecter le service Java. En l'absence de référence, il vous suffit de reconnecter l'interface du composant Java :

- Si ce service est appelé par un processus métier situé dans le même module, vous devez créer une connexion entre la référence de processus métier appropriée et l'interface de ce composant Java.
- Si ce service est appelé par un processus métier situé dans un autre module, créez une **exportation avec liaison SCA** et à partir de l'autre module, effectuez un glisser-déposer de cette exportation vers l'éditeur d'assemblage de ce module pour créer l'**importation avec liaison SCA** correspondante. Connectez la référence de processus métier appropriée à cette importation.

- Si ce service a été publié dans WebSphere Studio Application Developer Integration Edition pour faire l'objet d'une exposition externe, voir la section "Création d'exports SCA afin d'accéder au service migré" pour savoir comment le publier de nouveau.

Création d'un service Web Java : Option 2 :

Les services Web Application Developer Rational vous permettent de créer un service Web basé sur une classe Java.

Remarque : Avant de tenter une migration avec cette méthode, consultez les informations disponibles sur le site suivant : <http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.was.creation.ui.doc/tasks/twsbeanw.html>

Remarque : Cette option demande de configurer un environnement d'exécution de service Web dans WebSphere Integration Developer avant d'appeler l'assistant de service Web.

Si vous avez choisi l'approche partant du bas dans WebSphere Studio Application Developer Integration Edition pour générer WSDL sur la base d'une classe Java, procédez comme suit :

1. Créez un nouveau projet Web et copiez la classe Java sur laquelle vous voulez baser un service dans le dossier source Java de ce projet Web.
2. Avec le bouton droit, cliquez sur le projet d'application métier qui contient la classe Java sur laquelle le service sera basé.
3. Sélectionnez **Propriétés**, accédez aux propriétés **Serveur** et vérifiez que la zone **Environnement d'exécution cible** a la valeur **WebSphere Process Server v6.0** et que **Serveur par défaut** correspond au serveur **WebSphere Process Server v6.0** installé.
4. Démarrez le serveur de test et déployez l'application sur ce serveur puis vérifiez que le lancement a réussi.
5. Ensuite, avec le bouton droit, cliquez sur la classe Java sur laquelle vous voulez baser votre nouveau service puis sélectionnez **Services Web** → **Créer un service Web**.
6. Pour **Type de service Web**, sélectionnez **Service Web de bean Java** et décochez l'option **Démarrer le service Web dans le projet Web**, sauf si vous voulez déployer le service Web immédiatement. Vous pouvez aussi choisir de générer un proxy client. Cliquez sur **Suivant**.
7. La classe Java sur laquelle vous avez cliqué apparaît. Cliquez sur **Suivant**.
8. Vous devez maintenant configurer les options de déploiement du service. Cliquez sur le bouton **Editer**. Pour le type de serveur, choisissez **WPS Server v6.0** et, pour l'environnement d'exécution du service Web, choisissez **IBM WebSphere** et J2EE version **1.4**. Si vous n'arrivez pas à sélectionner une combinaison valide, reportez-vous à la section "Préparation de la migration" pour plus d'informations sur la migration des projets J2EE vers le niveau v1.4. Cliquez sur **OK**.
9. Pour le projet de service, entrez le nom du projet Web. Sélectionnez aussi le projet EAR approprié. Cliquez sur **Suivant**. Notez que ceci peut demander quelques minutes d'attente.
10. Dans le volet d'identification du bean Java du service Web, sélectionnez le fichier WSDL qui contiendra les définitions WSDL. Choisissez les méthodes qui seront disponibles dans le service Web et le style/codage approprié (Document/Literal, RPC/Literal, ou RPC/Encoded). Sélectionnez l'option **Define custom mapping for package to namespace**, puis choisissez un espace de nom uniquement associé à la classe Java en cours de migration pour tous les packages Java utilisés par l'interface de cette classe Java. L'espace de nom par défaut étant associé de façon unique au nom du package, des conflits peuvent se produire si vous créez un autre service Web utilisant les mêmes classes Java. Le cas échéant, renseignez les autres paramètres.
11. Cliquez sur **Suivant**. Dans le panneau **Web Service package to namespace mapping**, cliquez sur le bouton **Ajouter**. Dans la ligne créée, entrez le nom de package du bean Java, puis ajoutez l'espace de nom personnalisé qui identifie cette classe Java de façon unique. Continuez à ajouter des mappages pour tous les packages Java utilisés par l'interface du bean Java.
12. Cliquez sur **Suivant**. Notez que ceci peut demander quelques minutes d'attente.

13. Cliquez sur **Terminer**. Après avoir terminé l'assistant, copiez le fichier WSDL généré qui définit le service Java pour le projet de module Business Integration si le projet de service est un utilisateur du service Java. Vous le trouverez dans le projet Web de routeur généré, dans le dossier WebContent/WEB-INF/wsdl. Réactualisez/recompilez le projet de module Business Integration.
14. Passez dans Business Integration et développez le module puis la catégorie logique **Ports de service Web**.
15. Sélectionnez le port créé dans les étapes précédentes et faites-le glisser dans l'éditeur d'assemblage puis sélectionnez **Importation avec liaison de services Web**. Sélectionnez l'interface WSDL de la classe Java si cela vous est demandé. Le composant SCA qui consommait le composant Java dans la version 5.1 peut maintenant être connecté à cette importation pour terminer les étapes de migration de la reconnexion manuelle.

Notez que l'interface peut être légèrement différente de celle de la version 5.1 et qu'il vous faudra peut-être insérer un composant de médiation d'interface entre le consommateur 5.1 et la nouvelle importation. Pour ce faire, cliquez sur l'outil **Connexion** dans l'éditeur d'assemblage et connectez le composant source SCA à cette nouvelle **importation avec liaison de services Web**. Comme les interfaces sont différentes, un message vous indiquera que les **noeuds source et cible ont des interfaces différentes**. Choisissez de **créer un mappage d'interfaces entre le noeud source et le noeud cible**. Double-cliquez sur le composant de mappage créé dans Assembly Editor. L'éditeur d'assemblage s'ouvre. Voir le Centre de documentation pour savoir comment créer un mappage d'interfaces.

Si vous avez choisi l'approche partant du bas dans WebSphere Studio Application Developer Integration Edition pour créer des classes Java depuis une définition WSDL, procédez comme suit :

1. Créez un nouveau projet Web et copiez le fichier WSDL que vous voulez associer au squelette Java du dossier source de ce projet Web.
2. Avec le bouton droit, cliquez sur le fichier WSDL contenant le type de port à partir duquel vous voulez générer le squelette Java puis sélectionnez **Services Web → Générer le squelette de bean Java**.
3. Choisissez le type de service Web **Service squelette Web de bean Java** et complétez l'assistant.

Une fois l'assistant terminé, vous devez disposer de classes Java qui implémentent l'interface du service et ne dépendent pas des API WSIF.

Migration d'un service EJB :

Vous pouvez migrer un service EJB vers une importation SCA avec liaison de bean session sans état.

Si le projet de service WebSphere Studio Application Developer Integration Edition est dépendant d'un autre projet EJB, client EJB, ou Java, importez ces projets à l'aide de l'option **Fichier → Importer → Projet existant dans l'espace de travail** de l'assistant. C'est généralement le cas quand un EJB a été référencé depuis un projet de service. Si un des fichiers WSDL ou XSD référencés depuis le projet de service existe dans un autre type de projet, créez une nouvelle bibliothèque Business Integration sous le même nom que l'ancien projet (non service) et copiez tous les artefacts dans la bibliothèque.

Importez le projet de service à l'aide de l'Assistant de migration. Cette opération crée un module Business Integration avec les messages WSDL, les types de port, les connexions et les services générés dans WebSphere Studio Application Developer Integration Edition.

Dans Business Integration, développez le module pour voir son contenu. Pour ouvrir Assembly Editor, double-cliquez sur le premier élément sous le projet de module (même nom que le projet).

Vous avez plusieurs possibilités :

Avantages et inconvénients de chacune des options de nouvelle connexion d'un service EJB :

Il existe des avantages et des inconvénients pour chacune des options de nouvelle connexion d'un service EJB.

La liste suivante décrit les options ainsi que les avantages et les inconvénients de chaque option :

- La première option est censée donner de meilleures performances pendant l'exécution car l'appel d'un service Web est plus lent que l'appel d'un EJB.
- La première option peut propager le contexte alors que l'appel d'un service Web ne propage pas le contexte de la même manière.
- La deuxième option n'implique pas la création d'un code personnalisé.
- Il est possible que la deuxième option ne soit pas disponible pour certaines définitions d'interface EJB dans la mesure où la génération d'un service EJB est soumise à des restrictions. Voir la documentation de Rational Application Developer à l'adresse suivante :
<http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.doc/ref/rlimit.html>
- La deuxième option peut entraîner la modification de l'interface, et par conséquent, la modification du client SCA.
- La deuxième option requiert qu'un serveur WebSphere Process Server 6.0 soit installé et configuré pour fonctionner avec WebSphere Integration Developer. Pour afficher les environnements d'exécution configurés pour fonctionner avec WebSphere Integration Developer, cliquez sur **Fenêtre → Préférences → Serveur → Environnements d'exécution installés**. Sélectionnez l'entrée **WebSphere Process Server V6.0**, le cas échéant, et assurez-vous qu'elle pointe vers l'emplacement d'installation du produit. Vérifiez que cette entrée est activée si le serveur est présent et désactivée si le serveur n'est pas installé. Vous pouvez également cliquer sur le bouton **Ajouter...** pour ajouter un autre serveur.
- Si le composant Java a été généré dans WebSphere Studio Application Developer Integration Edition à l'aide de l'approche descendante dans laquelle le squelette EJB a été généré à partir d'un WSDL, les paramètres à l'intérieur et à l'extérieur de cette classe Java mettront probablement **WSIFFormatPartImpl** sous forme de sous-classe. Dans ce cas, choisissez la deuxième option pour générer un nouveau squelette EJB générique (indépendant des API DataObject ou WSIF) à partir de l'interface WSDL d'origine.

Création du composant EJB personnalisé : première option :

La technique de migration recommandée consiste à utiliser le type d'importation avec liaison de session sans état de WebSphere Integration Developer qui vous permet d'appeler un EJB de session sans état en tant que composant SCA. Pendant la migration, vous devez écrire du code Java personnalisé pour la conversion entre le style d'interface Java SCA et le style d'interface EJB existant.

Pour créer le composant EJB personnalisé :

1. Sous le projet de module, développez **Interfaces** et sélectionnez l'interface WSDL générée pour cet EJB dans WebSphere Studio Application Developer Integration.
2. Effectuez un glisser-déposer de cette interface vers l'éditeur d'assemblage. Une boîte de dialogue vous demande de sélectionner le type de composant à créer. Sélectionnez **Composant (sans type d'implémentation)** et cliquez sur **OK**.
3. Un composant générique apparaît dans le diagramme de l'assemblage. Sélectionnez-le et accédez à la vue **Propriétés**.
4. Dans l'onglet **Description**, vous pouvez remplacer le nom et le nom d'affichage du composant par un nom plus descriptif. Attribuez-lui le nom de votre EJB et ajoutez un suffixe tel que "JavaMed", car ce composant Java servira d'intermédiaire entre l'interface WSDL générée pour l'EJB dans WebSphere Studio Application Developer Integration et l'interface Java de l'EJB.
5. Dans l'onglet **Détails**, vous voyez que ce composant a une interface (celle pour laquelle vous avez effectué un glisser-déposer vers l'éditeur d'assemblage).
6. Dans l'éditeur d'assemblage, cliquez avec le bouton droit de la souris sur le composant que vous venez de créer et sélectionnez **Générer l'implémentation...** → **Java** Sélectionnez ensuite le package

dans lequel l'implémentation Java sera générée. Cette opération crée un service Java squelette conforme à l'interface WSDL selon le modèle de programmation SCA, où les types complexes sont représentés par un objet de type `commonj.sdo.DataObject` et où les types simples sont représentés par leurs équivalents objet Java.

Les exemples de code suivants affichent :

1. des définitions appropriées provenant de l'interface WSDL 5.1 ;
2. les méthodes Java de WebSphere Studio Application Developer Integration Edition 5.1 qui correspondent à l'interface WSDL ;
3. les méthodes Java de WebSphere Integration Developer 6.0 pour la même interface WSDL.

Le code suivant affiche les définitions appropriées provenant de l'interface WSDL 5.1 :

```
<types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
    attributeFormDefault="qualified"
    elementFormDefault="unqualified"
    targetNamespace="http://migr.practice.ibm.com/"
    xmlns:xsd1="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="xsd1:StockInfo"/>
</message>

<operation name="getStockInfo" parameterOrder="symbol">
  <input message="tns:getStockInfoRequest"
    name="getStockInfoRequest"/>
  <output message="tns:getStockInfoResponse"
    name="getStockInfoResponse"/>
</operation>
```

Le code suivant affiche les méthodes Java de WebSphere Studio Application Developer Integration Edition 5.1 qui correspondent à l'interface WSDL :

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

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

Le code suivant affiche les méthodes Java de WebSphere Integration Developer 6.0 pour la même interface 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.
}

```

Pour finir, vous devez entrer du code réel aux endroits où vous voyez les balises "//TODO" dans la classe d'implémentation Java générée. Commencez par créer une référence de ce composant Java vers l'EJB réel de sorte qu'il puisse accéder à l'EJB selon le modèle de programmation SCA :

1. Laissez l'éditeur d'assemblage ouvert et accédez à J2EE. Localisez le projet EJB contenant l'EJB pour lequel vous créez un service.
2. Développez **Descripteur de déploiement : <nom-projet>** et localisez l'EJB. Effectuez un glisser-déposer vers l'éditeur d'assemblage. Si un message vous informe que les dépendances du projet doivent être mises à jour, cochez la case **Ouvrir l'éditeur de dépendances du module...** et cliquez sur **OK**.
3. Sous la section J2EE, assurez-vous que le projet EJB est répertorié et, dans le cas contraire, ajoutez-le en cliquant sur le bouton **Ajouter....**
4. Enregistrez les dépendances du module et fermez cet éditeur. Vous voyez qu'une nouvelle importation a été créée dans l'éditeur d'assemblage. Vous pouvez le sélectionner et accéder à la vue Propriétés de l'onglet Description afin de remplacer le nom et le nom d'affichage de cette importation par un nom plus explicite. Dans l'onglet Connexion, vous remarquez que le type d'importation a automatiquement la valeur **Connexion de bean session sans état** et que le nom JNDI de l'EJB est déjà défini de manière appropriée.
5. Sélectionnez l'outil Connexion dans la palette de l'éditeur d'assemblage.
6. Cliquez sur le composant Java et relâchez le bouton de la souris.
7. Ensuite, cliquez sur l'importation EJB et relâchez le bouton de la souris.
8. Vous allez voir apparaître un message **Une référence correspondante va être créée sur le noeud source. Voulez-vous continuer ?** Cliquez sur **OK**. Une connexion est créée entre les deux composants.
9. Sélectionnez le composant Java dans l'éditeur d'assemblage et dans la vue Propriétés de l'onglet Détails, développez Références, puis sélectionnez la référence à l'EJB que vous venez de créer. Vous pouvez mettre à jour le nom de la référence si le nom généré n'est pas assez descriptif ou pas approprié. N'oubliez pas le nom de cette référence en vue d'une utilisation future.
10. Enregistrez le diagramme de l'assemblage.

Vous devez utiliser le modèle de programmation SCA pour appeler l'EJB à partir de la classe Java générée. Ouvrez la classe Java générée et suivez ces étapes pour écrire le code qui appellera le service EJB. Pour la classe d'implémentation Java générée :

1. Créez une variable privée (dont le type correspond à celui de votre interface EJB distante) :

```
private YourEJBInterface ejbService = null;
```
2. Si votre interface EJB contient des types complexes, créez également une variable privée pour BOFactory :

```
private BOFactory boFactory = (BOFactory)
    ServiceManager.INSTANCE.locateService("com/ibm/websphere/bo
    /BOFactory");
```
3. Dans le constructeur de la classe d'implémentation Java, utilisez les API SCA pour résoudre la référence EJB (pensez à indiquer le nom de la référence EJB que vous avez écrit dans une étape précédente) et affectez à la variable privée la même valeur que cette référence :

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

Pour chaque balise "//TODO" dans la classe d'implémentation Java générée :

1. Convertissez tous les paramètres en types de paramètre attendus par l'EJB.
2. Appelez la méthode appropriée sur la référence EJB à l'aide du modèle de programmation SCA, en envoyant les paramètres convertis.
3. Convertissez la valeur de retour de l'EJB en type de valeur de retour déclaré par la méthode d'implémentation Java générée.

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

Création d'un service Web EJB : Option 2 :

L'outil des services Web Rational Application Developer est une autre option qui vous permet de créer un service Web autour d'un EJB.

Remarque : Reportez-vous aux informations disponibles sur le site Web suivant avant de tenter une migration à l'aide de cette méthode :

<http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.was.creation.ejb.ui.doc/tasks/twsejwb.html>

Remarque : Cette option requiert qu'un environnement d'exécution de service Web soit configuré via WebSphere Integration Developer avant l'appel de l'Assistant de service Web.

Pour créer un service Web autour d'un EJB, procédez comme suit :

1. Cliquez avec le bouton droit de la souris sur le projet d'application d'entreprise situé dans le conteneur pour l'EJB autour duquel vous créez un service.

2. Sélectionnez **Propriétés**, accédez aux propriétés **Serveur** et vérifiez que la zone **Environnement d'exécution cible** a la valeur **WebSphere Process Server v6.0** et que **Serveur par défaut** correspond au serveur **WebSphere Process Server v6.0** installé.
3. Démarrez le serveur de test, déployez cette application vers le serveur et contrôlez son démarrage.
4. Dans J2EE, développez le **projet EJB** dans la vue de l'explorateur de projet. Développez le **descripteur de déploiement**, puis la catégorie **Beans de session**. Sélectionnez le bean autour duquel générer le service Web.
5. Cliquez-dessus avec le bouton droit de la souris et sélectionnez **Services Web → Créer un service Web**.
6. Sélectionnez **Service Web EJB** pour **Type de service Web** et désactivez l'option **Démarrer le service Web dans le projet Web**, à moins que vous ne souhaitiez déployer immédiatement le service Web. Cliquez sur **Suivant**.
7. Vérifiez que l'EJB sur lequel vous avez cliqué avec le bouton droit de la souris est sélectionné, puis cliquez sur **Suivant**.
8. Vous devez maintenant configurer les options de déploiement du service. Cliquez sur le bouton **Editer....** Sélectionnez **WPS Server v6.0** comme type de serveur ; pour l'exécution du service Web, sélectionnez **IBM WebSphere** et J2EE version **1.4**. Si vous ne pouvez pas sélectionner une combinaison valide à cette étape, reportez-vous à la section "Préparation de la migration" pour plus d'informations sur la migration des projets J2EE vers la version 1.4. Cliquez sur **OK**.
9. Pour le projet Service, entrez le nom du projet EJB contenant l'EJB. Sélectionnez également le projet EAR approprié. Cliquez sur **Suivant**. Notez que cette opération peut prendre quelques minutes.
10. Dans le panneau relatif à la configuration EJB du service Web, sélectionnez le projet routeur approprié à utiliser (choisissez le nom du projet Web routeur à créer pour que ce projet soit ajouté à la même application d'entreprise que l'EJB d'origine. Sélectionnez le transfert souhaité (**SOAP sur HTTP** ou **SOAP sur JMS**). Cliquez sur **Suivant**.
11. Sélectionnez le fichier WSDL qui contiendra les définitions WSDL. Sélectionnez les méthodes que vous souhaitez exposer sur le service Web et choisissez le style/codage approprié (Document/Literal, RPC/Literal ou RPC/Encoded). Sélectionnez l'option **Define custom mapping for package to namespace**, puis choisissez un espace de nom uniquement associé à l'EJB en cours de migration pour tous les packages Java utilisés par cet EJB. L'espace de nom par défaut étant associé de façon unique au nom du package, des conflits peuvent se produire si vous créez un autre service Web utilisant les mêmes classes Java. Le cas échéant, renseignez les autres paramètres. Il existe des restrictions pour chaque combinaison style/codage. Pour plus d'informations à ce sujet, voir <http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.doc/ref/rlimit.html>.
12. Cliquez sur **Suivant**. Dans le panneau **Web Service package to namespace mapping**, cliquez sur le bouton **Ajouter**. Dans la ligne créée, entrez le nom de package de l'EJB, puis ajoutez l'espace de nom personnalisé qui identifie cet EJB de façon unique. Continuez à ajouter des mappages pour tous les packages Java utilisés par l'interface EJB.
13. Cliquez sur **Suivant**. Notez que cette opération peut prendre quelques minutes.
14. Cliquez sur **Terminer**. Une fois l'Assistant terminé, vous devez copier le fichier WSDL généré qui décrit le service EJB vers le projet du module Business Integration si le projet de service était un consommateur du service EJB. Il est situé dans le projet Web routeur généré sous le dossier WebContent/WEB-INF/wsdl. Actualisez/Concevez de nouveau le projet du module Business Integration.
15. Accédez à Business Integration et développez le module migré, puis la catégorie logique **Ports de service Web**.
16. Sélectionnez le port généré dans les étapes précédentes et faites-le glisser vers l'éditeur d'assemblage, puis créez une **importation avec liaison de services Web**. Sélectionnez l'interface WSDL de l'EJB si vous y êtes invité. A présent, le composant SCA qui utilisait l'EJB dans la version 5.1 peut être connecté à cette importation pour exécuter les étapes de migration de nouvelle connexion manuelle.

Si vous avez utilisé une approche descendante dans WebSphere Studio Application Developer Integration Edition, en générant un squelette EJB à partir d'une définition WSDL, procédez comme suit :

1. Créez un projet Web et copiez le fichier WSDL à partir duquel vous souhaitez générer le squelette EJB dans le dossier source de ce projet Web.
2. Cliquez avec le bouton droit de la souris sur le fichier WSDL contenant le type de port à partir duquel vous souhaitez générer le squelette EJB et sélectionnez **Services Web → Générer un squelette de bean Java**.
3. Sélectionnez **Service squelette Web EJB** comme type de service Web et terminez l'Assistant.

Une fois l'assistant terminé, vous devez disposer d'un EJB qui implémente l'interface de service et qui ne dépend pas des API WSIF.

Notez que l'interface peut légèrement différer de l'interface 5.1, et vous devrez peut-être insérer un composant de médiation d'interface entre le consommateur 5.1 et la nouvelle importation. Pour ce faire, cliquez sur l'outil **Connexion** dans l'éditeur d'assemblage et connectez le composant source SCA à cette nouvelle **importation avec liaison de services Web**. Etant donné que les interfaces diffèrent, vous recevez le message suivant : **Les noeuds source et cible n'ont pas d'interfaces correspondantes**. Choisissez de **créer un mappage d'interface entre les noeuds source et cible**. Double-cliquez sur le composant de mappage créé dans l'éditeur d'assemblage. L'éditeur de mappage s'affiche. Consultez le centre d'informations pour obtenir des instructions sur la création d'un mappage d'interface.

Ensuite, vous devez reconnecter le service EJB. Comme il ne doit exister aucune référence, il vous suffit de reconnecter l'interface du composant Java :

- Si ce service est appelé par un processus métier situé dans le même module, vous devez créer une connexion entre la référence de processus métier appropriée et l'interface de ce composant.
- Si ce service est appelé par un processus métier situé dans un autre module, créez une **exportation avec liaison SCA** et à partir de l'autre module, effectuez un glisser-déposer de cette exportation vers l'éditeur d'assemblage de ce module pour créer l'**importation avec liaison SCA** correspondante. Connectez la référence de processus métier appropriée à cette importation.
- Si ce service a été publié dans WebSphere Studio Application Developer Integration Edition pour un usage externe, reportez-vous à la section "Migration d'un service non BPEL entrant" pour savoir comment le republier.

Migration d'un processus métier vers un appel de service de processus métier :

Ce scénario s'applique à un processus métier qui appelle un autre processus métier à l'aide d'une connexion de processus WSIF. Cette section décrit comment migrer un processus BPEL vers un appel de service BPEL à l'aide d'une connexion ou d'une importation/exportation avec liaison SCA.

Pour migrer un projet de service de connexion (BPEL) pour un service en sortie, procédez comme suit :

1. Dans Business Integration, développez le module pour voir son contenu. Pour ouvrir Assembly Editor, double-cliquez sur le premier élément sous le projet de module (même nom que le projet).
2. Ils existent plusieurs scénarios dans lesquels un processus BPEL peut appeler un autre processus BPEL. Recherchez le scénario adapté à votre application parmi les scénarios suivants :
 - Si le composant BPEL appelé se trouve dans le même module, créez une connexion à partir de la référence appropriée sur le premier composant BPEL vers l'interface appropriée sur le composant BPEL cible.
 - Si le composant BPEL appelé se trouve dans un autre module (celui-ci étant un projet de service migré) :
 - a. Créez une **exportation avec liaison SCA** pour le deuxième processus métier dans son diagramme de module.
 - b. Dans la vue Intégration métier, développez l'icône de l'assemblage du deuxième module dans le navigateur. Vous devez voir l'exportation que vous venez de créer.

- c. Effectuez un glisser-déposer de l'exportation de la vue Intégration métier sous le deuxième module vers l'éditeur d'assemblage ouvert du premier module. Une importation avec liaison SCA est créée dans le premier module. Si ce service a été publié dans WebSphere Studio Application Developer Integration Edition pour faire l'objet d'une exposition externe, voir la section "Création d'exports SCA afin d'accéder au service migré".
- d. Connectez la référence appropriée sur le premier processus métier à l'importation que vous venez de créer dans ce module.
- e. Enregistrez le diagramme de l'assemblage.
- Pour effectuer une liaison tardive lors de l'appel du second processus métier, procédez comme suit :
 - a. Ne connectez pas la référence de composant du premier processus métier. Ouvrez ce dernier dans l'éditeur BPEL puis, dans la section **Partenaires de référence**, sélectionnez le partenaire correspondant au second processus BPEL devant être appelé par la liaison tardive.
 - b. Dans la vue Propriétés de l'onglet **Description**, entrez le nom du second processus métier dans la zone **Modèle de processus**.
 - c. Enregistrez le processus métier. La configuration de l'appel par liaison tardive est maintenant terminée.

Migration d'un service Web (SOAP/JMS) :

Vous pouvez migrer un service Web (SOAP/JMS) dans une importation SCA avec liaison de services Web.

Pour migrer un projet de service SOAP/JMS sortant, procédez comme suit :

1. Vous devrez d'abord importer le projet de service à l'aide de l'Assistant de migration. Cela va créer un module Business Integration contenant les messages WSDL, les types de port, les connexions et les services créés dans WebSphere Studio Application Developer Integration Edition. Notez que si le service Web IBM (SOAP/JMS) que cette application va appeler est aussi un service Web WebSphere Studio Application Developer Integration Edition qui sera migré, des mises à jour de ce service Web peuvent avoir lieu lors de la migration. Dans ce cas, vous devez utiliser ici les fichiers WSDL migrés de ce service Web.
2. Dans Business Integration, développez le module pour voir son contenu. Pour ouvrir Assembly Editor, double-cliquez sur le premier élément sous le projet de module (même nom que le projet).
3. Ensuite, ajoutez une importation qui permettra à l'application d'interagir avec le service Web IBM (via SOAP/JMS) en fonction du modèle de programmation SCA. Vérifiez que l'interface WSDL, la connexion et les définitions de service sont présents dans le module migré ou dans une bibliothèque dont dépend le module migré.
4. Dans Business Integration, développez le module migré et ouvrez son diagramme d'assemblage dans Assembly Editor.
5. Développez la catégorie logique Ports de service Web et faites glisser le port qui correspond au service que vous voulez appeler dans Assembly Editor.
6. Sélectionnez **Importation avec liaison de services Web**.
7. Une fois l'importation créée, sélectionnez-la dans Assembly Editor et ouvrez la vue Propriétés. Dans l'onglet Connexion, vous verrez le port et le service auxquels l'importation est connectée.
8. Enregistrez le diagramme de l'assemblage.

Quand vous avez terminé, vous devez reconnecter le service :

- Si ce service est appelé par un processus métier dans le même module, créez une connexion entre la référence de processus métier appropriée et cette importation.
- Si ce service est appelé par un processus métier situé dans un autre module, créez une **exportation avec liaison SCA** et à partir de l'autre module, effectuez un glisser-déposer de cette exportation vers l'éditeur d'assemblage de ce module pour créer l'**importation avec liaison SCA** correspondante. Connectez la référence de processus métier appropriée à cette importation.

- Enregistrez le diagramme de l'assemblage.

Migration d'un service Web (SOAP/HTTP) :

Vous pouvez migrer un service Web (SOAP/HTTP) dans une importation SCA avec liaison de services Web.

Pour migrer un projet de service SOAP/HTTP sortant, procédez comme suit :

1. Vous devrez d'abord importer le projet de service à l'aide de l'Assistant de migration. Cela va créer un module Business Integration contenant les messages WSDL, les types de port, les connexions et les services créés dans WebSphere Studio Application Developer Integration Edition. Notez que si le service Web IBM (SOAP/HTTP) que cette application va appeler est aussi un service Web WebSphere Studio Application Developer Integration Edition qui sera migré, des mises à jour de ce service Web peuvent avoir lieu lors de la migration. Dans ce cas, vous devez utiliser ici les fichiers WSDL migrés de ce service Web.
2. Dans Business Integration, développez le module pour voir son contenu. Pour ouvrir Assembly Editor, double-cliquez sur le premier élément sous le projet de module (même nom que le projet).
3. Ensuite, ajoutez une importation qui permettra à l'application d'interagir avec le service Web IBM (via SOAP/HTTP) en fonction du modèle de programmation SCA. Vérifiez que l'interface WSDL, la connexion et les définitions de service sont présents dans le module migré ou dans une bibliothèque dont dépend le module migré.
4. Dans Business Integration, développez le module migré et ouvrez son diagramme d'assemblage dans Assembly Editor.
5. Développez la catégorie logique Ports de service Web et faites glisser le port qui correspond au service que vous voulez appeler dans Assembly Editor.
6. Sélectionnez **Importation avec liaison de services Web**.
7. Une fois l'importation créée, sélectionnez-la dans Assembly Editor et ouvrez la vue Propriétés. Dans l'onglet Connexion, vous verrez le port et le service auxquels l'importation est connectée.
8. Enregistrez le diagramme de l'assemblage.

Quand vous avez terminé, vous devez reconnecter le service :

- Si ce service est appelé par un processus métier dans le même module, créez une connexion entre la référence de processus métier appropriée et cette importation.
- Si ce service est appelé par un processus métier situé dans un autre module, créez une **exportation avec liaison SCA** et à partir de l'autre module, effectuez un glisser-déposer de cette exportation vers l'éditeur d'assemblage de ce module pour créer l'**importation avec liaison SCA** correspondante. Connectez la référence de processus métier appropriée à cette importation.
- Enregistrez le diagramme de l'assemblage.

Migration d'un service JMS :

Vous pouvez migrer un service JMS dans une importation SCA avec liaison JMS.

Remarque : Si le message JMS est envoyé à un adaptateur WebSphere Business Integration, voir "Migration des interactions avec les adaptateurs WebSphere Business Integration".

Pour migrer un projet de service JMS sortant, procédez comme suit :

1. Vous devrez d'abord importer le projet de service à l'aide de l'Assistant de migration. Cela va créer un module Business Integration contenant les messages WSDL, les types de port, les connexions et les services créés dans WebSphere Studio Application Developer Integration Edition.
2. Dans Business Integration, développez le module pour voir son contenu. Pour ouvrir Assembly Editor, double-cliquez sur le premier élément sous le projet de module (même nom que le projet).

3. Ensuite, ajoutez une importation qui permettra à l'application d'interagir avec une file d'attente JMS en fonction du modèle de programmation SCA.
4. Dans Assembly Editor, développez le projet de module migré et développez la catégorie **Interfaces** puis recherchez le type de port WSDL qui décrit le service Web que l'application doit appeler. Faites-le glisser dans l'éditeur d'assemblage.
5. Une boîte de dialogue **Création de composant** vous invite à sélectionner le type de composant à créer. Choisissez **Importation sans liaison**.
6. Une nouvelle importation est créée dans Assembly Editor. Sélectionnez-la puis ouvrez la vue Propriétés. Dans l'onglet Description, vous pouvez remplacer le nom et le nom d'affichage de l'importation par un nom plus descriptif.
7. Vous pouvez vous reporter aux fichiers de service et de liaison WSDL 5.1 pour plus de détails sur le service JMS en cours de migration. Ces fichiers peuvent être utilisés pour consigner des informations détaillées sur l'importation avec liaison JMS de la version 6.0. Recherchez-les dans le projet de service 5.1 (ils sont habituellement appelés *JMSBinding.wsdl et *JMSService.wsdl). Prenez connaissance des informations de liaison et de service qui s'y trouvent. Les informations sur la liaison permettent de déterminer si des messages de texte, des messages d'objet ou des liaisons de formats de données personnalisées ont été utilisées. Si tel est le cas, pensez également à créer une liaison de données personnalisée pour l'importation avec liaison JMS de la version 6.0. Les informations sur le service indiquent la fabrique de contextes initiale, le nom de la fabrique de connexions JNDI, le nom de destination JNDI et le style de destination (file d'attente).
8. Avec le bouton droit, cliquez sur l'importation et sélectionnez **Générer la connexion** puis **Connexion JMS**. Vous serez invité à entrer les paramètres suivants :

Sélectionnez le domaine de messagerie JMS :

- Point à point
- Publier/Abonner
- Indépendant du domaine

Sélectionnez la façon dont les données sont sérialisées entre l'objet métier et le message JMS :

- Texte
- Objet
- Fourni par l'utilisateur

Si vous sélectionnez Fourni par l'utilisateur :

Indiquez le nom qualifié complet de la classe d'implémentation com.ibm.websphere.sca.jms.data.JMSDataBinding. Vous devez indiquer une liaison de données définie par l'utilisateur si l'application doit définir des propriétés d'en-tête JMS qui ne sont habituellement pas disponibles dans la liaison d'importation JMS. Dans ce cas, vous pouvez créer une classe de liaison de données personnalisée étendant la liaison de données JMS standard "com.ibm.websphere.sca.jms.data.JMSDataBinding", et ajouter un code personnalisé permettant d'accéder directement à JMSMessage. Pour consulter les exemples JMS se trouvant dans "Creating and modifying bindings for import and export components", cliquez sur le lien ci-dessous.

La connectivité en entrée utilise la classe de sélecteur de fonction JMS par défaut :

<selected> ou <deselected>

9. Sélectionnez l'importation que vous venez de créer. Dans la vue Propriétés, sélectionnez l'onglet Connexion. Vous pouvez indiquer manuellement toutes les données de connexion en utilisant les valeurs que vous avez déjà indiquées dans WebSphere Studio Application Developer Integration Edition. Les données de connexion que vous pouvez indiquer sont les suivantes :
 - Connexion d'importation JMS (le plus important)
 - Connexion
 - Adaptateur de ressource
 - Destinations JMS

- Connexion de méthode

Quand vous avez terminé, vous devez reconnecter le service :

- Si ce service est appelé par un processus métier dans le même module, créez une connexion entre la référence de processus métier appropriée et cette importation.
- Si ce service est appelé par un processus métier situé dans un autre module, créez une **exportation avec liaison SCA** et à partir de l'autre module, effectuez un glisser-déposer de cette exportation vers l'éditeur d'assemblage de ce module pour créer l'**importation avec liaison SCA** correspondante. Connectez la référence de processus métier appropriée à cette importation.
- Enregistrez le diagramme de l'assemblage.

Migration d'un service J2C-IMS :

Vous pouvez migrer un service J2C-IMS dans une importation SCA avec liaison EIS ou une importation SCA avec liaison de services Web.

N'utilisez pas les artefacts de WebSphere Studio Application Developer Integration Edition qui ont été générés pour ce service IMS. Vous devrez recréer le service à l'aide des assistants disponibles dans WebSphere Integration Developer et reconnecter l'application manuellement.

Remarque : Activez la fonction de compilation automatique ou compilez le module manuellement.

Vous disposez des options suivantes :

Remarque : Pour les deux options, notez que si un service BPEL appelle ce service IMS, le BPEL devra changer légèrement car l'interface du service EIS sera sensiblement différente de l'ancienne interface 5.1. Pour cela, ouvrez l'éditeur de BPEL et modifiez le lien partenaire qui correspond au service EIS puis utilisez la nouvelle interface (fichier WSDL) quand vous exécuterez les étapes ci-après. Apportez les modifications nécessaires aux activités BPEL pour la nouvelle interface WSDL du service EIS.

Avantages et inconvénients de chacune des options de nouvelle connexion d'un service J2C-HOD :

Il existe des avantages et des inconvénients pour chacune des options de nouvelle connexion d'un service J2C-HOD.

La liste suivante décrit les options ainsi que les avantages et les inconvénients de chaque option :

- La première option utilise le composant SCA standard pour appeler le service HOD.
- La première option est soumise à certaines restrictions :
 - L'API de spécification SDO version 1 ne fournit pas l'accès au tableau d'octets C ou COBOL. Cela affectera les clients qui utilisent les multi-segments HOD.
 - La spécification SDO version 1 pour la sérialisation ne prend pas en charge la construction REDEFINE de langage COBOL ou la construction d'union de langage C.
- La deuxième option utilise l'approche JSR 109 standard pour la connexion au service HOD. Cette fonctionnalité est disponible avec Rational Application Developer.

Création d'une importation SCA pour appeler le service IMS : Option 1 :

Vous pouvez créer une importation SCA avec liaison EIS qui utilisera les objets DataObjects pour stocker le message ou les données afin de communiquer avec le système IMS.

Pour créer une importation SCA afin d'appeler le service IMS, procédez comme suit :

1. Créez un nouveau projet de module Business Integration pour contenir ce nouveau service IMS.

2. Pour recréer le service EIS, sélectionnez **Fichier → Nouveau → Autre → Business Integration → Reconnaissance des services d'entreprise**.
3. Cet assistant vous permet d'importer un service depuis un système EIS. Il est très proche de l'assistant de WebSphere Studio Application Developer Integration Edition qui créait le service EIS basé sur WSIF dans la version 5.1. Vous pouvez importer le nouvel adaptateur de ressource IMS J2C dans cet assistant. Naviguez jusqu'au répertoire d'installation de WebSphere Integration Developer et recherchez **Adaptateurs de ressource → ims15 → imsic09102.rar**.

Remarque : Voir le Centre de documentation pour plus d'informations sur les volets des propriétés et des opérations. Dans l'assistant Reconnaissance des services d'entreprise, quand vous ajoutez une opération, vous pouvez aussi créer des objets métier pour le type de données d'entrée ou de sortie de l'opération. Ceci requiert le fichier source C ou COBOL que vous avez utilisé dans l'assistant de WebSphere Studio Application Developer Integration Edition. Ces fichiers doivent avoir été copiés dans l'ancien projet de service pour que puissiez retrouver les fichiers source. Vous pouvez également importer les objets métier avec l'autre assistant. Pour cela, sélectionnez **Fichier → Nouveau → Autre → Business Integration → Enterprise Data Discovery**.

4. Une fois l'assistant terminé, ouvrez Business Integration et développez le module pour voir son contenu. Vous devez voir les nouveaux objets métier du module listés sous Types de données et les nouvelles interfaces listées sous Interfaces.
5. Pour ouvrir Assembly Editor, double-cliquez sur le premier élément sous le projet de module (même nom que le projet). Vous devez voir une importation sur la trame. Cette importation possède une liaison EIS et elle représente le service que vous venez de créer.

Voir la section "Création d'exportations SCA afin d'accéder au service migré" pour savoir comment publier ce service.

Créer un service Web avec le service J2C : option 2 :

Vous pouvez créer un service Web J2C et, si l'utilisateur de ce service est un composant SCA, consommer le service en tant que service Web IBM (SOAP/HTTP ou SOAP/JMS).

Pour créer un service Web avec le service J2C, procédez comme suit :

1. Pour créer le bean Java J2C, cliquez sur **Fichier → Nouveau → J2C → Bean Java J2C**
2. Choisissez la version 1.5 du **Connecteur IMS pour Java** et cliquez sur **Suivant**.
3. Cochez l'option **Connexion gérée** et entrez le nom de recherche JNDI. Cliquez sur **Suivant**.
4. Indiquez le projet, le module et le nom du nouveau bean Java. Le bean comprend une interface et une classe d'implémentation. Cliquez sur **Suivant**.
5. Ajoutez une méthode Java pour chaque fonction ou service auxquels vous voulez accéder depuis EIS. Vous pourrez ajouter d'autres méthodes ensuite dans l'éditeur source Java via la vue des fragments. Quand vous cliquez sur le bouton **Ajouter**, choisissez le nom puis la méthode et cliquez sur **Suivant**.
6. Vous pouvez maintenant cliquer sur **Parcourir**, pour réutiliser des types existants, ou sur **Nouveau**, pour lancer l'assistant de connexion de données Java de CICS/IMS (où vous pourrez référencer un fichier source COBOL ou C) et spécifier les types de données d'entrée ou de sortie.
7. Une fois créées les méthodes Java, cliquez sur **Suivant**.
8. Finissez les autres étapes de l'assistant pour créer votre bean Java J2C.
9. Cliquez sur **Fichier → Nouveau → J2C → Page Web, Service Web, ou EJB depuis Bean Java J2C** pour créer le service Web avec le bean Java J2C.
10. Terminez l'assistant.

Les consommateurs peuvent maintenant utiliser le service WSDL créé avec cet assistant pour appeler le service IMS.

Migration d'un service J2C-CICS ECI :

Vous pouvez migrer un service J2C-CICS ECI vers une importation SCA avec liaison EIS ou une importation SCA avec liaison de services Web.

Suivez les instructions contenues dans la section "Migration d'un projet de service J2C-HOD" ; vous devez importer le fichier RAR suivant *au lieu* du fichier RAR HOD :

- Accédez au répertoire d'installation de WebSphere Integration Developer et faites défiler la liste jusqu'à **Adaptateur de resfacts source** → **cics15** → **cicseci.rar**.

Si vous suivez la deuxième option pour créer un service Web J2C, sélectionnez l'adaptateur **ECIResourceAdapter** V1.5 dans le deuxième panneau de l'Assistant de création de bean Java J2C.

Voir aussi la rubrique concernant la migration d'un service J2C-IMS.

Migration d'un service J2C-CICS EPI :

Il n'existe pas de support direct pour le service J2C-CICS EPI dans WebSphere Integration Developer. Pour accéder à ce service depuis un module SCA, vous devrez effectuer une migration à l'aide du *scénario de consommation*.

Voir "Scénario de consommation pour la migration de service" pour savoir comment faire migrer ce type de service vers WebSphere Integration Developer.

Migration d'un service J2C-HOD :

Il n'existe pas de support direct pour le service J2C-HOD dans WebSphere Integration Developer. Pour accéder à ce service depuis un module SCA, vous devrez effectuer une migration à l'aide du *scénario de consommation*.

Voir "Scénario de consommation pour la migration de service" pour savoir comment faire migrer ce type de service vers WebSphere Integration Developer.

Migration d'un service de conversion :

Vous pouvez parfois migrer un service de transformateur dans une mappe de données SCA et une mappe d'interfaces. Vous pouvez également utiliser le *scénario de consommation* pour accéder à ce service depuis un module SCA.

Les composants mappe de données et mappe d'interfaces sont des nouveautés de la version 6.0. Ils offrent des fonctions identiques au service de transformateur de la version 5.1 mais n'ont pas les mêmes capacités de transformation XSL. Si vous ne pouvez pas remplacer votre service de transformateur par l'un de ces composants, vous devez effectuer la migration à l'aide du scénario de consommation car il n'existe pas de support direct pour le service de transformateur dans WebSphere Integration Developer. Suivez les étapes décrites dans la section "Scénario de consommation pour la migration de service" pour accéder à ce service depuis un module SCA.

Scénario de consommation pour la migration de services :

Lorsqu'il n'existe pas d'équivalent direct pour un type de service WebSphere Studio Application Developer Integration Edition, un scénario de consommation est nécessaire pour utiliser l'ancien service WebSphere Studio Application Developer Integration Edition tel quel lors de la nouvelle conception de l'application dans WebSphere Integration Developer.

Voici les étapes que vous devez suivre dans WebSphere Studio Application Developer Integration Edition *avant* d'appeler l'Assistant de migration :

1. Créez un projet Java pour conserver ce code de proxy client. Ne placez pas ce code de proxy client dans le projet de service car les classes de bean Java et les messages générés de type 5.1 seront ignorés par l'Assistant de migration automatique qui migre les projets de service.
2. Ouvrez WebSphere Studio Application Developer Integration Edition et cliquez avec le bouton droit de la souris sur le fichier WSDL contenant la connexion de transformateur ainsi que le service, puis sélectionnez **Services entreprise** → **Générer le proxy de service**. Vous devez indiquer le type de proxy à créer, mais seul le type **Web Services Invocation Framework (WSIF)** est disponible. Cliquez sur **Suivant**.
3. Vous pouvez maintenant indiquer le module et le nom de la classe Java de proxy de service à créer (vous créez le proxy dans le projet de service en cours). Cliquez sur **Suivant**.
4. Vous pouvez maintenant indiquer le style du proxy ; cliquez sur **Module de remplacement client**, sélectionnez les opérations que vous souhaitez inclure dans le proxy, puis cliquez sur **Terminer**. Cette opération crée une classe Java qui expose les mêmes méthodes que le service WebSphere Studio Application Developer Integration Edition, où les arguments pour les méthodes Java correspondent aux parties du message WSDL source.

Vous pouvez maintenant effectuer la migration vers WebSphere Integration Developer :

1. Copiez le projet Java du proxy client vers le nouvel espace de travail et importez-le à l'aide de l'option **Fichier** → **Importer** → **Projet existant dans l'espace de travail**.
2. Importez le projet de service à l'aide de l'Assistant de migration. Cette opération crée un module Business Integration avec les messages WSDL, les types de port, les connexions et les services générés dans WebSphere Studio Application Developer Integration Edition.
3. Dans Business Integration, développez le module pour voir son contenu. Pour ouvrir Assembly Editor, double-cliquez sur le premier élément sous le projet de module (même nom que le projet).
4. Pour créer le composant Java personnalisé, sous le projet de module, développez **Interfaces** et sélectionnez l'interface WSDL générée pour ce service de transformateur dans WebSphere Studio Application Developer Integration Edition.
5. Effectuez un glisser-déposer de cette interface vers l'éditeur d'assemblage. Une boîte de dialogue vous demande de sélectionner le type de composant à créer. Sélectionnez **Composant (sans type d'implémentation)** et cliquez sur **OK**.
6. Un composant générique apparaît dans le diagramme de l'assemblage. Sélectionnez-le et accédez à la vue **Propriétés**.
7. Dans l'onglet **Description**, vous pouvez remplacer le nom et le nom d'affichage du composant par un nom plus descriptif (dans le cas présent, attribuez-lui le nom de votre EJB et ajoutez un suffixe tel que "JavaMed", car ce composant Java servira d'intermédiaire entre l'interface WSDL générée pour le service de transformateur dans WebSphere Studio Application Developer Integration Edition et l'interface Java du proxy client du transformateur).
8. Dans l'onglet **Détails**, vous voyez que ce composant a une interface (celle pour laquelle vous avez effectué un glisser-déposer vers l'éditeur d'assemblage).
9. Dans l'éditeur d'assemblage, cliquez avec le bouton droit de la souris sur le composant que vous venez de créer et sélectionnez **Générer l'implémentation...** → **Java**. Sélectionnez ensuite le package dans lequel l'implémentation Java sera générée. Cette opération crée un service Java squelette conforme à l'interface WSDL selon le modèle de programmation SCA, où les types complexes sont représentés par un objet de type `commonj.sdo.DataObject` et où les types simples sont représentés par leurs équivalents objet Java.

Maintenant, vous devez entrer du code aux endroits où vous voyez les balises `//TODO` dans la classe d'implémentation Java générée. Il existe deux options :

1. Déplacez la logique de la classe Java d'origine vers cette classe, en l'adaptant à la nouvelle structure de données.
2. Créez une instance privée de l'ancienne classe Java à l'intérieur de cette classe Java générée et écrivez du code dans :

- a. Convertissez tous les paramètres de la classe d'implémentation Java générée en paramètres attendus par l'ancienne classe Java.
- b. Appelez l'instance privée de l'ancienne classe Java avec les paramètres convertis.
- c. Convertissez la valeur de retour de l'ancienne classe Java en type de valeur de retour déclaré par la méthode d'implémentation Java générée.

Une fois que vous avez exécuté les opérations ci-dessus, vous devez reconnecter le proxy client. Comme il ne doit exister aucune "référence", il vous suffit de reconnecter l'interface du composant Java :

- Si ce service est appelé par un processus métier situé dans le même module, vous devez créer une connexion entre la référence de processus métier appropriée et l'interface de ce composant Java.
- Si ce service est appelé par un processus métier situé dans un autre module, créez une **exportation avec liaison SCA** et à partir de l'autre module, effectuez un glisser-déposer de cette exportation vers l'éditeur d'assemblage de ce module pour créer l'**importation avec liaison SCA** correspondante. Connectez la référence de processus métier appropriée à cette importation.
- Si ce service a été publié dans WebSphere Studio Application Developer Integration Edition pour faire l'objet d'une exposition externe, voir la section "Création d'exportations SCA afin d'accéder au service migré" pour savoir comment le publier de nouveau.

Création d'exports SCA afin d'accéder au service migré :

Une exportation SCA doit être créée pour mettre le service migré à la disposition des consommateurs conformément au modèle SCA, et ce pour tous les services ayant fait l'objet d'un déploiement de code dans le projet de service WebSphere Studio Application Developer Integration Edition. Cela inclut tous les services appelés par les clients externes à l'application.

Si dans WebSphere Studio Application Developer Integration Edition, vous avez cliqué avec le bouton droit de la souris sur le processus BPEL ou un autre service WSDL et sélectionné **Services entreprise** → **Générer le code de déploiement**, vous devez suivre les étapes de migration manuelle ci-dessous. Notez que WebSphere Integration Developer est différent de WebSphere Studio Application Developer Integration Edition dans la mesure où il stocke toutes les options de déploiement. Lorsque le projet est créé, le code de déploiement est automatiquement mis à jour dans l'EJB et les projets Web générés. Il n'existe donc plus aucun moyen de **générer du code de déploiement** manuellement.

Cinq options de connexion ont été fournies dans la section Interfaces pour partenaires de l'assistant Générer un code de déploiement BPEL. Les informations de migration de service BPEL entrant suivantes fournissent des informations supplémentaires sur les propriétés et le type d'exportation à créer selon le ou les types de connexion de déploiement sélectionnés dans WebSphere Studio Application Developer Integration Edition :

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

Migration de la connexion EJB et de la connexion de processus EJB :

La connexion EJB et la connexion de processus EJB peuvent être migrées vers la construction SCA recommandée.

Dans WebSphere Studio Application Developer Integration Edition, ce type de connexion permettait aux clients de communiquer avec un processus BPEL ou un autre type de service en appelant un EJB. Notez que ce type de connexion n'était pas facultatif pour les microprocessus : il était toujours sélectionné dans la mesure où le EJB généré était utilisé en interne par les autres types de connexion.

Le nom JNDI de l'EJB généré était automatiquement généré comme une combinaison du nom du BPEL, de l'espace de nom cible et d'un horodatage de début de validité. Par exemple, ces attributs peuvent être trouvés en examinant les propriétés du processus BPEL dans les onglets Description et Contenu du serveur de l'éditeur BPEL :

Tableau 3. Espace de nom généré

Nom de processus	MonService
Espace de nom cible	http://www.example.com/process87787141/
Valide à partir de	01 jan 2003 02:03:04

L'espace de nom généré pour cet exemple est alors
com/example/www/process87787141/MonService01012003T020304.

Dans WebSphere Studio Application Developer Integration Edition, lorsque la connexion EJB était sélectionnée en tant que type de déploiement, aucune option n'était fournie.

Il existe quatre options pour la migration de la connexion de processus de WebSphere Studio Application Developer Integration Edition. Le type de client qui accède au service déterminera les options ci-dessous à exécuter :

Remarque : Une fois les étapes de migration manuelle terminées, le client doit également être migré vers le nouveau modèle de programmation. Voir la rubrique appropriée pour les types de clients suivants :

Tableau 4. Plus d'informations sur la migration des clients

Type de client	Pour plus d'informations, voir
Clients EJB qui appellent le bean de session généré. Ce type de client appelle une méthode EJB correspondant à l'opération BPEL à appeler.	Migration du client EJB
Client WSIF qui utilise la connexion de processus EJB.	Migration du client de connexion de processus EJB
API EJB générique de Business Process Choreographer	Migration d'un client utilisant l'API EJB générique de Business Process Choreographer
API de messagerie générique de Business Process Choreographer	Migration du client API de messagerie générique de Business Process Choreographer
Autre processus BPEL dans le même module	Non applicable : Connecter ensemble des composants BPEL à l'aide de l'éditeur d'assemblage
Autre processus BPEL dans un module différent	Non applicable : créer une Importation avec liaison SCA dans le module de référence et configurer ses connexions pour pointer vers l' Exportation avec liaison SCA que vous avez créée plus haut dans la première option.

Notez que si le processus métier transmet une référence à lui-même en dehors de son module (via une référence de service), vous devez toujours suivre la première option ci-dessous (vous pouvez toujours exécuter plusieurs de ces options) afin de créer une exportation avec liaison SCA pour le processus métier. Un seul processus métier par module peut transmettre sa référence de service à l'extérieur du module car son exportation doit être marquée comme exportation par défaut du module. Pour cela, affectez la valeur "true" à l'attribut appelé "default" d'une exportation, comme dans :

Référence de noeud final
par défaut

Vous devez marquer manuellement cette exportation de processus métier comme exportation par défaut en cliquant dessus avec le bouton droit de la souris dans la vue Intégration métier et en sélectionnant **Ouvrir avec**, puis **Editeur de texte**.

Migration pour EJB et pour les liaisons de processus EJB - Option 1 :

La première option de migration pour la connexion du processus EJB WebSphere Studio Application Developer Integration Edition consiste à rendre les processus métier accessibles à un autre composant dans le même module.

Dans l'éditeur d'assemblage, connectez cet autre composant au composant BPEL :

1. Sélectionnez l'élément **Connexion** dans la barre d'outils.
2. Cliquez sur l'autre composant pour le sélectionner en tant que source de la connexion.
3. Cliquez sur le composant **BPEL SCA** pour le sélectionner en tant que cible de la connexion.
4. Enregistrez le diagramme de l'assemblage.

Migration pour EJB et pour les liaisons de processus EJB - Option 2 :

La deuxième option de migration pour la connexion du processus EJB WebSphere Studio Application Developer Integration Edition consiste à rendre les processus métier accessibles à d'autres clients et modules SCA.

Remarque : Ces étapes sont obligatoires si les API génériques de Business Process Choreographer sont utilisées pour appeler le processus métier.

L'exportation avec liaison SCA permet à d'autres modules SCA d'accéder à un composant SCA. Pour créer une exportation avec liaison SCA :

1. Ouvrez l'éditeur d'assemblage pour le module créé par l'Assistant de migration.
2. Créez une exportation avec liaison SCA pour chaque interface du processus BPEL qui possède une connexion EJB générée dans WebSphere Studio Application Developer Integration Edition :
 - a. Cliquez avec le bouton droit de la souris sur le composant BPEL dans l'éditeur d'assemblage.
 - b. Sélectionnez **Exporter...**
 - c. Sélectionnez **Association SCA**.
 - d. S'il existe plusieurs interfaces pour le processus, sélectionnez la ou les interfaces à exporter avec ce type de connexion.
 - e. Une fois l'exportation SCA créée, sélectionnez-la dans l'éditeur d'assemblage et dans la vue Propriétés, sélectionnez le volet **Description**. Le nom de l'exportation et sa description sont répertoriés et peuvent être modifiés si nécessaire.
 - f. Enregistrez le diagramme de l'assemblage.

Migration pour EJB et pour les liaisons de processus EJB - Option 3 :

La troisième option de migration pour la connexion du processus EJB WebSphere Studio Application Developer Integration Edition consiste à rendre les modules accessibles à une entité non-SCA (par exemple, un JSP ou un client Java).

La référence autonome permet à tout client externe d'accéder à un composant SCA. Pour créer une référence autonome :

1. Ouvrez l'éditeur d'assemblage pour le module créé par l'Assistant de migration.
2. Créez une référence autonome pour chaque interface de processus BPEL qui possède une connexion EJB générée dans WebSphere Studio Application Developer Integration Edition :
 - a. Sélectionnez l'élément **Référence autonome** dans la barre d'outils.

- b. Cliquez sur le canevas de l'éditeur d'assemblage pour créer une entité SCA références autonomes.
- c. Sélectionnez l'élément **Connexion** dans la barre d'outils.
- d. Cliquez sur l'entité **Références autonomes** pour la sélectionner en tant que source de la connexion.
- e. Cliquez sur le composant **BPEL SCA** pour le sélectionner en tant que cible de la connexion.
- f. Un message d'alerte s'affiche : **Une référence correspondante va être créée sur le noeud source. Voulez-vous continuer ?**. Cliquez sur **OK**.
- g. Sélectionnez l'entité **Références autonomes** qui vient d'être créée et, dans la vue Propriétés, sélectionnez le volet **Description**.
- h. Développez le lien **Références** et sélectionnez la référence qui vient d'être créée. Le nom et la description de la référence sont répertoriés et peuvent être modifiés si nécessaire.
- i. S'il existe plusieurs interfaces pour le processus, sélectionnez la ou les interfaces à exporter avec ce type de connexion.
- j. Enregistrez le diagramme de l'assemblage.

Migration pour EJB et pour les liaisons de processus EJB - Option 4 :

La quatrième option de migration pour la connexion du processus EJB WebSphere Studio Application Developer Integration Edition consiste à rendre les processus métier accessibles à un client de services Web.

L'exportation avec liaison de services Web permet à un client de services Web externe d'accéder à un composant SCA. Pour créer une exportation avec liaison de services Web :

1. Ouvrez l'éditeur d'assemblage pour le module créé par l'Assistant de migration.
2. Créez une exportation avec liaison SCA pour chaque interface du processus BPEL qui possède une connexion EJB générée dans WebSphere Studio Application Developer Integration Edition :
 - a. Cliquez avec le bouton droit de la souris sur le composant BPEL dans l'éditeur d'assemblage.
 - b. Sélectionnez **Exporter...**
 - c. Sélectionnez **Association de services Web**.
 - d. S'il existe plusieurs interfaces pour le processus, sélectionnez la ou les interfaces à exporter avec ce type de connexion.
 - e. Sélectionnez le transport : **soap/http** ou **soap/jms**.
 - f. Une fois l'exportation de services Web créée, sélectionnez-la dans l'éditeur d'assemblage et dans la vue Propriétés, sélectionnez le volet **Description**. Le nom de l'exportation et sa description sont répertoriés et peuvent être modifiés si nécessaire.
 - g. Enregistrez le diagramme de l'assemblage.

Migration de la connexion JMS et de la connexion de processus JMS :

La connexion JMS et la connexion de processus JMS peuvent être migrées vers la construction SCA recommandée.

Dans WebSphere Studio Application Developer Integration Edition, ce type de connexion permettait aux clients de communiquer avec un processus BPEL ou un autre type de service en envoyant un message à un MDB. Notez que ce type de connexion n'était pas facultatif pour les processus longs car il était toujours sélectionné. En effet, ce type de connexion était le **seul** type de connexion autorisé pour les interfaces demande-réponse des processus longue durée. Pour les autres types de service, un MDB est généré et appelle le service approprié.

Le nom JNDI utilisé par la connexion JMS était une combinaison du nom du BPEL, de l'espace de nom cible et d'un horodatage de début de validité.

Dans WebSphere Studio Application Developer Integration Edition, lorsque la connexion JMS était sélectionnée comme type de déploiement pour un processus BPEL, les options suivantes étaient fournies :

- **Fabrique de connexion JNDI** : par défaut, jms/BPECF (il s'agit du nom JNDI de la fabrique de connexions de la file d'attente du conteneur du processus métier)
- **File d'attente de la destination JNDI** : par défaut, jms/BPEIntQueue (il s'agit du nom JNDI de la file d'attente interne du conteneur du processus métier cible)
- **URL de fournisseur JNDI : fournie par le serveur ou personnalisée** : vous devez entrer une adresse. L'adresse par défaut est iiop://localhost:2809.

Il existe cinq options pour la migration de la connexion de processus JMS de WebSphere Studio Application Developer Integration Edition. Le type de client qui accède au service déterminera les options ci-dessous à exécuter :

Remarque : Une fois les étapes de migration manuelle terminées, le client doit également être migré vers le nouveau modèle de programmation. Voir la rubrique appropriée pour les types de clients suivants :

Tableau 5. Plus d'informations sur la migration des clients

Type de client	Pour plus d'informations, voir
Client WSIF qui utilise la connexion de processus JMS	Migration du client API de messagerie générique de Business Process Choreographer et du client de connexion de processus JMS
API EJB générique de Business Process Choreographer	Migration du client API EJB générique de Business Process Choreographer
Migration métier via l'API de messagerie générique de Business Process Choreographer	Migration du client API de messagerie générique de Business Process Choreographer
Autre processus BPEL dans le même module	Non applicable : connecter ensemble des composants BPEL à l'aide de l'éditeur d'assemblage.
Autre processus BPEL dans un module différent	Non applicable : Créer une importation avec liaison SCA dans le module de référence et configurer ses connexions pour pointer vers l'exportation avec liaison SCA que vous avez créé plus haut dans la première option.

Notez que si le processus métier transmet une référence à lui-même en dehors de son module (via une référence de service), vous devez toujours suivre la première option ci-dessous (vous pouvez toujours exécuter plusieurs de ces options) afin de créer une exportation avec liaison SCA pour le processus métier. Un seul processus métier par module peut transmettre sa référence de service à l'extérieur du module car son exportation doit être marquée comme exportation par défaut du module. Pour cela, affectez la valeur "true" à l'attribut appelé "default" d'une exportation, comme dans :

Référence de noeud final par défaut

Vous devez marquer manuellement cette exportation de processus métier comme exportation par défaut en cliquant dessus avec le bouton droit de la souris dans la vue Intégration métier et en sélectionnant **Ouvrir avec**, puis **Editeur de texte**.

Migration pour JMS et pour les liaisons de processus JMS - Option 1 :

La première option de migration pour la connexion de processus JMS WebSphere Studio Application Developer Integration Edition est d'autoriser l'accès aux processus métier pour un autre composant dans le même module.

Dans l'éditeur d'assemblage, connectez cet autre composant au composant BPEL :

1. Sélectionnez l'option **Connexion** dans la barre d'outils.
2. Cliquez sur l'autre composant pour le sélectionner en tant que source de la connexion.
3. Cliquez sur le composant **BPEL SCA** pour le sélectionner en tant que cible de la connexion.
4. Enregistrez le diagramme de l'assemblage.

Migration pour JMS et pour les liaisons de processus JMS - Option 2 :

La deuxième option de migration pour la connexion de processus JMS WebSphere Studio Application Developer Integration Edition est d'autoriser l'accès aux processus métier pour d'autres clients et modules SCA.

L'exportation avec liaison SCA permet à d'autres modules SCA d'accéder à un composant SCA. Pour créer une exportation avec liaison SCA :

1. Ouvrez l'éditeur d'assemblage pour le module créé par l'Assistant de migration.
2. Créez une exportation avec liaison SCA pour chaque interface du processus BPEL qui possédait une connexion JMS générée dans WebSphere Studio Application Developer Integration Edition :
 - a. Cliquez avec le bouton droit de la souris sur le composant BPEL dans l'éditeur d'assemblage.
 - b. Sélectionnez **Exporter...**
 - c. Sélectionnez **Association SCA**.
 - d. S'il existe plusieurs interfaces pour le processus, sélectionnez la ou les interface(s) à exporter avec ce type de connexion.
 - e. Une fois l'exportation SCA créée, sélectionnez-la dans l'éditeur d'assemblage et dans la vue Propriétés, sélectionnez le volet **Description**. Le nom de l'exportation et sa description sont répertoriés et peuvent être modifiés si nécessaire.
 - f. Enregistrez le diagramme de l'assemblage.

Migration pour JMS et pour les liaisons de processus JMS - Option 3 :

La troisième option de migration pour la connexion de processus JMS WebSphere est d'autoriser l'accès aux processus métier pour une entité non-SCA (par exemple, un JSP ou un client Java).

La référence autonome autorise l'accès à un composant SCA pour tout client externe. Pour créer une référence autonome :

1. Ouvrez l'éditeur d'assemblage pour le module créé par l'Assistant de migration.
2. Créez une référence autonome pour chaque interface de processus BPEL qui possédait une connexion JMS générée dans WebSphere Studio Application Developer Integration Edition :
 - a. Sélectionnez l'élément **Références autonomes** dans la barre d'outils.
 - b. Cliquez sur le canevas de l'éditeur d'assemblage pour créer une entité SCA références autonomes.
 - c. Sélectionnez l'élément **Connexion** dans la barre d'outils.
 - d. Cliquez sur l'entité **Références autonomes** pour la sélectionner en tant que source de la connexion.
 - e. Cliquez sur le composant **BPEL SCA** pour le sélectionner en tant que cible de la connexion.
 - f. Un message d'alerte s'affiche : **Une référence correspondante va être créée sur le noeud source. Voulez-vous continuer ?**. Cliquez sur **OK**.
 - g. Sélectionnez l'entité **Références autonomes** qui vient d'être créée et, dans la vue Propriétés, sélectionnez le volet **Description**.
 - h. Développez le lien **Références** et sélectionnez la référence qui vient d'être créée. Le nom et la description de la référence sont répertoriés et peuvent être modifiés si nécessaire.
 - i. S'il existe plusieurs interfaces pour le processus, sélectionnez la ou les interfaces à exporter avec ce type de connexion.
 - j. Enregistrez le diagramme de l'assemblage.

Migration pour JMS et pour les liaisons de processus JMS - Option 4 :

La quatrième option de migration pour la connexion de processus JMS WebSphere Studio Application Developer Integration Edition est d'autoriser l'accès aux processus métier pour un client de services Web.

L'exportation avec liaison de services Web permet à un client de services Web externe d'accéder à un composant SCA. Pour créer une exportation avec liaison de services Web :

1. Ouvrez l'éditeur d'assemblage pour le module créé par l'Assistant de migration.
2. Créez une exportation avec liaison SCA pour chaque interface du processus BPEL qui possédait une connexion JMS générée dans WebSphere Studio Application Developer Integration Edition :
 - a. Cliquez avec le bouton droit de la souris sur le composant BPEL dans l'éditeur d'assemblage.
 - b. Sélectionnez **Exporter...**
 - c. Sélectionnez **Association de services Web**.
 - d. S'il existe plusieurs interfaces pour le processus, sélectionnez la ou les interface(s) à exporter avec ce type de connexion.
 - e. Sélectionnez le transport : **soap/http** ou **soap/jms**.
 - f. Une fois l'exportation de services Web créée, sélectionnez-la dans l'éditeur d'assemblage et dans la vue Propriétés, sélectionnez le volet **Description**. Le nom de l'exportation et sa description sont répertoriés et peuvent être modifiés si nécessaire.
 - g. Enregistrez le diagramme de l'assemblage.

Migration pour JMS et pour les liaisons de processus JMS - Option 5 :

La cinquième option de migration disponible pour la liaison des processus JMS WebSphere Studio Application Developer Integration Edition consiste à permettre au client JMS d'accéder aux processus métier.

L'exportation avec liaison JMS rend un composant SCA accessible par un client JMS externe. Pour créer une exportation avec liaison JMS :

1. La liaison de processus JMS 5.1 étant très différente de la liaison JMS 5.1 standard, vous devez créer et référencer de nouvelles resfacs source de file d'attente pour les services BPEL. Pour les services non BPEL, vous pouvez consulter les valeurs que vous avez sélectionnées pour le code de déploiement JMS dans WebSphere Studio Application Developer Integration Edition 5.1. Pour ce faire, recherchez les fichiers **JMSBinding.wsdl** et **JMSService.wsdl** dans le package approprié du dossier **ejbModule/META-INF** correspondant au projet EJB généré. Consultez les informations relatives à la liaison et au service contenues dans ce dossier. Les informations sur la liaison permettent de déterminer si des messages de texte, des messages d'objet ou des liaisons de formats de données personnalisées ont été utilisées. Si tel est le cas, pensez également à créer une liaison personnalisée de données pour l'**exportation avec liaison JMS** de la version 6.0. Les informations sur le service indiquent la fabrique de contextes initiale, le nom de la fabrique de connexions JNDI, le nom de destination JNDI et le style de destination (file d'attente).
2. Ouvrez l'éditeur d'assemblage pour le module créé par l'Assistant de migration.
3. Créez une exportation avec liaison JMS pour chaque interface du processus BPEL qui possédait une connexion JMS générée dans WebSphere Studio Application Developer Integration Edition.
4. Sélectionnez **Exporter...**
5. Sélectionnez **Exportation avec liaison JMS**.
6. S'il existe plusieurs interfaces pour le processus, sélectionnez la ou les interface(s) à exporter avec ce type de connexion.
7. Dans l'écran suivant (attributs de connexion d'exportation JMS), sélectionnez **Domaine de messagerie JMS**. Affectez à cet attribut la valeur **De point à point**.
8. Sélectionnez **la façon dont les données sont sérialisées entre l'objet métier et le message JMS** puis entrez les valeurs indiquées ci-après. Il est recommandé de sélectionner **Texte** plutôt qu'**Objet**.

L'option Texte correspond généralement au format XML et est indépendante de l'exécution. Elle permet l'intégration des services entre différents systèmes hétérogènes.

- a. Pour **Texte**, sélectionnez le **sélecteur de fonctions JMS par défaut** ou entrez le nom entièrement qualifié de la classe d'implémentation FunctionSelector.
 - b. Pour **Objet**, sélectionnez le **sélecteur de fonctions JMS par défaut** ou entrez le nom entièrement qualifié de la classe d'implémentation FunctionSelector.
 - c. Pour **Fourni par l'utilisateur**, entrez le nom entièrement qualifié de la classe d'implémentation de JMSDataBinding. Vous devez sélectionner **Fourni par l'utilisateur** si l'application doit accéder à des propriétés d'en-tête JMS qui ne sont pas disponibles dans la liaison d'importation JMS. Dans ce cas, vous devez créer une classe de liaison de données personnalisée étendant la liaison de données JMS standard **com.ibm.websphere.sca.jms.data.JMSDataBinding**, et ajouter un code personnalisé permettant d'accéder directement au message JMS. Indiquez ensuite le nom de la classe personnalisée pour cette zone. Pour consulter les exemples JMS se trouvant dans "Creating and modifying bindings for import and export components", cliquez sur le lien ci-dessous.
 - d. Pour **Fourni par l'utilisateur**, sélectionnez le **sélecteur de fonctions JMS par défaut** ou entrez le nom entièrement qualifié de la classe d'implémentation FunctionSelector.
9. Une fois l'exportation JMS créée, sélectionnez-la dans l'éditeur d'assemblage et sélectionnez le volet **Description** dans la vue Propriétés. Le nom de l'exportation et sa description sont répertoriés et peuvent être modifiés si nécessaire.
 10. Sélectionnez le volet Connexion pour afficher de nombreuses autres options.
 11. Enregistrez le diagramme de l'assemblage.

Migration de la liaison de services Web IBM (SOAP/JMS) :

La liaison de services Web IBM (SOAP/JMS) pour un processus BPEL ou un autre type de service peut être migrée vers la construction SCA recommandée.

Dans WebSphere Studio Application Developer Integration Edition, ce type de connexion permettait aux clients de communiquer avec un processus BPEL ou un autre type de service en appelant un service Web IBM, où le protocole de communication était JMS et le message observait les règles de codage SOAP.

L'exemple suivant montre certaines conventions utilisées lors de la génération d'un service Web IBM (SOAP/JMS) pour un service BPEL 5.1. Le nom JNDI du service Web IBM généré était une combinaison du nom du BPEL, de l'espace de nom cible et d'un horodatage de début de validité, ainsi que du nom de l'interface (type de port WSDL pour lequel le code de déploiement a été généré). Par exemple, ces attributs peuvent être trouvés en examinant les propriétés du processus BPEL dans les onglets Description et Contenu du serveur de l'éditeur BPEL :

Tableau 6. Espace de nom généré

Nom de processus	MonService
Espace de nom cible	http://www.example.com/process87787141/
Valide à partir de	01 jan 2003 02:03:04
Interface	Type de port du processus

L'espace de nom généré pour cet exemple est alors
com/example/www/process87787141/MonService01012003T020304PT.

Dans WebSphere Studio Application Developer Integration Edition, lorsque la liaison de services Web IBM (SOAP/JMS) était sélectionnée en tant que type de déploiement pour le processus BPEL ou tout autre type de service, les options suivantes étaient fournies :

- Pour le style de document, l'option par défaut était **DOCUMENT** / autre option : **RPC**
- Pour l'utilisation du document, l'option par défaut était **LITERAL** / autre option : **ENCODED**

- Pour l'URL fournisseur JNDI, il s'agissait de **Fourni par le serveur** ou de **Personnalisé** (une adresse doit être saisie, l'adresse par défaut étant `iiop://localhost:2809`)
- Pour le style de destination, l'option par défaut était **file d'attente** / l'autre option était **rubrique**
- Pour la fabrique de connexion JNDI, l'option par défaut était **jms/qcf** (il s'agit du nom JNDI de la fabrique de connexion de file d'attente pour la file d'attente MDB générée)
- Pour la file d'attente de destination JNDI, l'option par défaut était **jms/queue** (il s'agit du nom JNDI de la file d'attente MDB générée)
- Pour le port d'écoute MDB, l'option par défaut était `<Nom projet de service>Port d'écoute Mdb`

Un fichier WSDL spécifiant le service et la liaison SOAP/JMS de service Web IBM est créé dans le projet EJB généré mais pas dans le projet de service lui-même. Cela signifie que vous devez localiser manuellement ce fichier et le copier dans votre projet de module d'intégration métier si le code client du service Web IBM ne doit pas être modifié. Par défaut, ce fichier WSDL a été créé dans le projet EJB à l'emplacement `ejbModule/META-INF/wsdl/<nom du processus métier>_<nom du type de port de l'interface de processus métier>_JMS.wsdl`.

Le type de port WSDL et les messages de l'interface processus métier sont également copiés dans ce fichier WSDL, plutôt que de référencer le type de port WSDL et les messages existants définis dans le projet de service.

S'il est important que le code client de services Web IBM reste inchangé après la migration, les informations contenues dans ce fichier seront nécessaires pour les étapes de migration manuelle ci-dessous.

Il existe deux options pour la migration de la connexion de processus SOAP/JMS WebSphere Studio Application Developer Integration Edition. Vous devrez choisir entre migrer le client vers le modèle de programmation SCA ou le conserver comme client de services Web :

Remarque : Une fois les étapes de migration manuelle terminées, le client doit également être migré vers le nouveau modèle de programmation. Voir la rubrique appropriée pour les types de clients suivants :

Tableau 7. Plus d'informations sur la migration des clients

Type de client	Pour plus d'informations, voir
Client de services Web IBM	Migration du client (SOAP/JMS) de services Web IBM

Migration de la liaison de services Web IBM (SOAP/JMS) - Option 1 :

La première option de migration pour la connexion SOAP/JMS de WebSphere Studio Application Developer Integration Edition consiste à rendre le service accessible à un client de services Web.

L'exportation avec liaison de services Web permet à un client de services Web externe d'accéder à un composant SCA. Pour créer une exportation avec liaison de services Web, procédez comme suit :

1. Ouvrez l'éditeur d'assemblage pour le module créé par l'Assistant de migration.
2. Créez une exportation avec liaison SCA pour chaque interface de service pour laquelle une connexion (SOAP/JMS) de service Web IBM a été générée dans WebSphere Studio Application Developer Integration Edition :
 - a. Cliquez avec le bouton droit de la souris sur le composant SCA dans l'éditeur d'assemblage.
 - b. Sélectionnez **Exportation....**
 - c. Sélectionnez **Liaison de service Web**.
 - d. Si le composant est associé à plusieurs interfaces, sélectionnez celle(s) que vous souhaitez exporter avec ce type de connexion.

- e. Sélectionnez le transport **SOAP/JMS**.
3. Une fois l'exportation de services Web créée, sélectionnez-la dans l'éditeur d'assemblage et dans la vue Propriétés, sélectionnez le volet **Description**. Le nom de l'exportation et sa description sont répertoriés et peuvent être modifiés si nécessaire.
4. Enregistrez le diagramme de l'assemblage.
5. Sélectionnez le panneau du contenu Connexion. Observez qu'un élément connexion et service de type IBM Web Service WSDL a été directement généré dans le dossier projet du module. Il est nommé *composant-exporté* Export Nom Typeport WSDL Jms_Service.wsdl. Si vous examinez ce fichier, vous trouverez que la connexion Document/Literal wrapped est utilisée par défaut, puisqu'il s'agit du style préféré dans la version 6.0. Il s'agit du WSDL que les clients des services Web IBM utilisent pour appeler le service.
6. Suivez les étapes suivantes pour générer une nouvelle liaison de services Web et un service si vous voulez préserver le code client :
 - a. Copiez le fichier WSDL 5.1 dans le projet EJB 5.1 généré dans ejbModule/META-INF/wsdl/nom_processus métier/type_port_interface_processus métier]MS.wsdl vers le projet de module Business Integration.
 - b. Après avoir copié le fichier et recompilé le module, vous verrez peut-être des messages d'erreur car les types de schéma XML, les messages WSDL et les types de port WSDL utilisés par le service Web sont dupliqués dans le fichier WSDL du service Web IBM dans la version 5.1. Pour régler ce problème, supprimez les définitions en double du fichier WSDL du service ou de la liaison de services Web IBM et ajoutez à la place une importation WSDL pour l'interface WSDL réelle.
Remarque : Notez que quand WebSphere Studio Application Developer Integration Edition a généré le code de déploiement du service Web IBM, les définitions de schéma ont aussi été modifiées dans certains cas. Cela peut causer des incohérences pour les clients existants qui utilisent le service Web WSDL IBM. Par exemple, l'attribut de schéma "elementFormDefault" a pris la valeur "qualified" dans le schéma en ligne généré dans le fichier WSDL de service Web IBM si la définition de schéma d'origine n'était pas qualifiée. L'erreur suivante sera générée au cours de l'exécution : WWS3047E: Erreur : Impossible de désérialiser l'élément.
 - c. Avec le bouton droit, cliquez sur le fichier WSDL que vous venez de copier dans le module Business Integration et sélectionnez **Ouvrir avec puis Editeur WSDL**.
 - d. Accédez à l'onglet Source. Supprimez tous les messages et types de port définis dans ce fichier.
 - e. Vous voyez maintenant l'erreur "Le type de port '<TypePort' indiqué pour la connexion '<Connexion>' n'est pas défini. Pour régler ce problème, dans l'éditeur WSDL, dans l'onglet Graphique, cliquez avec le bouton droit dans la section Imports et sélectionnez **Ajouter une importation**.
 - f. Dans la vue Propriétés de l'onglet Propriétés générales, cliquez sur le bouton ... situé à droite de la zone Emplacement. Passez dans l'interface WSDL où résident le message WSDL et les définitions des types de port puis cliquez sur **OK** pour importer l'interface WSDL dans le fichier WSDL de la connexion/du service.
 - g. Enregistrez le fichier WSDL.
 - h. Réactualisez/recompilez le projet. Passez dans Business Integration. Ouvrez le diagramme de l'assemblage du module dans Assembly Editor.
 - i. Dans la vue de l'explorateur de projet, développez le module que vous migrez puis développez la catégorie logique **Ports de service Web**. Vous devez voir apparaître le port qui existe dans le fichier WSDL de la connexion/du service. Faites-le glisser dans l'éditeur d'assemblage.
 - j. Choisissez de créer une **exportation avec liaison de services Web** et sélectionnez le nom de port approprié. Ceci va créer une exportation utilisant l'ancienne liaison/service pour que les clients du service Web actuels ne demandent pas de modification. Si vous sélectionnez l'exportation que vous venez de créer dans Assembly Editor et que vous passez dans la vue Propriétés, vous devez voir dans l'onglet Connexion que les noms de service et le port 5.1 ont été entrés pour vous.
 - k. Enregistrez toutes les modifications.

- l. Juste avant de déployer l'application, vous pouvez changer la configuration du projet Web généré pour refléter l'adresse du service version 5.1 (vous devez faire ces modifications chaque fois que vous modifiez le module SCA qui cause la régénération de ce fichier). Si vous regardez la définition *Service* du fichier WSDL du service Web IBM que vous réutilisez depuis la version 5.1, vous verrez l'adresse de service codée dans le client 5.1 :
`<wsdl:soap:address location="http://localhost:9080/MyServiceWeb/services/MyServicePort"/>`
- m. Pour que les artefacts du projet Web 6.0 généré utilisent cette ancienne adresse de service, vous devez modifier le descripteur de déploiement du projet Web généré. Ouvrez le descripteur de déploiement dans WebSphere Integration Developer et, dans l'onglet Servlets, ajoutez un autre mappage d'URL proche du mappage d'URL existant pour cette exportation, avec le même nom de servlet mais avec un autre modèle d'URL.
- n. Notez aussi que si vous devez modifier la racine de contexte de ce projet Web pour qu'elle reflète la racine de contexte de l'adresse de service d'origine (dans cet exemple, la racine de contexte est "MyServiceWeb"), vous pouvez alors ouvrir le descripteur de déploiement de l'application J2EE Enterprise où réside ce projet Web et modifier la racine de contexte de ce module Web pour refléter celle de l'ancienne adresse de service. Vous verrez peut-être l'erreur suivante que vous pouvez ignorer : CHKJ3017E: Le projet Web <nom_projet web> est mappé à une racine de contexte non valide : <nouvelle racine de contexte> dans le projet EAR :
`<nom_application>`.

Migration de la liaison de services Web IBM (SOAP/JMS) - Option 2 :

La seconde option de migration pour la connexion de processus SOAP/JMS de WebSphere Studio Application Developer Integration Edition consiste à rendre des processus métier accessibles à une entité non-SCA (par exemple, JSP ou un client Java).

La référence autonome permet de rendre un composant SCA accessible à tout type de client externe. Pour créer une référence autonome, procédez comme suit :

1. Ouvrez l'éditeur d'assemblage pour le module créé par l'Assistant de migration.
2. Créez une référence autonome pour chaque interface de processus BPEL dotée d'une connexion (SOAP/JMS) de service Web IBM générée pour elle dans WebSphere Studio Application Developer Integration Edition :
 - a. Sélectionnez l'élément **Références autonomes** dans la barre d'outils.
 - b. Cliquez sur le canevas de l'éditeur d'assemblage pour créer une entité SCA de références autonomes :
 - c. Sélectionnez l'élément **Connexion** dans la barre d'outils.
 - d. Cliquez sur l'entité **Références autonomes** pour la sélectionner en tant que source de connexion.
 - e. Cliquez sur le composant **BPEL SCA** pour le sélectionner en tant que cible de connexion.
 - f. Un message d'alerte s'affiche : **Une référence correspondante va être créée sur le noeud source. Voulez-vous continuer ?**. Cliquez sur **OK**.
 - g. Sélectionnez l'entité **Références autonomes** qui vient d'être créée puis, dans la vue des propriétés, sélectionnez le panneau de contenu **Description**.
 - h. Développez le lien **Références** et sélectionnez la référence qui vient d'être créée. Le nom et la description de la référence sont répertoriés et peuvent être modifiés le cas échéant.
 - i. Si le processus est associé à plusieurs interfaces, sélectionnez celle(s) que vous souhaitez exporter avec ce type de connexion.
 - j. Enregistrez le diagramme de l'assemblage.

Migration de la liaison de services Web IBM (SOAP/HTTP) :

La liaison de services Web IBM (SOAP/HTTP) pour un processus BPEL ou un autre type de service peut être migrée vers la construction SCA recommandée.

Dans WebSphere Studio Application Developer Integration Edition, ce type de connexion permettait aux clients de communiquer avec un processus BPEL ou un autre type de service en appelant un service Web IBM, où le protocole de communication était HTTP et le message observait les règles de codage SOAP.

L'exemple suivant montre certaines conventions utilisées lors de la génération d'un service Web IBM (SOAP/HTTP) pour un service BPEL 5.1. Le nom JNDI du service Web IBM généré était une combinaison du nom du BPEL, de l'espace de nom cible et d'un horodatage de début de validité, ainsi que du nom de l'interface (type de port WSDL pour lequel le code de déploiement a été généré). Par exemple, ces attributs peuvent être trouvés en examinant les propriétés du processus BPEL dans les onglets Description et Contenu du serveur de l'éditeur BPEL :

Tableau 8. Espace de nom généré

Nom de processus	MonService
Espace de nom cible	http://www.example.com/process87787141/
Valide à partir de	01 jan 2003 02:03:04
Interface	Type de port du processus

L'espace de nom généré pour cet exemple est alors
com/example/www/process87787141/MonService01012003T020304PT.

Dans WebSphere Studio Application Developer Integration Edition, lorsque la liaison de services Web IBM (SOAP/HTTP) était sélectionnée en tant que type de déploiement pour le processus BPEL ou tout autre type de service, les options suivantes étaient fournies :

- Pour le style de document, l'option par défaut était **RPC / autre option : DOCUMENT**
- Pour l'utilisation du document, l'option par défaut était **ENCODED / autre option : LITERAL**
- Pour l'adresse de routeur, l'option par défaut était http://localhost:9080

Un fichier WSDL spécifiant le service et la liaison SOAP/HTTP de service Web IBM est créé dans les projets EJB et Web générés mais pas dans le projet de service lui-même. Cela signifie que vous devez localiser manuellement ce fichier et le copier dans votre projet de module d'intégration métier si le code client du service Web IBM ne doit pas être modifié. Par défaut, ce fichier WSDL a été créé dans le projet Web à l'emplacement WebContent/WEB-INF/wsd1/<nom du processus métier>_<nom du type de port de l'interface de processus métier>_HTTP.wsd1.

Le type de port WSDL et les messages de l'interface processus métier sont également copiés dans ce fichier WSDL, plutôt que de référencer le type de port WSDL et les messages existants définis dans le projet de service.

S'il est important que le code client de services Web IBM reste inchangé après la migration, les informations contenues dans ce fichier seront nécessaires pour les étapes de migration manuelle ci-dessous.

Il existe deux options pour la migration de la connexion de processus SOAP/HTTP WebSphere Studio Application Developer Integration Edition. Vous devrez choisir entre migrer le client vers le modèle de programmation SCA ou le conserver comme client de services Web :

Remarque : Une fois les étapes de migration manuelle terminées, le client doit également être migré vers le nouveau modèle de programmation. Voir la rubrique appropriée pour les types de clients suivants :

Tableau 9. Plus d'informations sur la migration des clients

Type de client	Pour plus d'informations, voir
Client de services Web IBM	Migration du client (SOAP/HTTP) de services Web IBM

Migration de la liaison de services Web IBM (SOAP/HTTP) - Option 1 :

La première option de migration pour connexion de processus SOAP/HTTP de WebSphere Studio Application Developer Integration Edition consiste à rendre des processus métier accessibles à un client de services Web.

L'exportation avec liaison de services Web permet à un client de services Web externe d'accéder à un composant SCA. Pour créer une exportation avec liaison de services Web, procédez comme suit :

1. Ouvrez l'éditeur d'assemblage pour le module créé par l'Assistant de migration.
2. Créez une exportation avec liaison SCA pour chaque interface de processus BPEL dotée d'une connexion (SOAP/HTTP) de service Web IBM générée pour elle dans WebSphere Studio Application Developer Integration Edition en cliquant avec le bouton droit de la souris sur le composant BPEL dans l'éditeur d'assemblage.
3. Sélectionnez **Exportation...**
4. Sélectionnez **Liaison de services Web**.
5. Si le composant est associé à plusieurs interfaces, sélectionnez celle(s) que vous souhaitez exporter avec ce type de connexion.
6. Sélectionnez le transport **SOAP/HTTP**.
7. Une fois l'exportation de services Web créée, sélectionnez-la dans l'éditeur d'assemblage et dans la vue Propriétés, sélectionnez le volet **Description**. Le nom de l'exportation et sa description sont répertoriés et peuvent être modifiés si nécessaire.
8. Enregistrez le diagramme de l'assemblage.
9. Suivez les étapes suivantes pour générer une nouvelle liaison de services Web et un service si vous voulez préserver le code client :
 - a. Copiez le fichier WSDL 5.1 depuis le projet EJB généré sous la version 5.1 dans le fichier `ejbModule/META-INF/wsdl/nom_processus_métier/type_port_interface_processus_métier_HTTP.wsdl` vers le projet de module Business Integration.
 - b. Après avoir copié le fichier et recompilé le module, vous verrez peut-être des messages d'erreur car les types de schéma XML, les messages WSDL et les types de port WSDL utilisés par le service Web sont dupliqués dans le fichier WSDL du service Web IBM dans la version 5.1. Pour régler ce problème, supprimez les définitions en double du fichier WSDL du service ou de la liaison de services Web IBM et ajoutez à la place une importation WSDL pour l'interface WSDL réelle.
Remarque : Notez que quand WebSphere Studio Application Developer Integration Edition a généré le code de déploiement du service Web IBM, les définitions de schéma ont aussi été modifiées dans certains cas. Cela peut causer des incohérences pour les clients existants qui utilisent le service Web WSDL IBM. Par exemple, l'attribut de schéma "elementFormDefault" a pris la valeur "qualified" dans le schéma en ligne généré dans le fichier WSDL de service Web IBM si la définition de schéma d'origine n'était pas qualifiée. L'erreur suivante sera générée au cours de l'exécution : WWS3047E: Erreur : Impossible de désérialiser l'élément.
 - c. Avec le bouton droit, cliquez sur le fichier WSDL que vous venez de copier dans le module Business Integration et sélectionnez **Ouvrir avec** puis **Editeur WSDL**.
 - d. Accédez à l'onglet Source. Supprimez tous les messages et types de port définis dans ce fichier.
 - e. Vous voyez maintenant l'erreur "Le type de port '<TypePort' indiqué pour la connexion '<Connexion>' n'est pas défini. Pour régler ce problème, dans l'éditeur WSDL, dans l'onglet Graphique, cliquez avec le bouton droit dans la section Imports et sélectionnez **Ajouter une importation**.
 - f. Dans la vue Propriétés de l'onglet Propriétés générales, cliquez sur le bouton ... situé à droite de la zone Emplacement. Passez dans l'interface WSDL où résident le message WSDL et les définitions des types de port puis cliquez sur **OK** pour importer l'interface WSDL dans le fichier WSDL de la connexion/du service.

- g. Enregistrez le fichier WSDL.
- h. Réactualisez/recompilez le projet. Passez dans Business Integration. Ouvrez le diagramme de l'assemblage du module dans Assembly Editor.
- i. Dans la vue de l'explorateur de projet, développez le module que vous migrez puis développez la catégorie logique **Ports de service Web**. Vous devez voir apparaître le port qui existe dans le fichier WSDL de la connexion/du service. Faites-le glisser dans l'éditeur d'assemblage.
- j. Choisissez de créer une **exportation avec liaison de services Web** et sélectionnez le nom de port approprié. Ceci va créer une exportation utilisant l'ancienne liaison/service pour que les clients du service Web actuels ne demandent pas de modification. Si vous sélectionnez l'exportation que vous venez de créer dans Assembly Editor et que vous passez dans la vue Propriétés, vous devez voir dans l'onglet Connexion que les noms de service et le port 5.1 ont été entrés pour vous.
- k. Enregistrez toutes les modifications.
- l. Juste avant de déployer l'application, vous pouvez changer la configuration du projet Web généré pour refléter l'adresse du service version 5.1 (vous devez faire ces modifications chaque fois que vous modifiez le module SCA qui cause la régénération de ce fichier). Si vous consultez la définition de service Web IBM WSDL réutilisée à partir de la version 5.1, vous verrez l'adresse de service définie dans le client 5.1 : `<wsdlsoap:address location="http://localhost:9080/MyServiceWeb/services/MyServicePort"/>`.
- m. Pour que les artefacts du projet Web 6.0 généré utilisent cette ancienne adresse de service, vous devez modifier le descripteur de déploiement du projet Web généré. Ouvrez le descripteur de déploiement dans WebSphere Integration Developer et, dans l'onglet Servlets, ajoutez un autre mappage d'URL proche du mappage d'URL existant pour cette exportation, avec le même nom de servlet mais avec un autre modèle d'URL.
- n. Notez aussi que si vous devez modifier la racine de contexte de ce projet Web pour qu'elle reflète la racine de contexte de l'adresse de service d'origine (dans cet exemple, la racine de contexte est "MyServiceWeb"), vous pouvez alors ouvrir le descripteur de déploiement de l'application J2EE Enterprise où réside ce projet Web et modifier la racine de contexte de ce module Web pour refléter celle de l'ancienne adresse de service. Vous verrez peut-être l'erreur suivante que vous pouvez ignorer : CHKJ3017E: Le projet Web <nom_projet web> est mappé à une racine de contexte non valide : <nouvelle racine de contexte> dans le projet EAR : <nom_application>.

Migration de la liaison de services Web IBM (SOAP/HTTP) - Option 2 :

La seconde option de migration pour la connexion de processus SOAP/HTTP de WebSphere Studio Application Developer Integration Edition consiste à rendre des processus métier accessibles à une entité non-SCA (par exemple, JSP ou un client Java).

La référence autonome permet de rendre un composant SCA accessible à tout type de client externe. Pour créer une référence autonome, procédez comme suit :

1. Ouvrez l'éditeur d'assemblage pour le module créé par l'Assistant de migration.
2. Créez une référence autonome pour chaque interface dotée d'une connexion (SOAP/HTTP) de service générée pour elle dans WebSphere Studio Application Developer Integration Edition :
 - a. Sélectionnez l'élément **Références autonomes** dans la barre d'outils.
 - b. Cliquez sur le canevas de l'éditeur d'assemblage pour créer une entité SCA de références autonomes :
 - c. Sélectionnez l'élément **Connexion** dans la barre d'outils.
 - d. Cliquez sur l'entité **Références autonomes** pour la sélectionner en tant que source de connexion.
 - e. Cliquez sur le composant **SCA** pour le sélectionner en tant que cible de connexion.
 - f. Un message d'alerte s'affiche : **Une référence correspondante va être créée sur le noeud source. Voulez-vous continuer ?**. Cliquez sur **OK**.

- g. Sélectionnez l'entité **Références autonomes** qui vient d'être créée puis, dans la vue des propriétés, sélectionnez le panneau de contenu **Description**.
- h. Développez le lien **Références** et sélectionnez la référence qui vient d'être créée. Le nom et la description de la référence sont répertoriés et peuvent être modifiés le cas échéant.
- i. Si le processus est associé à plusieurs interfaces, sélectionnez celle(s) que vous souhaitez exporter avec ce type de connexion.
- j. Enregistrez le diagramme de l'assemblage.

Migration de la connexion (SOAP/HTTP) de services Web Apache :

La liaison de services Web Apache (SOAP/HTTP) pour un processus BPEL ou un autre type de service peut être migrée vers la construction SCA recommandée.

Dans WebSphere Studio Application Developer Integration Edition, ce type de connexion donne aux clients la capacité de communiquer avec un processus BPEL ou tout autre type de service en appelant un service Web Apache.

Dans WebSphere Studio Application Developer Integration Edition, lorsque la connexion du service Web Apache a été sélectionnée comme type de déploiement pour un processus BPEL ou tout autre type de service, les options suivantes ont été données :

- Pour le style de document, il s'agissait de **RPC** (aucune autre option disponible)
- Pour une action SOAP, il s'agissait de **URN:nom Type de port WSDL**
- Pour l'adresse, il s'agissait de **http://localhost:9080/nom du projet de service Web/servlet/rpcrouter**
- Pour Utiliser le codage, l'option par défaut était **oui** (Si **oui**, alors le style de codage était défini sur : **http://schemas.xmlsoap.org/soap/encoding/**)

Un fichier WSDL spécifiant le service et la connexion Apache SOAP est créé dans le projet de service. Par défaut, ce fichier est créé dans le même répertoire que le service qu'il lie, avec le nom *<nom de processus métier>_<nom du type de port de l'interface de processus métier>_SOAP.wsdl*. Les messages et le type de port WSDL de l'interface du processus métier sont utilisés directement par ce service et cette connexion. Après la migration, vous *ne devriez pas* utiliser ce WSDL sauf éventuellement pour utiliser les mêmes espace de nom, port et noms de service dans le nouveau WSDL qui sera généré dans la version 6.

Il existe deux options pour la migration de la connexion de processus de WebSphere Studio Application Developer Integration Edition Web Service. Un choix devra être fait : migrer le client vers le modèle de programmation SCA ou le laisser en tant que modèle de programmation de services Web IBM. Il n'existe plus de connexion équivalente au type de liaison de services Web Apache (SOAP/HTTP) dans le modèle de programmation SCA V6.

Vous devez migrer ce service Web Apache pour utiliser le moteur IBM Web Service. Pour savoir comment effectuer cette migration et créer un service Web IBM (SOAP/HTTP), voir "Migration de la liaison de services Web IBM (SOAP/HTTP)".

Migration vers le modèle de programmation SCA :

Pour tout code Java de format libre qui interagit avec un service WebSphere Studio Application Developer Integration Edition, cette section montrera comment migrer le modèle de programmation WSIF vers le nouveau modèle de programmation SCA où les données qui transitent dans l'application sont stockées dans des objets SDO Eclipse. Cette section décrit également la migration manuelle des types de client les plus communs vers le nouveau modèle de programmation.

Cette section concerne les processus BPEL qui contiennent des fragments Java. Elle décrit comment migrer l'ancienne API des fragments Java vers la nouvelle API des fragments Java où les données qui transitent par l'application sont stockées dans les objets SDO (Service Data Objects) Eclipse. Lorsque cela

est possible, les fragments sont automatiquement migrés par l'Assistant de migration, mais il existe des fragments que l'Assistant de migration ne peut pas migrer complètement, ce qui signifie que des étapes manuelles sont nécessaires afin de terminer la migration.

Voici un récapitulatif des modifications apportées au modèle de programmation :

Modèle de programmation V5.1

1. Basé sur WSIF et WSDL
2. Proxy générés pour les services
3. Beans et gestionnaires de mise en forme pour les types

Modèle de programmation V6.0 (plus centré sur Java)

1. Services SCA basés sur les SDO avec balises doclet
2. Connexions d'interface pour les services
3. SDO et connexions de données pour les types

Migration des appels des API WSIFMessage vers les API SDO :

La section suivante indique comment effectuer une migration de l'ancien modèle de programmation de WebSphere Business Integration Server Foundation version 5.1 vers le nouveau modèle de programmation de WebSphere Process Server version 6.0. Dans l'ancien modèle de programmation, les données transitant par l'application étaient représentées sous la forme d'objets WSIFMessage avec une interface très spécifique. Dans le nouveau modèle, les données sont représentées sous la forme d'objets SDO (Service Data Objects), sans création d'une interface spécifique.

Tableau 10. Modifications et solutions pour la migration des appels des API WSIFMessage vers les API SDO

Modification	Solution
Les classes de conteneur WSIFMessage ne sont plus générées pour les types de message WSDL et les classes d'aide de bean Java ne sont plus générées pour les types de schéma complexes.	<p>Lorsque vous écrivez du code qui interagit avec les services SCA, vous devez utiliser les API SDO génériques pour manipuler les messages <code>commonj.sdo.DataObject</code> contenant les données qui transitent par l'application.</p> <p>Les définitions de message WSDL avec une seule partie simple seront maintenant représentées par un type Java simple qui représente directement la partie, plutôt qu'un conteneur soit placé autour des données réelles. Si la partie a un type complexe, les données sont représentées comme un <code>DataObject</code> conforme à la définition de type complexe.</p> <p>Les définitions de message WSDL qui possèdent plusieurs parties correspondent maintenant à un <code>DataObject</code> qui a des propriétés pour toutes les parties de message, où les types complexes sont représentés comme des propriétés 'référence-type' du <code>DataObject</code> parent, accessibles via les méthodes <code>getDataObject</code> et <code>setDataObject</code>.</p>
Vous ne devez plus utiliser les méthodes getter spécifiques pour les parties WSIFMessage et les beans Java générés.	Vous devez utiliser l'API SDO non spécifique pour obtenir les propriétés <code>DataObject</code> .
Les méthodes setter spécifiques pour les messages des variables BPEL ne sont plus disponibles.	L'API SDO non spécifique doit être utilisée pour définir les propriétés <code>DataObject</code> .
Vous ne devez plus utiliser les méthodes getter non spécifiques pour les propriétés WSIFMessage.	Vous devez utiliser l'API SDO non spécifique pour définir les propriétés <code>DataObject</code> .

Tableau 10. Modifications et solutions pour la migration des appels des API WSIFMessage vers les API SDO (suite)

Modification	Solution
Vous ne devez plus utiliser les méthodes setter non spécifiques pour les propriétés WSIFMessage.	Vous devez utiliser l'API SDO non spécifique pour définir les propriétés DataObject.
Le cas échéant, tous les appels API WSIFMessage doivent être migrés vers l'API SDO.	Dans la mesure du possible, migrez l'appel vers un appel API SDO équivalent. Sinon, concevez de nouveau la logique.

Migration du code client de WebSphere Business Integration Server Foundation :

Cette section décrit comment migrer les différents types de client disponibles pour les types de service WebSphere Business Integration Server Foundation 5.1.

Migration du client EJB :

Cette rubrique indique comment effectuer la migration de clients utilisant une interface EJB pour appeler un service.

1. Faites un glisser/déposer de l'exportation avec liaison SCA depuis le module migré vers l'instance d'Assembly Editor du nouveau module. Cela va créer une importation avec liaison SCA. Pour qu'un client obtienne une référence à cette importation, vous devez créer une référence autonome.
2. Sélectionnez l'option Référence autonome dans la palette. Cliquez sur le canevas Assembly Editor une fois pour créer une nouvelle référence autonome pour ce nouveau module.
3. Sélectionnez l'outil Connexion et cliquez sur la référence du service, puis sur **Importer**.
4. Cliquez sur **OK** quand un message vous dit qu'une référence correspondante sera créé sur le noeud source.
5. Vous recevez le message suivant : **Il est plus facile pour un client Java d'utiliser une interface Java avec cette référence. Voulez-vous convertir la référence WSDL en référence compatible Java ?**
 - a. Cliquez sur **Oui** si vous souhaitez que le client recherche ce service et effectue un cast vers une classe Java pour l'appeler à l'aide d'une interface Java. Cette nouvelle interface Java prend le nom du type de port WSDL. Le module de l'interface provient de l'espace de nom du type de port WSDL. Une méthode est définie pour chaque opération définie sur le type de port WSDL et chaque message WSDL est représenté par un argument dans les méthodes de l'interface.
 - b. Cliquez sur **Non** si vous souhaitez que le client recherche ce service et utilise l'interface générique `com.ibm.websphere.sca.Service` pour l'appeler à l'aide de l'opération d'appel en tant que service SCA générique.
6. Si besoin, donnez à la référence autonome un nom plus significatif en sélectionnant l'option Références autonomes dans Assembly Editor. Passez dans la vue Propriétés, onglet Détails, puis recherchez et sélectionnez la référence qui vient d'être créée. Modifiez son nom. Gardez en mémoire le nom choisi pour cette référence car le client devra utiliser celui-ci lors de l'appel de la méthode `locateService` de l'instance `com.ibm.websphere.sca.ServiceManager`.
7. Cliquez sur **Sauvegarder** pour sauvegarder le diagramme de l'assemblage.

Le nouveau module doit se trouver dans le chemin de classe local du client pour que l'on puisse accéder au module EJB migré qui s'exécute sur le serveur.

Voici à quoi ressemble le code client pour un service de type "CustomerInfo" :

```
// Créer un gestionnaire de services
ServiceManager serviceManager = ServiceManager.INSTANCE;

// Localiser le service CustomerInfo
CustomerInfo customerInfoService = (CustomerInfo) serviceManager.locateService(
"<name-of-standalone-reference-from-previous-step>");
```

```
// Appeler le service CustomerInfo
System.out.println(" [getMyValue] getting customer info...");
DataObject customer = customerInfoService.getCustomerInfo(customerID);
```

Le client doit modifier la structure du message. Auparavant, les messages étaient basés sur la classe WSIFMessage mais ils doivent maintenant être basés sur la classe `commonj.sdo.DataObject`.

Migration du client de connexion de processus EJB :

Cette rubrique indique comment effectuer la migration des clients utilisant la liaison de processus EJB WSIF pour accéder à un service BPEL.

Les clients qui utilisaient la connexion de processus EJB pour appeler un processus métier doivent maintenant utiliser l'API SCA pour appeler le service (le processus métier migré doit avoir une exportation avec liaison SCA) ou l'API du client IBM Web Service pour appeler le service (le processus métier migré doit avoir une exportation avec liaison de services Web).

Pour plus d'informations sur la génération de ce type de client, voir "Migration du client EJB", "Migration du client IBM Web Service (SOAP/JMS)" ou "Migration du client IBM Web Service (SOAP/HTTP)".

Migration du client IBM Web Service (SOAP/JMS) :

Cette rubrique indique comment effectuer la migration de clients utilisant des API de services Web (SOAP/JMS) pour appeler un service.

Aucune migration n'est nécessaire pour les clients existants pendant la migration. Notez que vous devez modifier manuellement le projet Web généré (créez un mappage de servlet) et parfois modifier la racine de contexte du projet Web dans le descripteur de déploiement de l'application d'entreprise pour publier le service à la même adresse que celle à laquelle il avait été publié sur WebSphere Business Integration Server Foundation. Voir "Migration de la liaison de services Web IBM (SOAP/JMS)".

Notez que contrairement à la version 5.1, dans laquelle un proxy client WSIF ou RPC pouvait être généré, dans la version 6.0, les outils prennent uniquement en charge la génération de client RPC car RPC est l'API préférée de la version 6.0 par rapport aux API WSIF.

Remarque : Pour générer un nouveau proxy client à partir de WebSphere Integration Developer, un serveur WebSphere Process Server ou WebSphere Application Server doit être installé sur votre système.

1. Vérifiez qu'un serveur WebSphere Process Server ou WebSphere Application Server est installé.
2. Dans la perspective Resfacts source ou Java, recherchez le fichier WSDL correspondant à l'**exportation avec liaison de services Web**, puis cliquez avec le bouton droit de la souris et sélectionnez **Services Web → Générer client** (notez que cet assistant ressemble beaucoup à l'assistant de la version 5.1).
3. Sélectionnez **Proxy Java** comme type du proxy client et cliquez sur **Suivant**.
4. L'emplacement de WSDL doit être indiqué. Cliquez sur **Suivant**.
5. Vous devez ensuite sélectionner les options appropriées pour indiquer la configuration de l'environnement client, notamment le serveur et l'exécution du service Web, la version de J2EE, le type de client (Java, EJB, Web, Client d'application). Cliquez sur **Suivant**.
6. Terminez les étapes restantes pour créer le proxy client.

Migration du client IBM Web Service (SOAP/HTTP) :

Cette rubrique indique comment effectuer la migration de clients utilisant des API de services Web (SOAP/HTTP) pour appeler un service.

Aucune migration n'est nécessaire pour les clients existants pendant la migration. Notez que vous devez modifier manuellement le projet Web généré (créez un mappage de servlet) et parfois modifier la racine

de contexte du projet Web dans le descripteur de déploiement de l'application d'entreprise pour publier le service à la même adresse que celle à laquelle il avait été publié sur WebSphere Business Integration Server Foundation. Voir "Migration de la liaison de services Web IBM (SOAP/HTTP)".

Si des modifications de conception ont été effectuées et que vous souhaitez créer un nouveau proxy client, suivez les étapes ci-dessous. Notez que contrairement à la version 5.1, dans laquelle un proxy client WSIF ou RPC pouvait être généré, dans la version 6.0, les outils prennent uniquement en charge la génération de client RPC car RPC est l'API préférée de la version 6.0 par rapport aux API WSIF.

Remarque : Pour générer un nouveau proxy client à partir de WebSphere Integration Developer, un serveur WebSphere Process Server ou WebSphere Application Server doit être installé sur votre système.

1. Vérifiez qu'un serveur WebSphere Process Server ou WebSphere Application Server est installé.
2. Sélectionnez le fichier WSDL correspondant à l'**exportation avec liaison de services Web**, puis cliquez avec le bouton droit de la souris et sélectionnez **Services Web** → **Générer client** (notez que cet assistant ressemble beaucoup à l'assistant de la version 5.1).
3. Pour le type du proxy client, sélectionnez **Proxy Java** et cliquez sur **Suivant**.
4. L'emplacement de WSDL doit être indiqué. Cliquez sur **Suivant**.
5. Vous devez ensuite sélectionner les options appropriées pour indiquer la configuration de l'environnement client, notamment le serveur et l'exécution du service Web, la version de J2EE, le type de client (Java, EJB, Web, Client d'application). Cliquez sur **Suivant**.
6. Terminez les étapes restantes pour créer le proxy client.

Migration du client Apache Web Service (SOAP/HTTP) :

Les API du client Apache Web Service ne sont pas appropriées pour appeler un service WebSphere Integration Developer. Le code client doit être migré pour que les API du client IBM Web Service (SOAP/HTTP) soient utilisables.

Pour plus d'informations, voir la section "Client (SOAP/HTTP) de services Web IBM".

Dans la version 5.1, si un proxy client était automatiquement généré, ce proxy utilisait les API WSIF pour interagir avec le service. Dans la version 6.0, les outils prennent uniquement en charge la génération de client RPC car RPC est l'API préférée de la version 6.0 par rapport aux API WSIF.

Remarque : Pour générer un nouveau proxy client à partir de WebSphere Integration Developer, un serveur WebSphere Process Server ou WebSphere Application Server doit être installé sur votre système.

1. Vérifiez qu'un serveur WebSphere Process Server ou WebSphere Application Server est installé.
2. Sélectionnez le fichier WSDL correspondant à l'**exportation avec liaison de services Web**, puis cliquez avec le bouton droit de la souris et sélectionnez **Services Web** → **Générer client** (notez que cet assistant ressemble beaucoup à l'assistant de la version 5.1).
3. Sélectionnez **Proxy Java** comme type du proxy client et cliquez sur **Suivant**.
4. L'emplacement de WSDL doit être indiqué. Cliquez sur **Suivant**.
5. Vous devez ensuite sélectionner les options appropriées pour indiquer la configuration de l'environnement client, notamment le serveur et l'exécution du service Web, la version de J2EE, le type de client (Java, EJB, Web, Client d'application). Cliquez sur **Suivant**.
6. Terminez les étapes restantes pour créer le proxy client.

Migration du client JMS :

Il est possible que vous deviez procéder à une migration manuelle des clients qui communiquaient précédemment avec un service 5.1 via l'API JMS (envoi d'un message JMS dans une file d'attente). Cette rubrique indique comment effectuer la migration de clients utilisant des API JMS pour appeler un service.

Vous devez vous assurer que l'**exportation avec liaison JMS** que vous avez créée lors d'une étape précédente pourra accepter le message de texte ou d'objet sans le modifier. Pour cela, vous serez peut-être amené à écrire une liaison de données personnalisée. Voir la section "Migration de la connexion JMS et de la connexion de processus JMS" pour plus d'informations.

Le client doit modifier la structure du message. Auparavant, les messages étaient basés sur la classe WSIFMessage mais ils doivent maintenant être basés sur la classe `commonj.sdo.DataObject`. Pour plus d'informations sur cette migration manuelle, voir la section "Migration des appels des API WSIFMessage vers les API SDO".

Migration d'un client utilisant l'API EJB générique de Business Process Choreographer :

Cette rubrique traite de la migration des clients utilisant l'API EJB générique de Process Choreographer 5.1 pour appeler un service BPEL.

La nouvelle version de l'API EJB générique utilise le format de message DataObjects. Le client doit modifier la structure du message. Auparavant, les messages étaient basés sur la classe WSIFMessage mais ils doivent maintenant être basés sur la classe `commonj.sdo.DataObject`. Notez que l'API EJB générique n'a pas beaucoup changé puisque `ClientObjectWrapper` fournit toujours un conteneur de message associé au format du message.

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

Le nom JNDI de l'ancienne API EJB générique qui prend les objets WSIFMessage est le suivant :

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

Il existe deux API EJB génériques dans la version 6.0 car les opérations de tâche manuelles sont maintenant disponibles dans un EJB séparé. Les noms JNDI V6.0 de ces API EJB génériques sont les suivants :

```
GenericBusinessFlowManagerEJB
Nom JNDI : com/ibm/bpe/api/BusinessFlowManagerHome
Interface : com.ibm.bpe.api.BusinessFlowManager
```

```
HumanTaskManagerEJB
Nom JNDI : com/ibm/task/api/TaskManagerHome
Interface : com.ibm.task.api.TaskManager
```

Migration du client API de messagerie générique de Business Process Choreographer et du client de connexion de processus JMS :

Il n'existe pas d'API de messagerie générique dans WebSphere Process Server 6.0. Voir "Migration de la connexion JMS et de la connexion de processus JMS" pour choisir une autre manière d'exposer le processus métier aux consommateurs et réécrire le client en fonction de la liaison sélectionnée.

Migration du client Web de Business Process Choreographer :

Cette rubrique indique comment effectuer la migration des paramètres des clients Web et des JSP personnalisés de Process Choreographer 5.1.

Les paramètres du client Web 5.1 ne peuvent pas être modifiés à l'aide de l'Assistant de migration, ni via l'Éditeur de tâche manuelle. Pour créer de nouveaux paramètres de client Web et de nouveaux JSP, utilisez WebSphere Integration Developer 6.0.

Migration des modifications du client Web

Dans la version 5.1, vous pouviez modifier l'apparence du client Web Struts en modifiant son JSP `Header.jsp` et sa feuille de style `dwc.css`.

Depuis la version 6.0, le client Web (renommé **BPC Explorer**) est basé sur Java Server Faces (JSF) plutôt que sur Struts ; la migration automatique des modifications du client Web n'est pas possible. Par conséquent, nous vous recommandons de consulter la documentation de l'explorateur BPC pour plus d'informations sur la personnalisation de la version 6.0 de cette application.

Les JSP définis par l'utilisateur peuvent être définis pour les processus métier ainsi que pour les actions de personnel. Le client Web utilise ces JSP afin d'afficher les messages d'entrée et de sortie pour le processus et les activités.

Ces JSP sont particulièrement utiles dans les cas suivants :

1. Les messages ont des parties non-primitives pour étendre l'utilisation de la structure de données du message.
2. Vous souhaitez étendre les fonctionnalités du client Web.

Les options disponibles sont différentes et plus nombreuses lorsque vous indiquez les paramètres du client Web pour un processus 6.0. Par conséquent, vous devrez utiliser WebSphere Integration Developer afin de concevoir de nouveau les paramètres du client Web pour les activités et les processus migrés :

1. Sélectionnez le canevas de processus ou une activité dans le processus.
2. Dans la vue Propriétés, sélectionnez l'onglet **Client** pour concevoir de nouveau les paramètres du client Web.
3. Migrez manuellement tous les JSP définis par l'utilisateur :
 - a. Voir la section "Migration du modèle de programmation SCA" concernant la modification du modèle de programmation.
 - b. Le client Web utilise les API génériques pour interagir avec les processus métier. Reportez-vous aux sections décrivant la migration d'appels vers ces API.
4. Indiquez le nom du nouveau JSP dans les paramètres du client Web 6.0 pour le processus.

Remarque : Les JSP de mappage ne sont pas nécessaires avec l'explorateur BPC 6.0 dans la mesure où DataObjects ne requiert aucun mappage personnalisé.

Migration des fragments Java de WebSphere Business Integration Server Foundation BPEL :

Cette section concerne les processus BPEL qui contiennent des fragments Java. Elle décrit comment migrer l'ancienne API des fragments Java vers la nouvelle API des fragments Java quand les données qui transitent par l'application sont stockées comme objets SDO (Service Data Objects) Eclipse.

Pour connaître les étapes de migration spécifiques à la transition de WSIFMessage vers SDO, voir la section "Migration des appels des API WSIFMessage vers les API SDO".

Si possible, les fragments sont migrés automatiquement par l'Assistant de migration mais celui-ci ne peut pas les migrer tous. Des étapes manuelles supplémentaires sont nécessaires pour compléter la migration. Pour plus d'informations sur les types de fragments Java devant être migrés manuellement, voir la rubrique sur les restrictions. Chaque fois que l'un de ces fragments est détecté, l'Assistant de migration explique pourquoi il ne peut pas être migré automatiquement et envoie un avertissement ou un message d'erreur.

Le tableau suivant illustre les modifications apportées à l'API et au modèle de programmation de fragments BPEL Java entre la version 5.1 et la version 6.0 de Business Process Choreographer :

Tableau 11. Modifications et solutions concernant la migration des fragments Java de WebSphere Business Integration Server Foundation BPEL

Modification	Solution
Les classes de conteneur WSIFMessage ne sont plus générées pour les types de message WSDL et les classes d'aide de bean Java ne sont plus générées pour les types de schéma complexe.	<p>Les variables BPEL sont directement accessibles par nom. Notez que les variables BPEL dont la définition de message WSDL possède une seule partie représenteront maintenant directement la partie concernée au lieu d'encapsuler les données. Les variables dont le type de message possède plusieurs parties auront un conteneur DataObject autour des parties (le conteneur dans WebSphere Application Developer Integration Edition était un WSIFMessage).</p> <p>Etant donné que les variables BPEL peuvent être utilisées directement dans les fragments 6.0, les variables locales ne sont pas aussi nécessaires que dans la version 5.1.</p> <p>Les méthodes getter spécifiques pour les variables BPEL initialisaient de manière implicite l'objet de conteneur WSIFMessage autour des parties du message. Il n'existe pas d'objet 'conteneur' pour les variables BPEL dont la définition de message WSDL n'a qu'une seule partie : dans ce cas, les variables BPEL représentent directement la partie (lorsque la partie est un type simple XSD, la variable BPEL est représentée en tant que type de conteneur d'objet Java tel que java.lang.String, java.lang.Integer, etc). Les variables BPEL avec des définitions de message WSDL contenant plusieurs parties sont traitées différemment : il existe encore un conteneur autour des parties et ce conteneur DataObject doit être initialisé de manière explicite dans le code de fragment Java 6.0 s'il n'a pas déjà été défini par une opération précédente.</p> <p>Si des variables locales des fragments 5.1 portaient le même nom que la variable BPEL, des conflits peuvent survenir. Dans la mesure du possible, essayez de résoudre cette situation.</p>
Les objets WSIFMessage ne sont plus utilisés pour représenter les variables BPEL.	Si l'une des classes Java personnalisées appelée à partir des fragments Java possède un paramètre WSIFMessage, elle devra être migrée de manière à accepter/renvoyer un DataObject.
Les méthodes getter spécifiques pour les variables BPEL ne sont plus disponibles.	Les variables sont directement accessibles par nom. Notez que les variables BPEL dont la définition de message WSDL possède une seule partie représenteront maintenant la partie directement au lieu d'avoir un conteneur autour des données réelles. Les variables dont le type de message possède plusieurs parties auront un conteneur DataObject autour des parties (le conteneur dans WebSphere Application Developer Integration Edition était un WSIFMessage).

Tableau 11. Modifications et solutions concernant la migration des fragments Java de WebSphere Business Integration Server Foundation BPEL (suite)

Modification	Solution
Les méthodes setter spécifiques pour les variables BPEL ne sont plus disponibles.	Les variables sont directement accessibles par nom. Notez que les variables BPEL dont la définition de message WSDL possède une seule partie représenteront maintenant directement la partie concernée au lieu d'encapsuler les données. Les variables dont le type de message possède plusieurs parties auront un conteneur DataObject autour des parties (le conteneur dans WebSphere Application Developer Integration Edition était un WSIFMessage).
Les méthodes getter non spécifiques pour les variables BPEL qui renvoient un WSIFMessage ne sont plus disponibles.	<p>Les variables sont directement accessibles par nom. Notez que les variables BPEL dont la définition de message WSDL possède une seule partie représenteront maintenant directement la partie concernée au lieu d'encapsuler les données. Les variables dont le type de message possède plusieurs parties auront un conteneur DataObject autour des parties (le conteneur dans WebSphere Application Developer Integration Edition était un WSIFMessage).</p> <p>Notez qu'il existait deux variations de la méthode <code>getVariableAsWSIFMessage</code> :</p> <pre>getVariableAsWSIFMessage(String variableName) getVariableAsWSIFMessage(String variableName, boolean forUpdate)</pre> <p>Pour une activité de fragment de code Java, l'accès par défaut est défini en lecture-écriture. Vous pouvez activer l'accès en lecture seule en indiquant @bpe.readOnlyVariables dans un commentaire du fragment de code, avec la liste des noms de variables concernées. Par exemple, vous pouvez mettre les variables B et D en lecture seule en procédant comme suit :</p> <pre>variableB.setString ("/x/y/z", variableA.getString("/a/b/c")); // @bpe.readOnlyVariables names="variableA" variableD.setInt("/x/y/z", variableC.getInt ("/a/b/c")); // @bpe.readOnlyVariables names="variableC"</pre> <p>Dans le cas d'un fragment de code Java inclus dans une condition, les variables sont en lecture seule par défaut. Vous pouvez les rendre accessibles en en lecture-écriture en indiquant @bpe.readWriteVariables...</p>
Les méthodes setter non spécifiques pour les variables BPEL ne sont plus disponibles.	Les variables sont directement accessibles par nom. Notez que les variables BPEL dont la définition de message WSDL possède une seule partie représenteront maintenant directement la partie concernée au lieu d'encapsuler les données. Les variables dont le type de message possède plusieurs parties auront un conteneur DataObject autour des parties (le conteneur dans WebSphere Application Developer Integration Edition était un WSIFMessage).

Tableau 11. Modifications et solutions concernant la migration des fragments Java de WebSphere Business Integration Server Foundation BPEL (suite)

Modification	Solution
Les méthodes getter non spécifiques pour les messages des variables BPEL ne sont pas appropriées pour les messages à une seule partie et ont été modifiées pour les messages à plusieurs parties.	<p>Migrez vers la méthode getter non spécifique pour les propriétés (DataObject) des variables BPEL.</p> <p>Notez que pour les variables BPEL dont la définition de message WSDL possède une seule partie, la variable BPEL représente directement la partie et vous devez accéder à la variable directement, sans l'aide d'une méthode getter.</p> <p>Il existait deux variations de la méthode <code>getVariablePartAsObject</code> :</p> <pre>getVariablePartAsObject(String variableName, String partName) getVariablePartAsObject(String variableName, String partName,boolean forUpdate)</pre> <p>Pour les messages à plusieurs parties, une fonctionnalité équivalente est fournie par cette méthode dans la version 6.0 :</p> <pre>getVariableProperty(String variableName, QName propertyName);</pre> <p>Dans la version 6.0, il n'est pas question de l'utilisation d'une variable pour l'accès en lecture seule (alors que c'était le cas dans la version 5.1 pour la première méthode ainsi que la deuxième méthode avec <code>forUpdate=false</code>). La variable est directement utilisée dans le fragment 6.0 et peut toujours être mise à jour.</p>
Les méthodes setter non spécifiques pour les messages des variables BPEL ne sont pas appropriées pour les messages à une seule partie et ont été modifiées pour les messages à plusieurs parties.	<p>Migrez vers la méthode setter non spécifique pour les propriétés (DataObject) des variables BPEL.</p> <p>Notez que pour les variables BPEL dont la définition de message WSDL possède une seule partie, la variable BPEL représente directement la partie et vous devez accéder à la variable directement, sans l'aide d'une méthode setter.</p> <p>Les appels à la méthode suivante doivent être migrés :</p> <pre>setVariableObjectPart(String variableName, String partName, Object data)</pre> <p>Pour les messages à plusieurs parties, une fonctionnalité équivalente est fournie par cette méthode dans la version 6.0 :</p> <pre>setVariableProperty(String variableName, QName propertyName,Serializable value);</pre>
Les méthodes getter spécifiques pour les liens partenaires BPEL ne sont plus disponibles.	Migrez vers les méthodes getter non spécifiques pour les liens partenaires BPEL.
Les méthodes setter spécifiques pour les liens partenaires BPEL ne sont plus disponibles.	Migrez vers les méthodes setter non spécifiques pour les liens partenaires BPEL.

Tableau 11. Modifications et solutions concernant la migration des fragments Java de WebSphere Business Integration Server Foundation BPEL (suite)

Modification	Solution
Les méthodes getter spécifiques pour les ensembles de corrélation BPEL ne sont plus disponibles.	<p>Fragment V5.1 :</p> <pre>String corrSetPropStr = getCorrelationSetCorrSet- APropertyCustomerName(); int corrSetPropInt = getCorrelationSet- CorrSetBPropertyCustomerId();</pre> <p>Fragment V6.0 :</p> <pre>String corrSetPropStr = (String) getCorrelationSetProperty ("CorrSetA", new QName("CustomerName")); int corrSetPropInt = (Integer) getCorrelationSetProperty ("CorrSetB", new QName("CustomerId"))).intValue();</pre>
Paramètre supplémentaire requis pour les méthodes getter non spécifiques pour les propriétés personnalisées d'activité BPEL.	<p>Fragment V5.1 :</p> <pre>String val = getActivityCustomProperty("propName");</pre> <p>Fragment V6.0 :</p> <pre>String val = getActivityCustomProperty ("name-of-current-activity", "propName");</pre>
Paramètre supplémentaire requis pour les méthodes setter non spécifiques pour les propriétés personnalisées d'activité BPEL.	<p>Fragment V5.1 :</p> <pre>String newVal = "new value"; setActivityCustomProperty ("propName", newVal);</pre> <p>Fragment V6.0 :</p> <pre>String newVal = "new value"; setActivityCustomProperty ("name-of-current-activity", "propName", newVal);</pre>
La méthode raiseFault(QName faultQName, Serializable message) n'existe plus.	Le cas échéant, migrez vers la méthode raiseFault(QName faultQName, String variableName) ; sinon, migrez vers la méthode raiseFault(QName faultQName) ou créez une nouvelle variable BPEL pour l'objet Serializable.

Migration des interactions avec les adaptateurs WebSphere Business Integration :

Si le client JMS est un adaptateur WebSphere Business Integration, vous pouvez avoir besoin d'utiliser les outils de Reconnaissance des services d'entreprise pour créer l'importation avec liaison JMS. Cette importation utilise une liaison de données spéciale pour sérialiser le SDO au format attendu par l'adaptateur WebSphere Business Integration.

Pour accéder aux outils de Reconnaissance des services d'entreprise :

1. Sélectionnez **Fichier** → **Nouveau** → **Autre** → **Business Integration** → **Reconnaissance des services d'entreprise**. Cliquez sur **Suivant**.
2. Sélectionnez **Module d'importation d'artefacts de l'adaptateur WBI**. Cliquez sur **Suivant**.
3. Entrez le chemin du fichier de configuration (.cfg) de l'adaptateur WebSphere Business Integration et le répertoire qui contient le schéma XML des objets métier utilisés par l'adaptateur. Cliquez sur **Suivant**.
4. Examinez la requête générée et, si elle est correcte, cliquez sur **Exécuter la requête**. Dans la liste des **Objets découverts par la requête**, sélectionnez les objets que vous voulez ajouter (un par un) et cliquez sur le bouton >> **Ajouter**.
5. Acceptez les paramètres de configuration pour l'objet métier et cliquez sur **OK**.
6. Répétez cette procédure pour chaque objet métier.
7. Cliquez sur **Suivant**.
8. Pour le **format des objets métier à l'exécution**, sélectionnez **SDO**. Pour le **projet cible**, sélectionnez le module que vous venez de migrer. Laissez vide le champ **Dossier**.
9. Cliquez sur **Terminer**.

Cet outil va migrer les anciens XSD au format attendu par la liaison de données spéciale. Vous devez donc supprimer du module les anciens XSD de l'adaptateur WebSphere Business Integration et utiliser les nouveaux XSD. Si le module n'est pas supposé recevoir les messages de l'adaptateur, supprimez les exports générés par cet outil. Si le module n'est pas supposé envoyer des messages à l'adaptateur, supprimez l'importation. Voir le centre de documentation pour plus d'informations sur cette fonction.

Migration d'interfaces WSDL avec des types de tableau encodé soap :

Cette section décrit comment migrer ou gérer des schémas XML avec des types de tableau encodé soap.

Les types de tableau encodé soap et dotés du style RPC sont traités par la version 6.0 comme des séquences illimitées d'un type concret. Il n'est pas recommandé de créer des types XSD qui référencent les types soapenc:Array, car le modèle de programmation se déplace vers le style encapsulé Document/Literal plutôt que vers le style RPC.

Dans certains cas, une application SCA doit appeler un service externe qui utilise le type soapenc:Array. Il arrive que cette situation ne puisse pas être évitée. Voici comment vous pouvez la gérer :

Exemple de code WSDL :

```
<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>

<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>

<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>

```

Exemple de code pour un client de ce service Web :

```

// Trouver le service du
fournisseur et rechercher l'opération doFindVendor

Service findVendor=(Service)ServiceManager.INSTANCE.locateService("vendorSearch");
OperationType doFindVendorOperationType=findVendor.getReference().
getOperationType("doGoogleSearch");

// Créer l'objet DataObject utilisé en entrée
DataObject doFindVendor=DataFactory.INSTANCE.create(doFindVendorOperationType.getInputType());
doFindVendor.setString("productId", "12345");
doFindVendor.setString("productName", "Refrigerator");

// Appeler le service FindVendor
DataObject findVendorResult = (DataObject)findVendor.invoke(doFindVendorOperationType, doFindVendor);

// Afficher les résultats
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

```

Voici un autre exemple dans lequel le type racine de l'objet de données est soapenc:Array. Remarquez la façon dont l'objet de données sampleElements est créé à l'aide du second schéma listé ci-dessus. Le type de l'objet de données est obtenu en premier, avant la propriété de sampleStructElement. Il s'agit véritablement d'une propriété de signet qui est uniquement utilisée pour obtenir une propriété valide permettant d'ajouter les DataObjects à la séquence. Le modèle suivant peut être utilisé dans votre scénario :

Exemple de code WSDL :

```

<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>

```

Exemple de code pour un client de ce service Web :

```

// Créer l'objet DataObject utilisé en entrée
// et obtenir la séquence SDO pour l'élément any
DataFactory dataFactory=DataFactory.INSTANCE;
DataObject arrayOfStruct =
dataFactory.create("http://soapinterop.org/xsd","ArrayOfSOAPStruct");
Sequence sequence=arrayOfStruct.getSequence("any");

// Obtenir la propriété SDO pour l'élément exemple à utiliser
// dans ce cas pour renseigner la séquence.
// Cet élément a été défini dans le fichier SampleElements.xsd.
DataObject
sampleElements=dataFactory.create("http://sample/elements",
"DocumentRoot");
Property property =
sampleElements.getType().getProperty("sampleStructElement");

    // Ajouter les éléments à la séquence
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);

// Appeler l'opération echoStructArray
System.out.println("[client] invoking echoStructArray operation");
DataObject echoArrayOfStruct = (DataObject)interopTest.invoke("echoStructArray", arrayOfStruct);

```

```
// Afficher les résultats
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 projets EJB WebSphere Business Integration :

Dans WebSphere Studio Application Developer Integration Edition, les projets EJB peuvent avoir des fonctionnalités WebSphere Business Integration spécifiques, telles que la Messagerie étendue (CMM) et CMP/A (Component-Managed Persistence Anywhere). Les descripteurs de déploiement pour ce type de projet doivent être migrés ; cette section explique comment effectuer cette migration.

Pour effectuer cette migration, suivez les étapes ci-dessous :

1. Copiez le projet EJB WebSphere Business Integration vers le nouvel espace de travail 6.0 et importez-le à partir de WebSphere Integration Developer à l'aide de l'option **Fichier → Importer → Projet existant dans l'espace de travail** de l'assistant. En outre, vous pouvez éventuellement exécuter l'Assistant de migration J2EE.
2. Fermez toutes les instances de WebSphere Integration Developer en cours d'exécution dans l'espace de travail V6.0.
3. Exécutez le script suivant pour migrer les descripteurs de déploiement WebSphere Business Integration dans le projet EJB :

Sous Windows :

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

Sous Linux :

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

Les paramètres suivants sont pris en charge (l'espace de travail et le nom du projet sont obligatoires) :

Syntaxe : WSADIEJBProjectMigration.bat
 [-e dossier-eclipse] -d espace-travail -p projet

dossier-eclipse : Emplacement de votre dossier Eclipse -- il s'agit généralement de l'élément 'eclipse' situé sous le dossier d'installation du produit.

espace-travail : Espace de travail contenant le projet WSADIE EJB à migrer.

projet : Nom du projet à migrer.

Par exemple,

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

4. Lorsque vous ouvrez, WebSphere Integration Developer, vous devez actualiser le projet EJB afin d'obtenir les fichiers mis à jour.
5. Recherchez le fichier **ibm-web-ext.xmi** dans le projet EJB. Si vous le trouvez, vérifiez que la ligne suivante est présente dans ce fichier sous l'élément :

```
<webappext:WebAppExtension> element:
<webApp href="WEB-INF/web.xml#WebApp"/>
```
6. Supprimez l'ancien code de déploiement généré dans la version 5.1. Recréez le code de déploiement en suivant les instructions de WebSphere Application Server.

Correction manuelle des conflits d'espaces de nom :

Sous WebSphere Studio Application Developer Integration Edition 5.1, il était possible de définir deux types WSDL ou XSD différents dotés du même nom et du même espace de nom cible. WebSphere Integration Developer 6.0 n'assure plus cette prise en charge. Si après avoir créé les projets migrés, vous constatez que des définitions figurent en double, vous devez effectuer une migration manuelle.

Procédez comme suit :

1. Si les définitions sont identiques, supprimez-en une puis nettoyez et recréez le projet. Corrigez les erreurs éventuelles en modifiant les fichiers WSDL/XSD existants pour qu'ils pointent vers le fichier contenant la définition que vous *n'avez pas* supprimée.
2. Si les définitions sont *différentes* et doivent toutes deux être utilisées dans le service migré, modifiez le nom de la définition ou celui de son espace de nom cible. Si seules quelques définitions du fichier sont des doublons, il est préférable de les renommer. Si toutes les définitions du fichier sont des doublons, il est recommandé de modifier leur espace de nom cible. Nettoyez et recréez le projet en vous assurant que les artefacts à utiliser dans la ou les définition(s) modifiée(s) font bien référence au nouveau nom ou espace de nom.
3. Si un fichier WSDL contient deux instructions d'importation pour un même espace de nom, vous pouvez définir une chaîne de façon que chacune de ces instructions importe la suivante. Ainsi, il n'existera au final qu'une seule importation par fichier WSDL pour un espace de nom donné. Ensuite, nettoyez et recréez le projet.

Suppression manuelle des définitions WSIF (Web Services Invocation Framework) :

Après avoir terminé la migration des artefacts source, supprimez toutes les définitions WSDL de liaisons et de services WSIF 5.1 qui ont été configurées pour des projets 6.0 mais ne sont plus utilisées. Le scénario de consommation pour la migration de services est le seul cas dans lequel une liaison ou un service WSIF peut encore être utilisé.

Les espaces de nom WSDL ci-après indiquent qu'une définition de liaison ou de service est un service WSIF 5.1 pouvant être supprimé s'il n'est plus utilisé :

Espace de nom WSIF JMS :

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

Espace de nom WSIF Java :

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

Espace de nom WSIF JMS :

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

Espace de nom WSIF d'un processus métier :

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

Espace de nom WSIF du transformateur :

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

Espace de nom WSIF IMS :

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

Espace de nom WSIF CICS-ECI :

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

Espace de nom WSIF CICS-EPI :

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

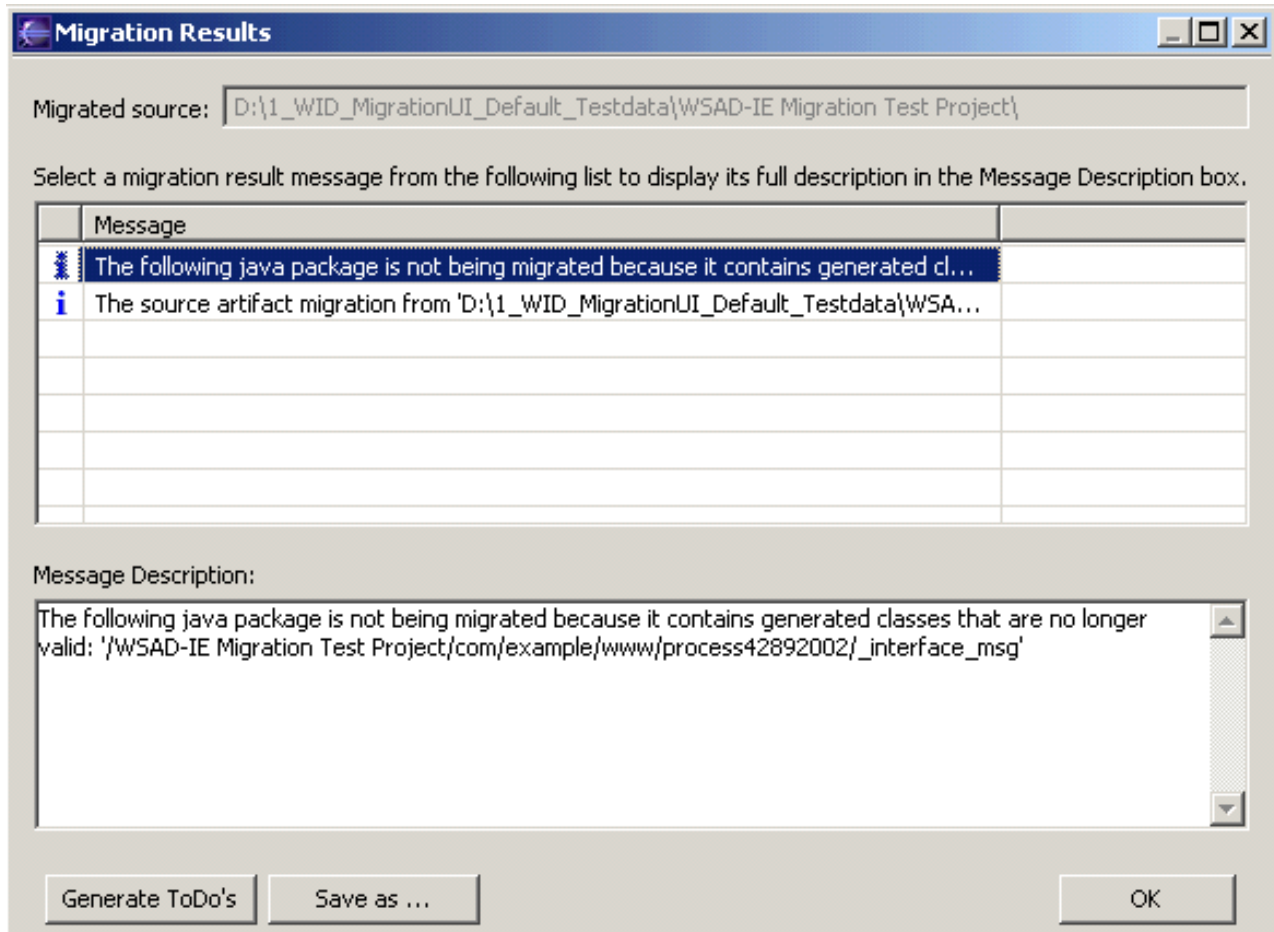
Espace de nom WSIF HOD :

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

Vérification de la migration de l'artefact source

Lorsque l'exécution de l'assistant d'installation est terminée, une liste répertoriant les erreurs, les avertissements et/ou les messages informatifs s'affiche. Vous pouvez utiliser ces messages pour vérifier la migration de l'artefact source.

La page suivante apparaît lorsque l'exécution de l'Assistant de migration est terminée :



Examinez chaque message pour déterminer si une action doit être mise en oeuvre pour corriger immédiatement un artefact dont la migration n'a pas été entièrement effectuée.

Pour vérifier qu'une partie de la migration a été effectuée, passez sur la perspective Business Integration et vérifiez que tous les processus et interfaces WSDL de l'ancien projet de service apparaît dans le nouveau module. Créez le projet et corrigez toutes les erreurs empêchant sa création.

Une fois terminées les étapes manuelles de la migration de l'application Business Integration, exportez l'application en tant que fichier EAR et installez-la sur un serveur WebSphere Process Server en configurant les ressources appropriées.

Effectuez les étapes de migration manuelles requises pour migrer un code client ou pour générer un nouveau code client à l'aide de WebSphere Integration Developer. Assurez-vous que le client peut accéder à l'application et que celle-ci se comporte de la même façon que dans l'environnement d'exécution précédent.

Traitement des problèmes de migration des artefacts source

Si la migration des artefacts source depuis WebSphere Studio Application Developer Integration Edition échoue, vous pouvez résoudre ces échecs de plusieurs manières.

Voici quelques exemples de ces échecs :

- Si vous recevez les messages suivants :

```
"Message d'erreur de
migration"
Motif : Echec de migration fatal
Message : Contactez votre représentant IBM.
```

Vous devez consulter le fichier journal de WebSphere Integration Developer contenu dans le dossier .metadata du nouvel espace de travail pour voir les détails de l'erreur. Si possible, traitez la cause de l'erreur, supprimez le module créé dans le nouvel espace de travail et tentez une nouvelle migration.

Si l'Assistant de Migration se termine sans que ce message s'affiche, la liste des messages d'informations, d'avertissement et d'erreur s'affiche. Cela signifie que certaines parties du projet de service n'ont pas pu être migrées automatiquement et que des modifications manuelles doivent être effectuées pour terminer la migration.

Méthodes recommandées pour le processus de migration de l'artefact source

Il existe un certain nombre de méthodes recommandées pour le processus de migration de l'artefact source de WebSphere Studio Application Developer Integration Edition.

Les pratiques suivantes montrent comment concevoir des services de WebSphere Studio Application Developer Integration Edition pour assurer leur migration vers le nouveau modèle de programmation :

- Essayez d'utiliser l'activité **Affecter** autant que possible (par opposition au service de conversion qui est nécessaire uniquement lorsqu'une transformation avancée est requise). Vous devez utiliser cette méthode car un composant intermédiaire doit être créé pour que le module SCA appelle un service de transformateur. De plus, il n'existe pas de prise en charge de l'outil spécifique dans WebSphere Integration Developer pour les services de transformateur créés avec la version 5.1 (vous devez utiliser l'éditeur WSDL ou XML pour modifier le XSLT intégré dans le fichier WSDL si vous devez changer le comportement du service de transformateur).
- Spécifiez une partie par message WSDL et par spécification d'interopérabilité de services Web (WS-I), ainsi que le style préféré de la version 6.0.
- Utilisez le style doc-literal WSDL car il s'agit du style préféré dans la version 6.0.
- Vérifiez que chaque type complexe dispose d'un nom et qu'il peut être identifié de façon unique par son nom et son espace de nom cible. La méthode recommandée pour définir les types complexes et des éléments correspondants est la suivante (définition du type complexe suivie d'une définition d'élément utilisant ce type) :

```
<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>
```

L'exemple suivant montre un type complexe anonyme à éviter, car il peut provoquer des incidents lorsqu'un SDO est sérialisé en XML (élément contenant une définition de type complexe anonyme) :

```
<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>

```

- Si vous publiez un service pour des clients externes, générez un code de déploiement de service à l'aide des services Web IBM (par opposition à Apache SOAP/HTTP) car les services Web IBM sont directement pris en charge dans V6.0, contrairement aux services Web Apache.
- Il existe deux façons d'organiser des fichiers WSDL et XSD dans la version 5.1 pour réduire la réorganisation que vous devez effectuer pendant la migration. Dans la version 6.0, les artefacts partagés comme les fichiers WSDL et XSD doivent être situés dans les projets BI (*modules Business Integration et Bibliothèques*) afin d'être référencés par un service BI :
 - Conservez tous les fichiers WSDL partagés par plusieurs projets de service dans un projet Java que les projets de service peuvent référencer. Lors de la migration vers la version 6.0, vous créerez une bibliothèque Business Integration avec le même nom que le projet Java 5.1 partagé. Copiez tous les artefacts du projet Java 5.1 partagé vers la bibliothèque de sorte que l'Assistant de migration puisse résoudre les artefacts lorsqu'il migre les projets de service utilisant ces artefacts.
 - Conservez une copie locale de tous les fichiers WSDL/XSD qu'un projet de service référence dans le projet de service lui-même. Les projets de service WebSphere Studio Application Developer Integration Edition seront migrés vers un module Business Integration dans WebSphere Integration Developer. Un module ne peut pas avoir de dépendance sur d'autres modules (un projet de service ayant des dépendances sur un autre projet de service pour le partage des fichiers WSDL ou XSD ne migrera pas correctement).
- Evitez d'utiliser l'API de messagerie générique (Generic MDBs) de Business Process Choreographer (BPC) car elle n'est pas fournie dans la version 6.0. Une interface MDB offrant une connexion tardive n'est *pas* disponible dans la version 6.0.
- Utilisez l'API EJB générique de Business Process Choreographer par opposition à l'appel des beans de session générés spécifiques à une version particulière d'un processus. Ces beans de session ne seront pas générés dans la version 6.0.0.0.
- Si vous possédez un processus métier avec plusieurs réponses pour la même opération, assurez-vous que si l'une d'elles a des paramètres client, toutes les réponses pour cette opération ont les mêmes paramètres client, étant donné que dans la version 6.0, un seul ensemble de paramètres client par réponse d'opération est pris en charge.
- Concevez les fragments Java BPEL selon les directives suivantes :
 - Evitez l'envoi de paramètres WSIFMessage à toute classe Java personnalisée. Essayez de ne pas dépendre du format de données WSIFMessage dès que cela est possible.
 - Evitez, si possible, l'utilisation d'API métadonnées WSIF.
- Evitez de déclarer des variables locales dans des fragments Java BPEL portant le même nom que les variables BPEL. Etant donné que dans la version 6.0, les variables BPEL sont directement accessibles dans les fragments, des variables locales portant le même nom peuvent entraîner des conflits.
- Dans la mesure du possible, évitez de créer des services EJB ou Java descendants car le squelette Java/EJB généré à partir des messages et des types de port WSDL dépendra des classes WSIF (par exemple : WSIFFormatPartImpl). Créez d'abord les interfaces Java/EJB et créez un service autour de la classe Java ou de l'EJB (approche ascendante).
- Evitez de créer ou d'utiliser des interfaces WSDL qui référencent le type soapenc:Array, dans la mesure où ce type d'interface n'est pas pris en charge en mode natif dans le modèle de programmation SCA.

- Evitez de créer des types de message dont l'élément de niveau supérieur est un type de tableau (l'attribut maxOccurs est supérieur à un), dans la mesure où ce type d'interface n'est pas pris en charge en mode natif dans le modèle de programmation SCA.
- Définissez les interfaces WSDL de manière précise (dans la mesure du possible, évitez les types complexes XSD qui référencent le type `xsd:anyType`).
- Pour tout WSDL et XSD généré à partir d'un EJB ou d'un bean Java, vérifiez que l'espace de nom cible est unique (le nom du package et celui de la classe Java sont représentés par l'espace de nom cible) afin d'éviter des conflits lors de la migration vers WebSphere Process Server V6. WebSphere Process Server V6 ne permet pas à deux définitions WSDL/XSD différentes d'avoir le même nom et le même espace de nom cible. Cette situation se produit fréquemment si l'Assistant de service Web ou la commande Java2WSDL est utilisé sans indication explicite de l'espace de nom cible. L'espace de nom cible est unique pour le nom de package de l'EJB ou du bean Java, mais pas pour la classe elle-même. Des incidents peuvent donc survenir lorsqu'un service Web est généré pour plusieurs EJB ou beans Java d'un même package. La solution consiste à définir un package personnalisé pour le mappage de l'espace de nom dans l'Assistant de service Web. Vous pouvez aussi utiliser l'option de ligne de commande Java2WSDL **-namespace** pour vous assurer que l'espace de nom des fichiers générés correspond uniquement à la classe concernée.
- Si possible, utilisez un espace de nom unique pour chaque fichier WSDL. Les restrictions de la spécification WSDL 1.1 concernant l'importation de deux fichiers WSDL différents ayant le même espace de nom sont mises en oeuvre de façon très stricte dans WebSphere Integration Developer 6.0.

Restrictions du processus de migration (pour la migration des artefacts source)

Il existe certaines restrictions pour le processus de migration des artefacts source de WebSphere Studio Application Developer Integration Edition.

Les listes suivantes détaillent certaines des restrictions applicables au processus de migration des artefacts source :

Restrictions générales

- L'Assistant de Migration ne peut pas gérer l'intégralité des espaces de travail de WebSphere Studio Application Developer Integration Edition. Il est conçu pour migrer un projet de service WebSphere Studio Application Developer Integration Edition à la fois.
- L'Assistant de Migration ne migre pas les application binaires, il migre uniquement les artefacts source trouvés dans un projet de service WebSphere Studio Application Developer Integration Edition.
- Les beans de règles métier sont obsolètes dans WebSphere Process Server 6.0. Cependant, au cours de l'installation de WebSphere Process Server, une option permet d'installer la prise en charge des beans de règles métier obsolètes de sorte qu'ils s'exécuteront "tels quels" sur un serveur WebSphere Process Server 6.0. En revanche, il n'existe pas de prise en charge de l'outil spécifique pour les anciens beans de règles métier. Si vous souhaitez que les artefacts des anciens beans de règles métier soient compilés dans les outils, vous devez consulter la documentation de WebSphere Integration Developer afin d'ajouter ces composants obsolètes au serveur de test intégré à WebSphere Process Server 6.0, puis ajouter manuellement les fichiers jar obsolètes dans le chemin d'accès de classe du projet en tant que fichiers jar externes. Vous devez utiliser le nouvel outil de règles métier disponible dans WebSphere Integration Developer pour recréer les règles métier selon les spécifications de la version 6.0.
- La prise en charge de la messagerie étendue est obsolète dans WebSphere Process Server 6.0. Cependant, au cours de l'installation de WebSphere Process Server, une option permet d'installer la prise en charge des fonctions de la messagerie étendue obsolète de sorte que les applications existantes puissent s'exécuter "telles quelles" sur un serveur WebSphere Process Server 6.0. En revanche, il n'existe pas de prise en charge de l'outil spécifique pour les anciennes fonctions de messagerie étendue. Si vous souhaitez que les artefacts de l'ancienne messagerie étendue soient compilés dans les outils, vous devez consulter la documentation de WebSphere Integration Developer afin d'ajouter ces composants obsolètes au serveur de test intégré à WebSphere Process Server 6.0, puis ajouter manuellement les fichiers jar obsolètes dans le chemin d'accès de classe du projet en tant que fichiers jar externes.

- La liaison de données JMS standard fournie ne permet pas d'accéder aux propriétés d'en-tête JMS personnalisées. Vous devez écrire une liaison de données personnalisée pour que les services SCA puissent accéder à ces propriétés.

Restrictions du modèle de programmation SCA

- La spécification SDO version 1 ne fournit pas l'accès au tableau d'octets C ou COBOL. Cela affectera les personnes qui utilisent les multi-segments IMS.
- La spécification SDO version 1 pour la sérialisation ne prend pas en charge la construction REDEFINE de langage COBOL ou la construction d'union de langage C.
- Lorsque vous concevez de nouveau vos artefacts source selon le modèle de programmation SCA, notez que le style WSDL encapsulé Document/Literal (qui est le style par défaut pour les nouveaux artefacts créés à l'aide des outils de WebSphere Integration Developer) ne prend pas en charge la surcharge de méthode. Les autres styles WSDL étant encore pris en charge, il est recommandé d'utiliser dans ce cas un style/codage WSDL autre que le style encapsulé Document/Literal.
- La prise en charge en mode natif des tableaux est limitée. Pour appeler un service externe qui expose les types soapenc:Array à une interface WSDL, vous devrez créer une interface WSDL qui définit un élément dont l'attribut "maxOccurs" est supérieur à un (il s'agit de l'approche recommandée pour concevoir un type de tableau).

Restrictions techniques d'un processus de migration BPEL

- **Réponses multiples par opération BPEL.** Dans WebSphere Business Integration Server Foundation, un processus métier pouvait avoir une activité recevoir et des activités répondre multiple pour la même opération.
- **Restrictions pour la migration des fragments Java BPEL** Le modèle de programmation a changé de manière significative entre WebSphere Studio Application Developer Integration Edition et WebSphere Integration Developer et les API WebSphere Studio Application Developer Integration Edition prises en charge ne peuvent pas toutes être migrées vers les API correspondantes de WebSphere Integration Developer. Toute logique Java peut être trouvée dans les fragments Java BPEL ; l'outil de migration automatique peut donc ne pas être capable de convertir chaque fragment Java vers le nouveau modèle de programmation. La plupart des appels d'API de fragments standard seront automatiquement migrés du modèle de programmation de fragments Java de la version 5.1 vers le modèle de programmation de fragments Java de la version 6.0. Les appels API WSIF sont migrés vers les appels API DataObject lorsque cela est possible. Toutes les classes Java personnalisées qui acceptent les objets WSIFMessage auront besoin d'une migration manuelle afin qu'elles acceptent et retournent les objets `com.ibm.sdo.DataObject` :
 - **API de métadonnées WSIFMessage.** Toute utilisation de l'API WSIF doit être éliminée lorsque cela est possible. L'Assistant de migration ne peut pas migrer automatiquement les métadonnées WSIFMessage et les autres API WSIF vers l'équivalent SDO. Une migration manuelle est donc nécessaire.
 - **API EndpointReference/EndpointReferenceType.** Ces classes ne sont pas migrées automatiquement. La migration manuelle est nécessaire car les méthodes getter/setter des liens partenaires traitent les objets `com.ibm.sdo.DataObject` à la place des objets `com.ibm.websphere.srm.bpel.wsaddressing.EndpointReferenceType` depuis la version 5.1.
 - **Types complexes avec noms en double** Si une application déclare des types complexes (dans WSDLs ou XSDs) avec des espaces de nom et des noms locaux identiques, ou des espaces de nom différents mais des noms locaux identiques, les fragments Java qui utilisent ces types peuvent ne pas être migrés correctement. Vérifiez la validité des fragments quand l'Assistant de migration a terminé.
 - **Types complexes avec noms locaux identiques aux classes Java dans le package java.lang.** Si une application déclare des types complexes (dans WSDLs ou XSDs) avec des noms locaux identiques aux classes dans le package `java.lang` package de J2SE 1.4.2, les fragments Java qui utilisent la classe `java.lang` correspondante peuvent ne pas être migrés correctement. Vérifiez la validité des fragments quand l'Assistant de migration a terminé.

- **Variables BPEL portant le même nom que les variables locales dans les fragments Java BPEL.** Si vous avez défini des variables locales avec le même nom que les variables BPEL dans vos fragments Java BPEL, vous devrez résoudre ce problème manuellement, car les variables BPEL sont maintenant accessibles par nom dans les fragments Java, et cela peut entraîner un conflit.
- **Variables BPEL en lecture seule et en lecture-écriture.** Dans les fragments Java 5.1, il était impossible de définir une variable BPEL en "lecture seule" (les modifications apportées à l'objet n'étaient pas répercutées sur la valeur de la variable). Il était également possible de définir une variable BPEL en "lecture-écriture" (les modifications apportées à l'objet étaient répercutées sur la valeur de la variable). L'exemple ci-dessous présente quatre moyens de rendre un fragment de code Java accessible en lecture seule dans un fragment Java BPEL 5.1 :

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

L'exemple ci-dessous présente deux moyens de rendre une variable BPEL accessible en lecture-écriture dans un fragment Java BPEL 5.1 :

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

Dans la version 6.0, l'accès en lecture seule et en lecture-écriture aux variables BPEL est géré fragment par fragment. Vous pouvez ajouter un commentaire spécial au fragment Java BPEL pour indiquer si les mises à jour de cette variable doivent être supprimées ou conservées après l'exécution du fragment de code. Les paramètres d'accès par défaut pour les types de fragments Java BPEL 6.0 sont les suivants :

```
BPEL Java Snippet Activity
Default Access: read-write
Override Default Access with comment containing:
    @bpe.readOnlyVariables names="variableA,variableB"
```

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

Lors de la migration, ces commentaires sont créés automatiquement si une variable a fait l'objet d'un accès d'un type autre que l'accès par défaut de la version 6.0. En cas de conflit (accès à une variable BPEL en lecture seule et en lecture-écriture dans un même fragment), un avertissement s'affiche et l'accès est défini sur lecture-écriture. Si vous recevez cet avertissement, assurez-vous que le fait que cette variable soit accessible en lecture-écriture ne pose pas de problème. Sinon, corrigez ce paramètre manuellement à l'aide de l'éditeur BPEL de WebSphere Integration Developer.

- **Propriétés primitives à valeurs multiples dans des types complexes.** Dans la version 5.1, les propriétés à valeurs multiples sont représentées par des tableaux de type de propriété. Ainsi, des appels pour obtenir et définir la propriété utilisent des tableaux. Dans la version 6.0, java.util.List est utilisé pour cette représentation. La migration automatique gère tous les cas où la propriété à valeurs multiples est un certain type d'objet Java, mais dans le cas où le type de la propriété est une primitive Java (int, long, short, byte, char, float, double et boolean), les appels pour obtenir et définir le tableau entier ne sont pas convertis. Dans un tel cas, la migration manuelle peut nécessiter l'ajout d'une boucle pour lier/délier les primitives dans/à partir de leur classe de conteneur Java correspondante (Integer, Long, Short, Byte, Character, Float, Double et Boolean) pour l'utilisation dans le reste du fragment.
- **Instanciation de classes générées représentant des types complexes.** Dans la version 5.1, les classes générées de types complexes définies dans une application pouvaient être facilement instanciées dans un fragment Java à l'aide du constructeur aucun argument par défaut. En voici un exemple :

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

Dans la version 6.0, une classe de fabrique spéciale doit être utilisée pour instancier ces types, ou une instance du type contenant peut être utilisée pour créer le sous-type. Si une variable de processus BPEL InputVariable a été définie comme ayant le type InputMessage, alors la version 6.0 du fragment précédent sera :

```
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");
```

Le convertisseur de fragments essaie d'effectuer ce changement, mais si l'ordre dans lequel s'effectuent les instanciations originales ne suit pas le schéma parent-puis-enfant, une migration manuelle sera nécessaire (c'est-à-dire que le convertisseur n'essaie pas de réorganiser intelligemment les déclarations d'instanciation dans le fragment).

- Sous WebSphere Business Integration Server Foundation 5.1, les références dynamiques étaient représentées par des parties de messages WSDL de type EndpointReferenceType ou par l'élément EndpointReference inclus dans l'espace de nom :

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

Ces références seront migrées de l'espace de nom du processus métier standard vers le type d'élément référence de service standard :

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

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

Reportez-vous à la documentation sur l'éditeur BPEL pour obtenir des instructions sur l'importation manuelle de ces définitions de schéma dans votre projet afin de résoudre correctement toutes les références.

- **BPEL variable message type** : Un type de message WSDL doit être indiqué pour toutes les variables BPEL utilisées dans des fragments Java. Les fragments Java accédant à des variables BPEL sans attribut "messageType" ne peuvent pas faire l'objet d'une migration.

Remarques

L'utilisation de la documentation XDoclet incluse dans ce produit IBM a été autorisée sous la mention de copyright suivante : Copyright (c) 2000-2004, XDoclet Team. Tous droits réservés.

Parties basées sur des *modèles de conception : éléments de logiciel orienté-objet réutilisables*, par Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides, Copyright (c) 1995 by Addison-Wesley Publishing Company, Inc. All rights reserved.

U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Le présent document peut contenir des informations ou des références concernant certains produits, logiciels ou services IBM non annoncés dans ce pays. Pour plus de détails, référez-vous aux documents d'annonce disponibles dans votre pays, ou adressez-vous à votre partenaire commercial IBM. Toute référence à un produit, logiciel ou service IBM n'implique pas que seul ce produit, logiciel ou service puisse être utilisé. Tout autre élément fonctionnellement équivalent peut être utilisé, s'il n'enfreint aucun droit d'IBM. Il est de la responsabilité de l'utilisateur d'évaluer et de vérifier lui-même les installations et applications réalisées avec des produits, logiciels ou services non expressément référencés par IBM.

IBM peut détenir des brevets ou des demandes de brevet couvrant les produits mentionnés dans la présente documentation. La remise de cette documentation ne vous donne aucun droit de licence sur ces brevets ou demandes de brevet. Si vous désirez recevoir des informations concernant l'acquisition de licences, veuillez en faire la demande par écrit à l'adresse suivante :

IBM EMEA Director of Licensing
IBM Europe Middle-East Africa Tour Descartes
92066 Paris-La Défense Cedex 50
France

Pour le Canada, veuillez adresser votre courrier à :

IBM Director of Commercial Relations
IBM Canada Ltd
3600 Steeles Avenue East
Markham, Ontario
L3R 9Z7 Canada

Les informations sur les licences concernant les produits utilisant un jeu de caractères double octet peuvent être obtenues par écrit à l'adresse suivante :

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

Le paragraphe suivant ne s'applique ni au Royaume-Uni, ni dans aucun pays dans lequel il serait contraire aux lois locales. LE PRESENT DOCUMENT EST LIVRE EN L'ETAT SANS AUCUNE GARANTIE EXPLICITE OU IMPLICITE. IBM DECLINE NOTAMMENT TOUTE RESPONSABILITE OU CONDITION RELATIVE A CES INFORMATIONS EN CAS DE CONTREFACON AINSI QU'EN CAS DE DEFAUT D'APTITUDE A L'EXECUTION D'UN TRAVAIL DONNE. Certaines juridictions n'autorisent pas l'exclusion des garanties implicites, auquel cas l'exclusion ci-dessus ne vous sera pas applicable.

Le présent document peut contenir des inexactitudes ou des coquilles. Ce document est mis à jour périodiquement. Chaque nouvelle édition inclut les mises à jour. IBM peut, à tout moment et sans préavis, modifier les produits et logiciels décrits dans ce document.

Les références à des sites Web non IBM sont fournies à titre d'information uniquement et n'impliquent en aucun cas une adhésion aux données qu'ils contiennent. Les éléments figurant sur ces sites Web ne font pas partie des éléments du présent produit IBM et l'utilisation de ces sites relève de votre seule responsabilité.

IBM pourra utiliser ou diffuser, de toute manière qu'elle jugera appropriée et sans aucune obligation de sa part, tout ou partie des informations qui lui seront fournies.

Les licenciés souhaitant obtenir des informations permettant : (i) l'échange des données entre des logiciels créés de façon indépendante et d'autres logiciels (dont celui-ci), et (ii) l'utilisation mutuelle des données ainsi échangées, doivent adresser leur demande à :

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

Ces informations peuvent être soumises à des conditions particulières, prévoyant notamment le paiement d'une redevance.

Le logiciel sous licence décrit dans cette documentation et tous les éléments sous licence disponibles s'y rapportant sont fournis par IBM conformément aux dispositions de l'ICA, des Conditions internationales d'utilisation des logiciels IBM ou de tout autre accord équivalent.

Les données de performance indiquées dans ce document ont été déterminées dans un environnement contrôlé. Par conséquent, les résultats peuvent varier de manière significative selon l'environnement d'exploitation utilisé. Certaines mesures évaluées sur des systèmes en cours de développement ne sont pas garanties sur tous les systèmes disponibles. En outre, elles peuvent résulter d'extrapolations. Les résultats peuvent donc varier. Il incombe aux utilisateurs de ce document de vérifier si ces données sont applicables à leur environnement d'exploitation.

Les informations concernant des produits non IBM ont été obtenues auprès des fournisseurs de ces produits, par l'intermédiaire d'annonces publiques ou via d'autres sources disponibles. IBM n'a pas testé ces produits et ne peut confirmer l'exactitude de leurs performances ni leur compatibilité. Elle ne peut recevoir aucune réclamation concernant des produits non IBM. Toute question concernant les performances de produits non IBM doit être adressée aux fournisseurs de ces produits.

Toute instruction relative aux intentions d'IBM pour ses opérations à venir est susceptible d'être modifiée ou annulée sans préavis, et doit être considérée uniquement comme un objectif.

Le présent document peut contenir des exemples de données et de rapports utilisés couramment dans l'environnement professionnel. Ces exemples peuvent mentionner des noms fictifs de personnes, de sociétés, de marques ou de produits à des fins illustratives ou explicatives uniquement. Toute ressemblance avec des noms de personnes, de sociétés ou des données réelles serait purement fortuite.

LICENCE DE COPYRIGHT :

Le présent logiciel contient des exemples de programmes d'application en langage source destinés à illustrer les techniques de programmation sur différentes plateformes d'exploitation. Vous avez le droit de copier, de modifier et de distribuer ces exemples de programmes sous quelque forme que ce soit et sans paiement d'aucune redevance à IBM, à des fins de développement, d'utilisation, de vente ou de distribution de programmes d'application conformes aux interfaces de programmation des plateformes pour lesquels ils ont été écrits ou aux interfaces de programmation IBM. Ces exemples de programmes n'ont pas été rigoureusement testés dans toutes les conditions. Par conséquent, IBM ne peut garantir expressément ou implicitement la fiabilité, la maintenabilité ou le fonctionnement de ces programmes. Vous avez le droit de copier, de modifier et de distribuer ces exemples de programmes sous quelque

forme que ce soit et sans paiement d'aucune redevance à IBM, à des fins de développement, d'utilisation, de vente ou de distribution de programmes d'application conformes aux interfaces de programmation IBM.

Toute copie totale ou partielle de ces programmes exemples et des oeuvres qui en sont dérivées doit comprendre une notice de copyright, libellée comme suit :

(C) (nom de votre société) (année). Des segments de code sont dérivés des Programmes exemples d'IBM Corp. (C) Copyright IBM Corp. 2000, 2005. Tous droits réservés.

Si vous visualisez ces informations en ligne, il se peut que les photographies et illustrations en couleur n'apparaissent pas à l'écran.

Documentation sur l'interface de programmation

La documentation sur l'interface de programmation aide les utilisateurs à créer des applications en utilisant le produit.

Les interfaces de programmation génériques vous permettent d'écrire des applications, qui bénéficient des services proposés par les outils du produit.

Toutefois, lesdites informations peuvent également contenir des données de diagnostic, de modification et d'optimisation qui permettent de déboguer votre application.

Avertissement : N'utilisez pas les informations de diagnostic, de modification et d'optimisation en guise d'interface de programmation car elles peuvent être modifiées sans préavis.

Marques

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



SC11-2366-00

