

버전 6.0.1



이주 안내서

버전 6.0.1



이주 안내서

주!

이 정보와 이 정보가 지원하는 제품을 사용하기 전에, "주의사항"에 있는 일반 정보를 읽으십시오.

목차

WebSphere Integration Developer를 사용한 응용 프로그램 이주	1
PDF 버전	1
학습서: WebSphere Integration Developer로 이주	1
WebSphere Integration Developer로 이주	1
WebSphere InterChange Server에서 WebSphere Process Server로 이주	1

WebSphere MQ Workflow에서 WebSphere Integration Developer로 이주	36
소스 아티팩트를 WebSphere Studio Application Developer Integration Edition에서 WebSphere Integration Developer로 이주	42
주의사항	115

WebSphere Integration Developer를 사용한 응용프로그램 이주

WebSphere® Integration Developer 버전 6.0에서는 기존 환경을 이주하는 데 필요한 도구를 제공합니다.

다음 주제에서는 WebSphere Integration Developer로의 이주에 대한 개념, 참조 및 단계별 지시사항을 설명합니다.

PDF 버전

이 이주 정보는 PDF 형식으로도 사용 가능합니다.

이 문서는 PDF 파일로도 사용 가능합니다.

PDF 형식의 문서를 보려면 Adobe Acrobat이 필요합니다. 이 소프트웨어의 무료 버전은 www.adobe.com에서 구할 수 있습니다.

학습서: WebSphere Integration Developer로 이주

이 학습서에서는 WebSphere Integration Developer를 사용하여 기존 서비스 프로젝트를 이주한 다음 해당 프로젝트를 모듈 프로젝트로 이주하는 방법에 대한 정보를 제공합니다.

이 학습서에는 한 개의 단원이 있으며 이 단원은 동영상 형식으로 되어 있습니다.

- 단원 1: 서비스 프로젝트 이주에서는 기존 서비스 프로젝트를 가져와서 모듈 프로젝트로 이주하는 방법을 보여 줍니다.

다음 링크를 클릭하여 학습서를 시작하십시오.

주: 다음 링크는 이 웹 사이트에서 작동하지 않습니다. 링크가 올바르게 작동하려면 WebSphere Integration Developer가 설치되어 있어야 합니다.

학습서: WebSphere Integration Developer로 이주

WebSphere Integration Developer로 이주

WebSphere Integration Developer 버전 6.0에서는 기존 환경을 이주하는 데 필요한 도구를 제공합니다.

다음 주제에서는 WebSphere Integration Developer로의 이주에 대한 개념, 참조 및 단계별 지시사항을 설명합니다.

WebSphere InterChange Server에서 WebSphere Process Server로 이주

WebSphere InterChange Server에서 WebSphere Process Server로의 이주는 다음 기능을 통해 지원됩니다.

주: WebSphere Process Server 릴리스에서의 이주와 관련된 제한사항에 대한 정보는 릴리스 정보[®]를 참조하십시오.

- 다음에서 호출할 수 있는 이주 도구를 통한 소스 아티팩트의 자동 이주:
 - WebSphere Integration Developer의 파일 → 가져오기 메뉴
 - WebSphere Integration Developer의 환영 화면
 - WebSphere Process Server의 이주 마법사 - 첫 번째 단계
 - **reposMigrate** 명령행 유틸리티
- 여러 WebSphere InterChange Server API의 런타임에서의 기본 지원
- 기존 어댑터를 WebSphere Process Server와 호환 가능하도록 현재 WebSphere Business Integration 어댑터 기술에 대한 지원.

소스 아티팩트 이주가 지원되더라도 결과 응용프로그램이 WebSphere Process Server에서 예상하는 대로 작동하는지 또는 이주 후 재디자인이 필요한지 등을 확인하기 위해 확장 분석 및 테스트를 수행하는 것이 좋습니다. 이러한 권장사항은 WebSphere InterChange Server와 이 버전의 WebSphere Process Server 사이 기능 패리티의 다음 제한사항에 기반합니다. 이 버전의 WebSphere Process Server에는 다음과 같은 WebSphere InterChange Server 기능과 동일한 기능은 지원되지 않습니다.

- 액세스 인터페이스
- 어댑터 초기 동기 요청 응답
- 제한시간이 있는 동기 전송 서비스 호출
- Async_in 서비스 호출
- 격리
- 이벤트 순차화
- 즉시 배치/동적 갱신
- 보안 - 감사
- 보안 - 세부 RBAC
- 그룹 지원
- 스케줄러 - 일시정지 오퍼레이션
- 보안 설명자 이주 방지
- 맵 및 협업 템플릿 이주 중에 사용자 정의 XML 스니펫 변환은 지원되지 않음

WebSphere InterChange Server에서 지원되는 이주 경로

WebSphere Process Server 이주 도구는 WebSphere InterChange Server 버전 4.2.2, 4.2.3 및 4.3에서의 이주를 지원합니다.

버전 4.2.2 이전의 모든 WebSphere InterChange Server 릴리스는 WebSphere Process Server로 이주하기 전에 먼저 버전 4.2.2, 4.2.3 또는 4.3으로 이주해야 합니다.

WebSphere InterChange Server에서의 이주 준비

WebSphere InterChange Server에서 WebSphere Process Server로 이주하기 전에 먼저 환경이 올바르게 준비되었는지 확인해야 합니다. WebSphere Process Server는 WebSphere InterChange Server에서 이주하는 데 필요한 도구를 제공합니다.

이러한 이주 도구는 다음 방법으로 호출할 수 있습니다.

- WebSphere Integration Developer의 파일 → 가져오기 메뉴
- WebSphere Integration Developer의 환영 화면
- WebSphere Process Server의 이주 마법사 - 첫 번째 단계

이주 도구에 대한 입력은 WebSphere InterChange Server에서 내보낸 저장소 jar 파일입니다. 따라서 이러한 옵션을 통해 이주 도구에 액세스하기 전에 다음을 수행해야 합니다.

1. WebSphere Process Server로 이주할 수 있는 버전의 WebSphere InterChange Server를 실행 중인지 확인해야 합니다. "지원되는 WebSphere InterChange Server 이주 경로" 주제를 참조하십시오.
2. WebSphere InterChange Server 문서에서 설명한 바와 같이 WebSphere InterChange Server **repos_copy** 명령을 사용하여 WebSphere InterChange Server에서 저장소 JAR 파일로 소스 아티팩트를 내보내야 합니다. 이 JAR 파일은 이주 도구의 입력이 됩니다.

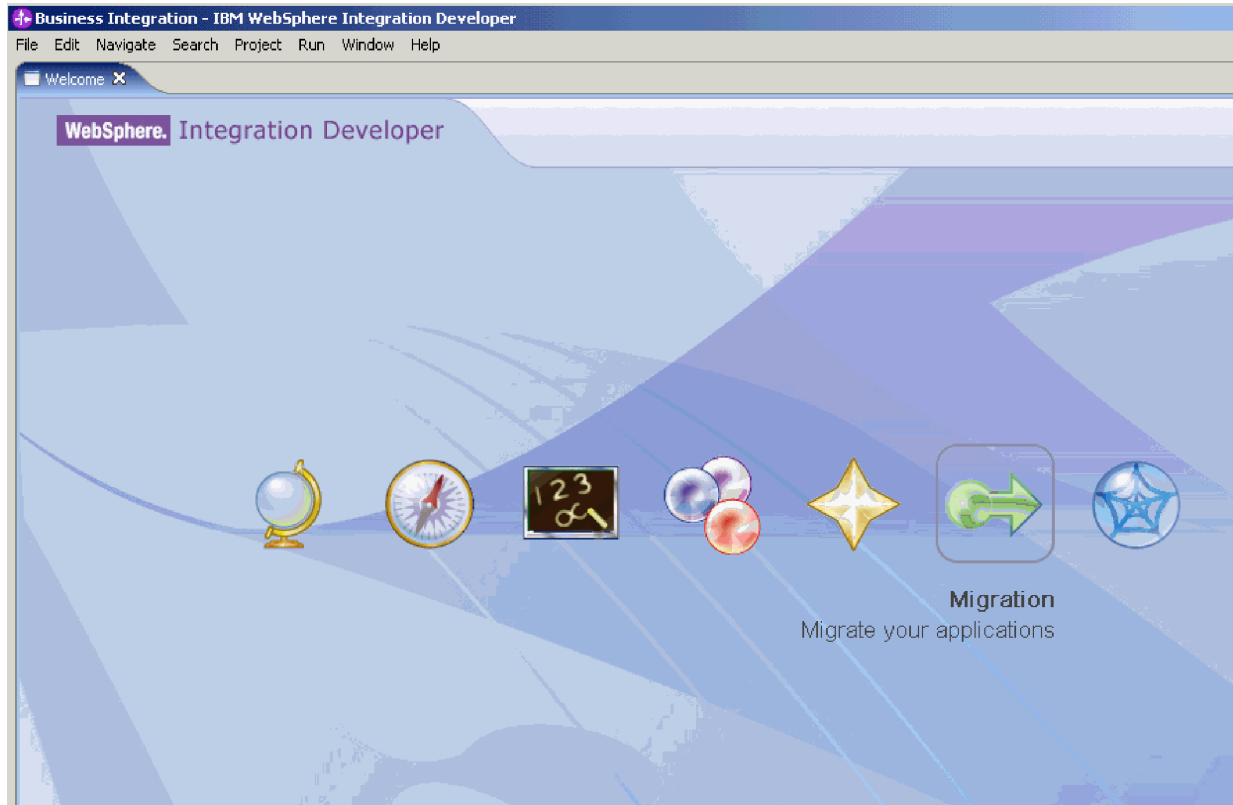
이주 마법사를 사용한 WebSphere InterChange Server 이주

WebSphere Integration Developer 이주 마법사를 사용하여 기존의 WebSphere InterChange Server 아티팩트를 이주할 수 있습니다.

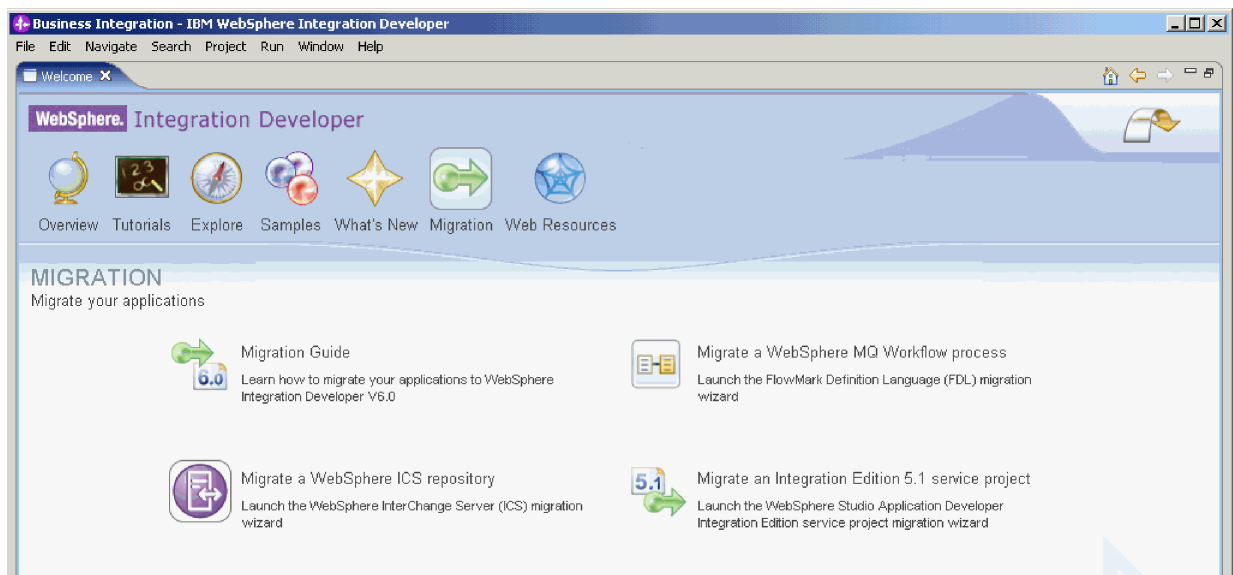
이주 마법사를 사용하여 다음 단계에 따라 WebSphere InterChange Server 아티팩트를 이주하십시오.



1. 환영 페이지에서 을 클릭하여 이주 페이지를 여십시오.



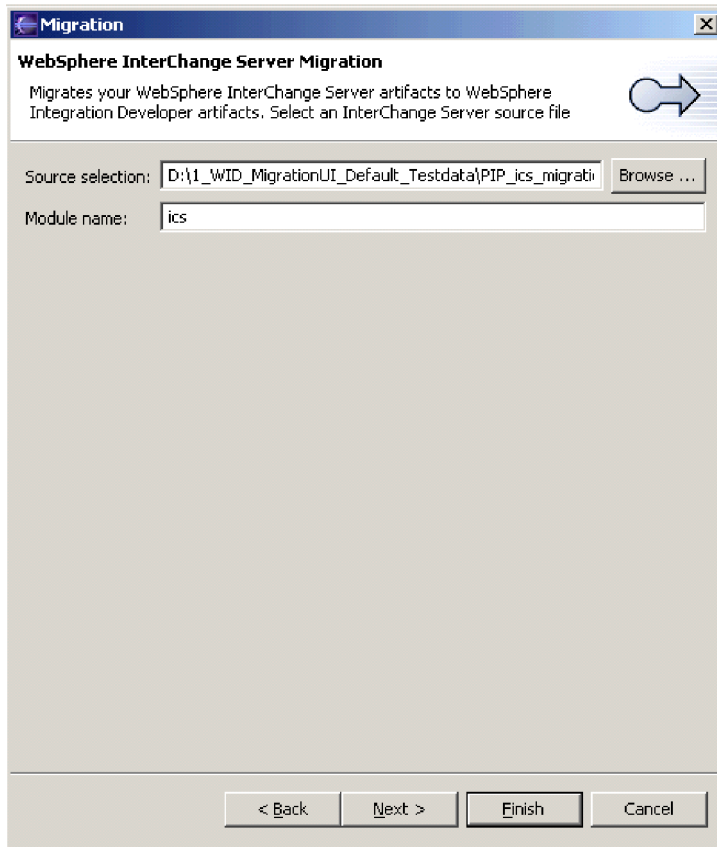
2. 이주 페이지에서 WebSphere ICS 저장소 이주 옵션을 선택하십시오.



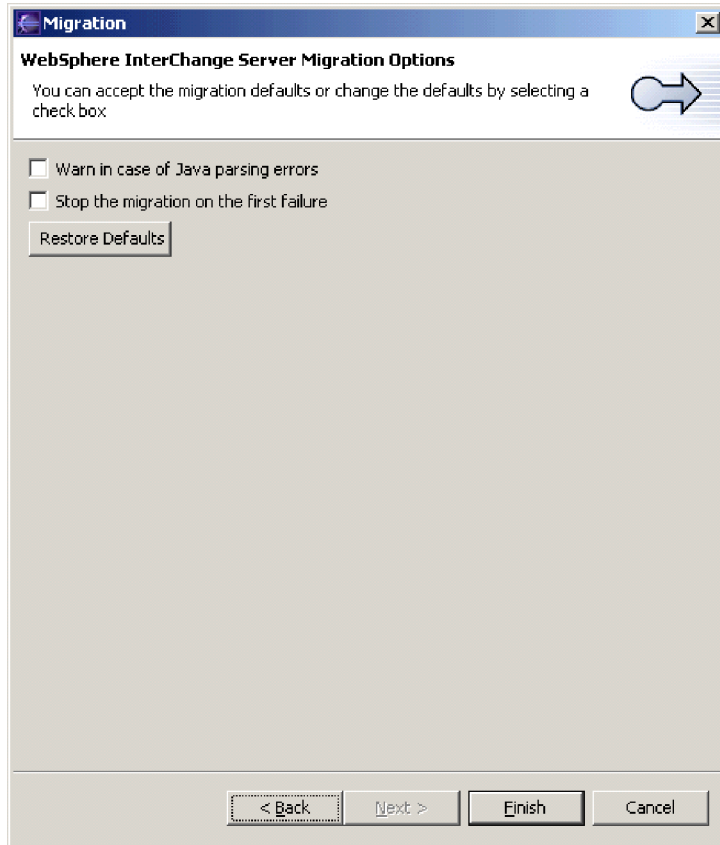
또한 파일 → 가져오기 메뉴 옵션에서 **WebSphere InterChange Server JAR** 파일을 선택하고 다음을 클릭하여 이주 마법사를 시작할 수 있습니다.

3. 이주 마법사가 열립니다. 찾아보기 단추를 클릭하고 파일을 탐색하여 소스 선택 필드에 소스 파일의 이름을 입력하십시오. 관련 필드에 모듈 이름을 입력하십시오. 다음을 클릭하십시오.

- 4 IBM WebSphere Integration Developer: 이주 안내서



- 이주 옵션 창이 열립니다. 이 페이지에서 이주 기본값을 승인하거나 옵션을 변경하려면 선택란을 선택하십시오.



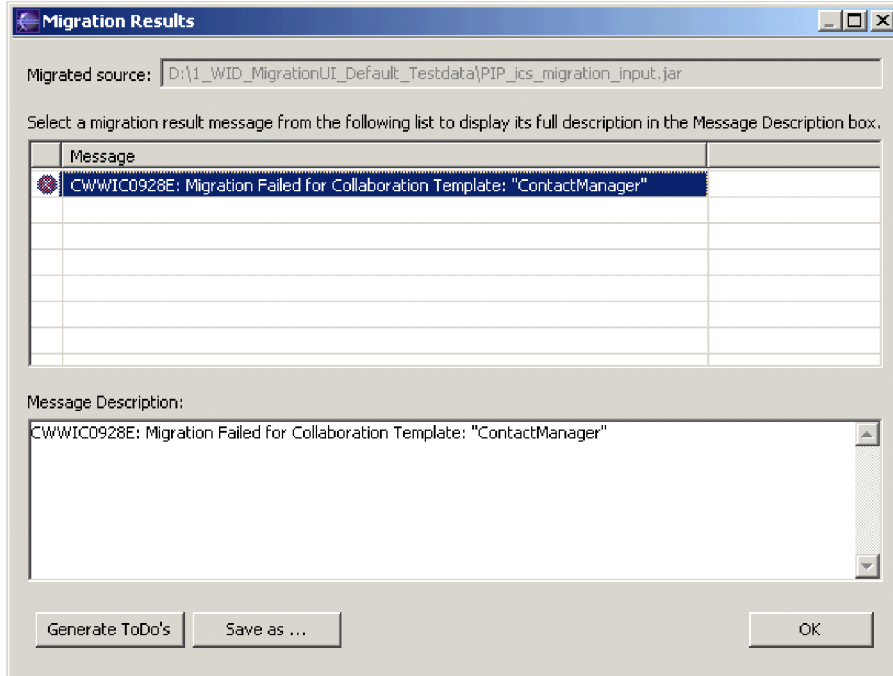
완료를 클릭하십시오.

WebSphere InterChange Server 이주 확인

WebSphere InterChange Server JAR 파일을 이주하는 동안 오류가 보고되지 않으면, 아티팩트 이주가 성공한 것입니다. 이주 마법사가 완료되지 않으면 오류, 경고 또는 정보 메시지로 구성된 목록이 표시됩니다. 이 메시지를 사용하여 WebSphere InterChange Server 이주를 확인할 수 있습니다.

주: WebSphere InterChange Server에서 WebSphere Process Server로 이주하려면 복잡하므로, 프로덕션에 삽입하기 전에 WebSphere Process Server에서 실행 중인 결과 응용프로그램의 확장 테스트를 수행하여 예상한 대로 작동하는지 확인해야 합니다.

이주 마법사가 완료되었을 때 다음 페이지가 표시됩니다.



이주 결과 창에서 이주 메시지 목록을 볼 수 있습니다. 메시지를 클릭하여 세부사항의 전체 설명을 보십시오. 필요하다면 이 메시지 목록에서, 수행할 작업 생성 단추를 클릭하여 수정 항목의 목록을 작성할 수 있습니다.

WebSphere InterChange Server에서 이주 실패에 대한 처리

WebSphere InterChange Server에서 이주에 실패하면, 이를 처리하는 두 가지 방법이 있습니다.

주: WebSphere InterChange Server에 익숙한 경우 첫 번째 옵션을 선호할 수 있습니다. 그러나 WebSphere Process Server 및 새 아티팩트에 익숙해지면 WebSphere Integration Developer에서 이주한 아티팩트를 수정하도록 선택할 수 있습니다.

1. 오류 네이처가 허용되면, WebSphere InterChange Server 아티팩트를 WebSphere InterChange Server 도구 세트를 사용하여 조정하고, JAR 파일을 다시 내보낸 후 이주를 재시도할 수 있습니다.
2. WebSphere Integration Developer에서 아티팩트를 편집하여 그 결과로 발생하는 WebSphere Process Server 아티팩트의 모든 오류를 수정할 수 있습니다.

이주 도구에 의해 처리되는 WebSphere InterChange Server 아티팩트

이주 도구를 사용하면 일부 WebSphere InterChange Server 아티팩트를 자동으로 이주할 수 있습니다.

다음 아티팩트를 이주할 수 있습니다.

- WebSphere Process Server 비즈니스 오브젝트가 된 비즈니스 오브젝트
- WebSphere Process Server 맵이 된 맵
- WebSphere Process Server 관계 및 역할이 된 관계
- WebSphere Process Server BPEL 및 WSDL이 된 협업 템플릿
- WebSphere Process Server SCA 아티팩트가 된 협업 오브젝트

- WebSphere Process Server SCA 아티팩트가 된 커넥터 정의

이주 도구를 사용하여 WebSphere Process Server의 자원을 다음 WebSphere InterChange Server 아티팩트/자원에 대해 자동으로 구성할 수 있습니다.

- DBConnection 풀
- 관계 데이터베이스
- 스케줄러 항목

이주 도구에서는 다음 WebSphere InterChange Server 아티팩트를 처리하지 않습니다.

- 벤치마크 아티팩트

지원되는 WebSphere InterChange Server API

WebSphere Process Server 및 WebSphere Integration Developer에서 제공하는 WebSphere InterChange Server 소스 아티팩트 이주 도구 외에, WebSphere InterChange Server에서 제공되는 많은 API도 지원됩니다. 이주 도구는 이주 시 가능한 많은 사용자 정의 스니펫 코드를 보존하여 이들 WebSphere InterChange Server API와 결합하여 작동합니다.

주: 이러한 API는 새로운 Process Server API를 사용할 수 있도록 수정될 때까지 이주된 WebSphere InterChange Server 응용프로그램만을 지원하도록 제공됩니다. WebSphere InterChange Server API는 모두 권장되지 않으며 후속 릴리스에서 제거됩니다.

Process Server에서 지원되는 WebSphere InterChange Server API가 아래 나열되어 있습니다. 이들 API는 WebSphere InterChange Server에서 제공하는 함수와 유사한 함수를 Process Server에 제공합니다. 해당 API의 함수에 대한 설명은 WebSphere InterChange Server 문서를 참조하십시오.

BusObj

Collaboration/

- BusObj(DataObject)
- BusObj(String)
- BusObj(String, Locale)
- copy(BusObj)
- duplicate():BusObj
- equalKeys(BusObj):boolean
- equals(Object):boolean
- equalsShallow(BusObj):boolean
- exists(String):boolean
- get(int):Object
- get(String):Object
- getBoolean(String):boolean

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

- setWithCreate(String, BusObjArray)
- setWithCreate(String, Object)
- toString():String
- validData(String, boolean):boolean
- validData(String, BusObj):boolean
- validData(String, BusObjArray):boolean
- validData(String, double):boolean
- validData(String, float):boolean
- validData(String, int):boolean
- validData(String, long):boolean
- validData(String, Object):boolean
- validData(String, String):boolean

BusObjArray

Collaboration/

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

BaseDLM

DLM/

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

- `raiseException(String, int)`
- `raiseException(String, int, Object[])`
- `raiseException(String, int, String)`
- `raiseException(String, int, String, String)`
- `raiseException(String, int, String, String, String)`
- `raiseException(String, int, String, String, String, String)`
- `raiseException(String, int, String, String, String, String, String)`
- `raiseException(String, String)`
- `releaseRelConnection(boolean)`
- `trace(int, int)`
- `trace(int, int, Object[])`
- `trace(int, int, String)`
- `trace(int, int, String, String)`
- `trace(int, int, String, String, String)`
- `trace(int, int, String, String, String, String)`
- `trace(int, int, String, String, String, String, String)`
- `trace(int, String)`
- `trace(String)`

CwDBConnection

CwDBConnection/

CxCommon/

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

- rollback()

CwDBConstants

CwDBConnection/

CxCommon/

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

CwDBStoredProcedureParam

CwDBConnection/

CxCommon/

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

DtpConnection

Dtp/

CxCommon/

- beginTran()
- commit()
- executeSQL(String)
- executeSQL(String, Vector)
- 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, String)
- ServiceCallException - "ConsumerException"
- ServiceCallTransportException - "ServiceCallTransportException"
- SystemException - "SystemException"
- trace(int, int)
- trace(int, int, Object[])
- trace(int, int, String)
- trace(int, int, String, String)
- trace(int, int, String, String, String)

- trace(int, int, String, String, String, String)
- trace(int, int, String, String, String, String, String)
- trace(int, String)
- trace(String)
- TransactionException - "TransactionException"

CxExecutionContext

CxCommon/

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

CollaborationException

Collaboration/

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

Filter

crossworlds/

com/

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

Globals

crossworlds/

com/

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

SmartCollabService

crossworlds/

com/

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

StateManagement

crossworlds/

com/

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

EventKeyAttrDef

EventManagement/

CxCommon/

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

EventQueryDef

EventManagement/

CxCommon/

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

FailedEventInfo

EventManagement/

CxCommon/

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

CwBiDiEngine

- BiDiBOTransformation(BusinessObject, String, String, boolean):BusinessObj
- BiDiBusObjTransformation(BusObj, String, String, boolean):BusObj
- BiDiStringTransformation(String, String, String):String

WebSphere InterChange Server XML에서 WebSphere Process Server DataObject 매핑

Legacy Adapters를 사용하여 WebSphere Process Server에 연결하는, 다음 알고리즘을 통해 WebSphere Process Server DataObject가 WebSphere InterChange Server XML로부터 작성되는 방법을 이해할 수 있습니다. 이 정보는 데이터 값이 배치된 위치와 WebSphere InterChange Server에서 사용된 데이터 값을 대체하도록 선택된 데이터 값을 또한 표시합니다.

일반사항

- ChangeSummary에서 verb를 설정할 경우, 모든 설정이 **markCreate/Update/Delete** API를 사용하여 수행됩니다.
- ChangeSummary/EventSummary에서 verb 설정의 경우, **Create, Update** 및 **Delete** verb는 ChangeSummary에서 설정되지만 모든 기타 verb는 EventSummary에서 설정됩니다.
- ChangeSummary에서 verb 가져오기:
 - isDelete가 true일 경우, verb는 **Delete**입니다.

주: isDelete가 true일 경우 isCreate 값이 무시됩니다. 이러한 상황은 DataObject에서 삭제 지점 허브를 입력하기 전에 작성이 표시된 경우 또는 허브에서 작성 및 삭제가 일어난 경우 발생할 수 있습니다.

- isDelete가 false이고 isCreate가 true일 경우, verb는 **Create**입니다.
- isDelete가 false이고 isCreate가 false일 경우, verb는 **Update**입니다.
- DataObject가 **Update** 대신 **Create**로 식별되지 않게 하려면, 로깅이 사용 가능할 경우 다음을 수행하십시오.
 - DataObject 작성 중에 로깅 일시중지
 - DataObject 갱신에 대한 로깅 재개(또는 **markUpdated** API 사용)

로딩

로딩은 WebSphere InterChange Server 런타임 XML을 WebSphere Business Integration BusinessGraph AfterImage 인스턴스로 로드합니다.

- 적절한 BusinessGraph 인스턴스가 작성됩니다.
- 나중에 쿼을 때 항목이 지워지지 않도록 ChangeSummary Logging을 켭니다.
- ChangeSummary에 원하지 않는 정보가 입력되지 않도록 ChangeSummary Logging을 일시정지합니다.
- 최상위 레벨 BusinessObject의 속성이 DataObject에 작성됩니다(아래 "속성 처리" 섹션 참조).
- 최상위 레벨 BusinessObject에 하위 BusinessObject가 있을 경우, 순환식으로 처리됩니다.

- 해당 하위 BusinessObject의 속성이 DataObject에 작성됩니다(아래 "속성 처리" 섹션 참조).
- 최상위 레벨 BusinessObject의 verb가 BusinessGraph의 최상위 레벨 verb로 설정되며 요약에 설정됩니다.
- 하위 BusinessObject의 verb가 요약에 설정됩니다.

저장

저장은 WebSphere Business Integration BusinessGraph AfterImage 인스턴스를 WebSphere InterChange Server 런타임 XML에 저장합니다. 입력 BusinessGraph가 AfterImage가 아닐 경우 예외가 발생합니다.

- WebSphere InterChange Server XML 문서 인스턴스가 작성됩니다.
- ChangeSummary의 삭제된 모든 DataObject는 원래의 위치에 존재했던 것처럼 다루어집니다.

주: 이러한 삭제 취소 프로세스는 목록 순서를 유지하지 않습니다.

- 최상위 레벨 BusinessObject의 verb가 BusinessGraph의 최상위 레벨 verb로 설정됩니다.
- 최상위 레벨 BusinessObject의 속성은 DataObject로부터 설정됩니다(아래 "속성 처리" 섹션 참조).
- DataObject에 하위 DataObject가 있을 경우, 이들은 귀납(반복)으로 처리됩니다.
- 하위 DataObject verb의 처리 방법은 다음과 같습니다.
 - EventSummary 또는 ChangeSummary에 하위 DataObject에 대한 verb가 없을 경우, verb가 ""(비어 있는 문자열)로 설정됩니다.
 - EventSummary에는 verb가 존재하지만 ChangeSummary에는 존재하지 않을 경우, 해당 verb를 사용합니다.
 - ChangeSummary에는 verb가 존재하지만 EventSummary에는 존재하지 않을 경우, 해당 verb를 사용합니다.
 - ChangeSummary 및 EventSummary 둘 다에 verb가 존재할 경우, ChangeSummary의 값이 EventSummary의 값보다 우선적으로 선택됩니다.
- 하위 DataObject의 속성은 BusinessObject에 설정됩니다(아래 "속성 처리" 섹션 참조).

속성 처리

- 아래에서 다루어지지 않는 모든 값은 있는 그대로 로드되거나 저장됩니다.
- ObjectEventId는 EventSummary로부터 로드되거나 또는 EventSummary로 저장됩니다.
- **CxBlank** 및 **CxIgnore**의 경우:
 - WebSphere Business Integration BusinessObject의 변환 측면에서 **CxBlank** 및 **CxIgnore**는 다음과 같이 설정/식별됩니다.
 - **CxIgnore** - 널(null) Java™ 값을 사용하여 설정 취소 또는 설정됨
 - **CxBlank** - 아래 테이블에 표시된 대로 유형 종속 값
 - WebSphere InterChange Server XML의 변환 측면에서 **CxBlank** 및 **CxIgnore**는 다음과 같이 설정/식별됩니다.

표 1. CxBlank 및 CxIgnore 설정

유형	CxIgnore	CxBlank
Int	Integer.MIN_VALUE	Integer.MAX_VALUE
Float	Float.MIN_VALUE	Float.MAX_VALUE
Double	Double.MIN_VALUE	Double.MAX_VALUE
String/date/longtext	“CxIgnore”	“”
하위 BusinessObject	(비어 있는 요소)	해당 없음

WebSphere InterChange Server 이주 프로세스의 우수 사례

다음 가이드라인은 WebSphere InterChange Server에 사용되는 통합 아티팩트의 개발을 지원하기 위해 만들어진 것입니다. 이 가이드라인을 준수하여 WebSphere InterChange Server 아티팩트를 손쉽게 WebSphere Process Server로 이주할 수 있습니다.

이러한 권장사항은 새 통합 솔루션 개발에 대한 지침으로만 사용하는 것이 좋습니다. 기존의 콘텐츠는 이 가이드라인을 준수하지 않을 수 있습니다. 또한 해당 가이드라인을 위반해야 하는 경우도 있습니다. 이러한 경우에는 아티팩트를 이주하는 데 필요한 재작업의 양을 최소화하기 위해, 가이드라인의 위반 범위를 제한하도록 주의해야 합니다. 여기에서 요약된 가이드라인에는 일반적인 WebSphere InterChange Server 아티팩트 개발에 대한 우수 사례는 포함되어 있지 않습니다. 이 가이드라인은 나중에 아티팩트를 쉽게 이주하는 데 적용할 수 있는 고려사항으로 범위가 제한되어 있습니다.

일반 개발

대부분의 통합 아티팩트에 폭넓게 적용할 수 있는 몇 가지 고려사항이 있습니다. 일반적으로, 도구에서 제공되는 기능을 활용하고 해당 도구에 의해 강제되는 메타데이터 모델에 따르는 아티팩트는 대부분 자연스럽게 이주합니다. 또한 대규모 확장 및 외부 종속성을 갖는 아티팩트는 이주 시에 좀 더 직접적인 개입을 필요로 할 수도 있습니다.

다음 목록은 이후의 좀 더 쉬운 이주를 위해 WebSphere InterChange Server 기반 솔루션의 일반적 개발에 대한 우수 사례를 요약한 것입니다.

- 실시간 자동화 프로세스 통합 솔루션을 위한 WebSphere InterChange Server의 사용
- 시스템 및 구성요소 디자인의 문서화
- 통합 아티팩트 편집을 위한 개발 도구의 사용
- 도구 및 Java 스니펫을 사용하여 규칙을 정의하는 우수 사례 활용

당연한 사항이지만 통합 솔루션은 WebSphere InterChange Server에서 제공하는 프로그래밍 모델 및 아키텍처를 준수해야 합니다. WebSphere InterChange Server는 실시간 자동화 프로세스 통합 솔루션에 최적화되어 있습니다. 또한 WebSphere InterChange Server에 있는 각 통합 구성요소는 아키텍처 안에서 올바르게 정의된 역할을 수행합니다. 이 모델에서 크게 벗어나는 경우 WebSphere Process Server의 적합한 아티팩트로 콘텐츠를 이주하는 작업은 더욱 어려워집니다.

나중에 이주 프로젝트를 크게 성공할 수 있게 하는 또 다른 우수 사례는 시스템 디자인을 문서화하는 것입니다. 기능상의 디자인 및 서비스 요구사항의 품질, 프로젝트 간에 공유되는 아티팩트의 독립성 그리고 배치 동안에 발생하는 디자인 결정을 포함하여 통합 아키텍처 및 디자인을 캡처하십시오. 이 작업은 이주 중의 시스템 분석을 지원하며 재작업 노력을 최소화합니다.

아티팩트 정의를 작성, 구성 및 수정하려면 반드시 제공된 개발 도구만을 사용해야 합니다. 아티팩트 메타데이터의 수동 조작(예: XML 파일의 직접 편집)은 이주할 아티팩트를 손상시킬 수도 있으므로 피해야 합니다.

협업 템플릿, 맵, 공통 코드 유틸리티 및 기타 구성요소 안의 Java 코드를 개발할 경우 고려해야 할 몇 가지 사항이 있습니다.

- 공개된 API만 사용.
- 활동 편집기만 사용.
- 어댑터를 사용하여 EIS에 액세스.
- Java 스니펫 코드에서 외부 종속성 방지.
- 이식성을 위해 J2EE 개발 규칙 준수.
- 스레드를 출력하거나 Thread Synchronization 기본요소를 사용하지 마십시오. 사용해야 한다면, 이주할 때 비동기 Bean을 사용하도록 변환해야 합니다.
- java.io.*을 사용하여 디스크 I/O를 수행하지 마십시오. 데이터를 저장하려면 JDBC를 사용하십시오.
- 소켓 I/O, 클래스로딩, 기본 라이브러리 로딩 등과 같이 EJB 컨테이너에 예약되었을 수 있는 함수를 실행하지 마십시오. 해당 함수를 실행해야 한다면, 이주할 때 EJB 컨테이너 함수를 사용하도록 스니펫을 수동으로 변환해야 합니다.

아티팩트에 대한 제품 문서에 출력된 API만 사용하십시오. 이러한 API는 WebSphere InterChange Server 개발 안내서에 자세히 설명되어 있습니다. 많은 경우에 호환성 API는 WebSphere Process Server에서 제공되며 여기에는 출력된 API만 포함됩니다. WebSphere InterChange Server에는 개발자가 사용하려는 여러 내부 인터페이스가 있지만, 인터페이스가 계속 지원될 것으로 확신할 수는 없습니다.

맵과 협업 템플릿에서 비즈니스 로직 및 변환 규칙을 디자인할 때 활동 편집기 도구를 최대한 활용하십시오. 이 경우 비즈니스 로직은 새 아티팩트로 더욱 쉽게 변환될 수 있는 메타데이터를 통해 기술됩니다. 도구에서 재사용하려는 오퍼레이션의 경우 가능하면 활동 편집기의 "내 컬렉션" 기능을 사용하십시오. WebSphere InterChange Server의 클래스 경로에 Java 아카이브(*.jar) 파일로 포함된 펄드 개발 공통 코드 유틸리티 라이브러리는 수동으로 이주해야 하므로 사용하지 않는 것이 좋습니다.

개발해야 하는 Java 코드 스니펫에서 해당 코드는 가능한 단순하고 작게 작성하는 것이 좋습니다. Java 코드에서 정교화 수준은 기본 평가, 오퍼레이션, 계산, 데이터 형식화, 유형 변환 등과 관련된 스크립팅 순서로 유지되어야 합니다. 좀 더 폭넓고 정교한 응용프로그램 로직이 필요하면 WebSphere Application Server에서 실행되는 EJB를 사용하여 로직을 캡슐화하는 것이 좋습니다. 그런 다음 웹 서비스 호출을 사용하여 해당 EJB를 WebSphere InterChange Server에서 호출하십시오. 개별적으로 이주해야 하는 씨드파티 또는 외부 라이브러리보다는 표준 JDK 라이브러리를 사용하십시오. 또한 단일 코드 스니펫 안에 있는 관련된 모든 로직을 수집

하고 연결 및 트랜잭션 컨텍스트가 다중 코드 스니펫에 걸쳐 있는 로직은 사용하지 마십시오. 예를 들어, 데이터베이스 오퍼레이션의 경우 연결 확보, 트랜잭션 시작 및 종료, 연결 릴리스 등과 관련된 코드는 하나의 코드 스니펫으로 되어야 합니다.

일반적으로 EIS(Enterprise Information System)와 소통하도록 작성된 코드는 맵 또는 협업 템플릿이 아니라 어댑터 안에 위치해야 합니다. 이것은 일반적으로 아키텍처 디자인에 대한 우수 사례에 해당됩니다. 또한 코드 안에서 연결 관리 및 가능한 JNI(Java Native Interface) 구현과 같은 써드파티 라이브러리 및 관련 고려 사항에 대한 전제조건을 생략하는 데 도움을 줍니다.

적합한 예외 핸들을 사용하여 가능한 안전하게 코드를 작성하십시오. 또한 현재 J2SE 환경에서 실행되는 코드라 하더라도 해당 코드를 J2EE 응용프로그램 서버 환경에서 실행할 수 있게 호환되도록 작성하십시오. 정적 변수, 스레드 출력 및 디스크 I/O 금지 등과 같은 J2EE 개발 규칙을 준수하십시오. 이러한 규칙은 일반적으로 준수할 최상의 우수 사례이지만 이식성에 적합해야 합니다.

공통 코드 유틸리티

앞서 설명한 바와 같이 WebSphere InterChange Server 환경에서 통합 아티팩트에 걸쳐 사용할 공통 코드 유틸리티 라이브러리의 개발은 피하는 것이 좋습니다. 통합 아티팩트에 걸쳐 코드 재사용이 필요하다면 활동 편집기 도구의 “내 컬렉션” 기능을 활용하는 것이 좋습니다. 또한 WebSphere Application Server에서 실행되는 EJB를 사용하여 로직을 캡슐화하고 웹 서비스 호출을 사용하여 해당 EJB를 WebSphere InterChange Server에서 호출하십시오. 일부 공통 코드 유틸리티 라이브러리는 WebSphere Process Server에서 올바르게 실행될 수도 있지만 사용자는 사용자 정의 유틸리티의 이주에 책임이 있습니다.

데이터베이스 연결 풀

맵 및 협업 템플릿에서 사용자 정의 데이터베이스 연결 풀은 프로세스 인스턴스에 걸친 간단한 데이터 찾아보기 및 좀 더 정교한 상태 관리에 매우 유용합니다. WebSphere InterChange Server에서 데이터베이스 연결 풀은 WebSphere Process Server에서 표준 JDBC 자원으로 렌더링되며 기본 기능은 동일합니다. 그러나 연결 및 트랜잭션이 관리되는 방식은 서로 다를 수도 있습니다.

나중의 이식성을 최대화하려면 협업 템플릿 또는 맵 안의 Java 스니펫 노드에 걸쳐 데이터베이스 트랜잭션을 활성화하지 마십시오. 예를 들어, 연결 확보, 트랜잭션 시작 및 종료, 연결 릴리스 등과 관련된 코드는 하나의 코드 스니펫으로 되어야 합니다.

비즈니스 오브젝트

비즈니스 오브젝트의 개발에 대한 기본 고려사항은 아티팩트 구성을 위해 제공된 도구만 사용하는 것, 명시적 데이터 유형 사용 및 데이터 속성의 길이 사용, 문서화된 API만 활용 등입니다.

WebSphere Process Server 안의 비즈니스 오브젝트는 SDO(Service Data Object)에 기반하며 SDO는 강력한 유형의 데이터 속성을 사용합니다. WebSphere InterChange Server 및 어댑터의 비즈니스 오브젝트의 경우 데이터 속성은 강력한 유형이 아니며 사용자는 때때로 비문자열 데이터 속성에 대해 문자열 데이터 유형을 지정하기도 합니다. WebSphere Process Server의 문제를 방지하려면 데이터 유형의 스펙에서 명시적이어야 합니다.

WebSphere Process Server 안의 비즈니스 오브젝트는 구성요소 간에 전달될 때 런타임 시 직렬화될 수도 있기 때문에, 데이터 속성에 대해 필요한 길이로 명시하여 시스템 자원의 사용을 최소화하는 것이 중요합니다. 예를 들어, 문자열 속성에 최대 255 문자 길이는 사용하지 마십시오. 또한 현재 기본값이 255 문자인 길이 속성에 0을 지정하지 마십시오. 속성에 필요한 정확한 길이를 지정하십시오.

XSD NCName 규칙은 WebSphere Process Server의 비즈니스 오브젝트 속성 이름에 적용되므로 비즈니스 오브젝트 속성의 이름에 공백이나 ":"를 사용하지 마십시오. 공백 또는 ":"를 갖는 비즈니스 오브젝트 속성 이름은 WebSphere Process Server에서 올바르지 않습니다. 이주하기 전에 비즈니스 오브젝트 속성의 이름을 바꾸십시오.

비즈니스 오브젝트에서 배열을 사용하는 경우 맵 및 관계에서 배열에 색인화할 때 배열의 순서에 의존할 수는 없습니다. 이 경우 WebSphere Process Server로 이주되는 생성은 색인 순서를 보장하지 않으며 특히 항목이 삭제되었을 때 그러합니다.

앞서 설명한 바와 같이 비즈니스 오브젝트 정의를 편집하는데 Business Object Designer 도구만을 사용하고 통합 아티팩트 내부의 비즈니스 오브젝트용으로 출력된 API만을 사용해야 합니다.

협업 템플릿

앞에서 설명했던 여러 가이드라인은 협업 템플릿의 개발에 적용됩니다.

프로세스를 메타데이터로 적절하게 설명하려면 협업 템플릿의 작성 및 수정에 항상 Process Designer 도구를 사용하고 메타데이터 파일을 직접 편집하지 마십시오. 가능하면 활동 편집기 도구에서 메타데이터를 최대한으로 사용하여 필요한 로직을 설명하십시오.

이주에 필요할 수도 있는 수동 재작업을 최소화하려면 협업 템플릿 안의 문서화된 API만 사용하십시오. 정적 변수는 사용하지 마십시오. 대신 비정적 변수 및 협업 특성을 사용하여 비즈니스 로직의 요구사항을 해결하십시오. Java 스니펫에서 Java 규정자 final, transient 및 native를 사용하지 마십시오. 이러한 규정자는 협업 템플릿 이주의 결과인 BPEL Java 스니펫에 강제 적용될 수 없습니다.

나중의 이식성을 최대화하려면 사용자 정의 데이터베이스 연결 풀에 명시적 연결 릴리스 호출 및 명시적 트랜잭션 묶음(예: 명시적 확약 및 명시적 롤백)을 사용하지 마십시오. 대신 컨테이너 관리 암시적 연결 정리 및 암시적 트랜잭션 묶음을 활용하십시오. 또한 협업 템플릿 안의 Java 스니펫 코드에 걸쳐 시스템 연결 및 트랜잭션을 활성화하지 마십시오. 이 사항은 사용자 정의 데이터베이스 연결 풀뿐만 아니라 외부 시스템에 대한 모든 연결에도 적용됩니다. 이미 설명한 바와 같이 외부 EIS를 사용한 오퍼레이션은 어댑터에서 관리되어야 하며 데이터베이스 오퍼레이션과 관련된 코드는 하나의 코드 스니펫 안에 포함되어야만 합니다. 이것은 BPEL 비즈니스 프로세스 구성요소로 렌더링될 때 인터럽트 가능한 플로우로, 선택적으로 배치될 수 있는 협업 안에 필요할 수도 있습니다. 이 경우에 프로세스는 활동 간에 전달되는 상태 및 글로벌 변수 정보만 갖는 몇몇 개별적인 트랜잭션으로 구성될 수 있습니다. 임의의 시스템 연결에 대한 컨텍스트 또는 해당 프로세스 트랜잭션을 확장하는 관련 트랜잭션은 손실됩니다.

협업 템플릿 특성 이름에는 특수 문자를 사용하지 마십시오. 특수 문자는 해당 문자가 이주되는 BPEL 특성 이름에 올바르게 표시되지 않습니다. 이주에 의해 생성되는 BPEL에서 구문 오류를 피하려면 이주하기 전에 특성의 이름을 바꿔 특수 문자를 제거하십시오.

"this"를 사용하는 변수를 참조하지 마십시오. 예를 들어, "this.inputBusObj" 대신 "inputBusObj"를 사용하십시오.

시나리오 범위의 변수 대신 클래스 레벨 범위의 변수를 사용하십시오. 시나리오 범위는 이주 시에 앞으로 진행되지 않습니다.

Java 스니펫에 선언된 모든 변수를 기본값으로 초기화하십시오. 예를 들어, "Object myObject = null;"과 같이 초기화하십시오. 이주 전에 선언되는 동안 모든 변수가 초기화되는지 확인하십시오.

협업 템플릿의 사용자 수정 가능 섹션에 Java import 문이 있는지 확인하십시오. 협업 템플릿의 정의에서 import 필드를 사용하여 Java 패키지를 가져오도록 지정하십시오.

맵

협업 템플릿에 대해 앞에서 설명했던 여러 가이드라인은 맵에도 적용됩니다.

맵이 메타데이터로 적절하게 설명되려면 맵의 작성 및 수정에 항상 Map Designer 도구를 사용하고 메타데이터 파일을 직접 편집하지는 마십시오. 가능하다면 활동 편집기 도구에서 메타데이터를 최대한으로 사용하여 필요한 로직을 설명하십시오.

맵에서 하위 비즈니스 오브젝트를 참조할 때 해당 하위 비즈니스 오브젝트에 대해 하위 맵을 사용하십시오.

Java 코드를 SET에서 "value"로 사용하지 마십시오. 해당 코드는 WebSphere Process Server에 올바르게 읽기 때문입니다. 대신 상수를 사용하십시오. 예를 들어, 설정 값이 "xml version=" + "1.0" + " encoding=" + "UTF-8"인 경우 이 값은 WebSphere Process Server에서 유효하지 않습니다. 이 경우에는 이주하기 전에 해당 값을 "xml version=1.0 encoding=UTF-8"로 변경하십시오.

이주에 필요할 수도 있는 수동 재작업을 최소화하려면 협업 템플릿 안의 문서화된 API만 사용하십시오. 정적 변수는 사용하지 마십시오. 대신 비정적 변수 및 협업 특성을 사용하여 비즈니스 로직의 요구사항을 해결하십시오. Java 스니펫에서 Java 규정자 final, transient 및 native를 사용하지 마십시오.

비즈니스 오브젝트에서 배열을 사용하는 경우 맵에서 배열로 색인화할 때 배열 순서에 의존할 수는 없습니다. 이 경우 WebSphere Process Server로 이주되는 생성은 색인 순서를 보장하지 않으며 특히 항목이 삭제되었을 때 그러합니다.

나중의 이식성을 최대화하려면 사용자 정의 데이터베이스 연결 풀에 명시적 연결 릴리스 호출 및 명시적 트랜잭션 묶음(예: 명시적 확약 및 명시적 롤백)을 사용하지 마십시오. 대신 컨테이너 관리 암시적 연결 정리 및 암시적 트랜잭션 묶음을 활용하십시오. 또한 변환 노드 경계에 걸쳐 Java 스니펫 코드에서 시스템 연결 및 트랜잭션을 활성화하지 마십시오. 이 사항은 사용자 정의 데이터베이스 연결 풀뿐만 아니라 외부 시스템에 대한 모든 연결에도 적용됩니다. 이미 설명한 바와 같이 외부 EIS를 사용한 오퍼레이션은 어댑터에서 관리되어야 하며 데이터베이스 오퍼레이션과 관련된 코드는 하나의 코드 스니펫 안에 포함되어야만 합니다.

관계

관계의 경우 관계 정의는 WebSphere Process Server에서 사용하도록 이주될 수 있으며 관계 테이블 스키마 및 인스턴스 데이터는 WebSphere Process Server에서 재사용될 수 있고 또한 WebSphere InterChange Server 및 WebSphere Process Server 간에 동시에 공유됩니다.

관계에 대한 핵심 고려사항은 관련 구성요소를 구성하도록 제공된 도구만 사용하는 것, 통합 아티팩트 내부의 관계용으로 출력된 API만 사용하는 것입니다.

Relationship Designer 도구만을 사용하여 관계 정의를 편집해야 합니다. 또한 WebSphere InterChange Server만 관계 스키마를 구성하도록 허용하십시오. 이 스키마는 관계 정의 배치 시 자동으로 생성됩니다. 데이터베이스 도구 또는 SQL 스크립트를 사용하여 관계 테이블 스키마를 직접 변경하지 마십시오.

또한 관계 테이블 스키마 안의 관계 인스턴스 데이터를 직접 수정하려면 Relationship Designer에서 제공하는 기능을 사용하십시오.

앞서 설명한 바와 같이 통합 아티팩트 안의 관계용으로 출력된 API만 사용해야 합니다.

액세스 프레임워크 클라이언트

CORBA IDL 인터페이스 API를 채택하는 새 클라이언트를 개발하지 마십시오. 이 클라이언트는 WebSphere Process Server에서 지원되지 않습니다.

WebSphere MQ Workflow에서 WebSphere Integration Developer로 이주

WebSphere Integration Developer는 WebSphere MQ Workflow에서 이주하는 데 필요한 도구를 제공합니다.

이주 마법사를 사용하면 WebSphere MQ Workflow의 buildtime 컴포넌트에서 내보낸 비즈니스 프로세스의 FDL 정의를 Business Process Choreographer의 해당 아티팩트로 변환할 수 있습니다. 생성된 Business Process Choreographer 아티팩트는 비즈니스 오브젝트에 대한 XMLSchema 정의, WSDL 정의, BPEL 및 TEL 정의로 구성됩니다.

변환 도구에는 **export deep** 옵션을 사용하여 WebSphere MQ Workflow Buildtime에서 내보낸 문법적으로 완벽한 프로세스 모델의 FDL 정의가 필요합니다. 이 옵션에는 모든 필요한 데이터, 프로그램 및 서브프로세스 스펙이 포함되어야 합니다. 또한 WebSphere MQ Workflow Buildtime에서 FDL을 내보낼 때 WebSphere MQ Workflow 프로세스 모델에서 참조하는 사용자 정의 프로세스 실행 서버 정의(UPES)를 선택했는지 확인하십시오.

주: 이주 마법사는 다음의 이주를 처리하지 않습니다.

- WebSphere MQ Workflow 런타임 인스턴스
- WebSphere MQ Workflow 프로그램 실행 에이전트(PEA) 또는 WebSphere MQ Workflow 프로세스 실행 서버(z/OS®용 PES)에서 호출한 프로그램 응용프로그램

WebSphere MQ Workflow에서의 이주 준비

WebSphere MQ Workflow에서 WebSphere Integration Developer로 이주하려면 먼저 환경을 올바르게 준비했는지 확인해야 합니다.

맵의 범위와 완성도는 이주 시 다음 가이드라인을 어느 정도 준수했는지에 따라 달라집니다.

- 순수한 스텝 활동이 아닌 경우 FDL 프로그램 활동이 사용자 정의 프로세스 실행 서버(UPES)와 연결되는지 확인하십시오.
- WebSphere MQ Workflow 프로그램 활동에 대한 스텝 지정이 TEL 기본값 스텝 **verb**를 준수하는지 확인하십시오.
- 짧고 간단한 이름을 사용하여 이주된 프로세스 모델을 읽기 쉽도록 하십시오. FDL 이름은 규칙에 위배되는 BPEL 이름이 될 수도 있습니다. 이주 마법사에서 FDL 이름을 유효한 BPEL 이름으로 자동 변환하도록 도와줍니다.

이주 마법사는 이주가 불가능하여 수작업으로 실행 가능한 비즈니스 프로세스 편집기 아티팩트를 선택해야 하는 WebSphere MQ Workflow 생성(PEA 또는 PES 프로그램 활동, 일부 동적인 스텝 지정 등)에 대해서도 문법적으로 올바른 비즈니스 프로세스 편집기 생성을 작성합니다.

다음 표에서는 적용된 맵핑 규칙에 대해 대략적으로 설명합니다.

표 2. 맵핑 규칙

WebSphere MQ Workflow	BPC(Business Process Choreographer)
프로세스	실행 모드가 <i>longRunning</i> 인 프로세스; 프로세스의 인바운드 및 아웃바운드 인터페이스에 대한 파트너 링크
소스 및 싱크	프로세스 입력 및 프로세스 출력에 대한 변수, <i>Receive</i> 활동 및 <i>reply</i> 활동
프로그램 활동	호출 활동
프로세스 활동	호출 활동
비어 있음 활동	비어 있음 활동
블록	BPEL 활동이 임베드된 범위
활동의 종료 조건	<i>While</i> 활동(실제 활동을 엔클로징)
활동의 시작 조건	활동의 조인 조건
활동의 스텝 지정	개인 <i>타스</i> 활동
활동의 입력 컨테이너 및 출력 컨테이너	호출 활동의 입/출력(I/O)을 지정하는 데 사용된 변수
제어 커넥터; 변환 조건	링크; 변환 조건
데이터 커넥터	지정 활동
글로벌 데이터 컨테이너	변수

소형 프로젝트에 대한 이주 프로세스는 가능하면 초기에 시도하는 것이 좋습니다. 이주 마법사는 WebSphere MQ Workflow 프로세스 모델에서 비즈니스 프로세스 편집기 프로세스 모델로의 변환을 단순화하지만, 프로세스는 새 프로그래밍 모델을 작성하면서 일대일로 맵핑될 수 없음에 주의해야 합니다. 기본 프로세스 스펙 언어

(FDL 및 BPEL)의 시맨틱 범위는 중복되는 영역도 있지만, 전체적으로 겹쳐지지는 않습니다. 이런 점들을 숙지하지 못하고 있다면, 비즈니스 프로세스 편집기의 새로운 장점들을 제대로 활용할 수 없을 것입니다. 웹 서비스는 문제있는 솔루션을 새 솔루션으로 대체하도록 요청하는 새 기술에 대한 약속을 표시합니다.

일반적으로 항상 생성된 아티팩트를 검토하고 수정해야 합니다. 이주를 시도하거나 이주 작업을 완료하도록 하려는 추가 노력이 필요할 수 있습니다.

이주 마법사를 사용한 WebSphere MQ Workflow 이주


이주 마법사를 사용하면 WebSphere MQ Workflow의 buildtime 컴포넌트에서 내보낸 비즈니스 프로세스의 FDL 정의를 Business Process Choreographer의 해당 아티팩트로 변환할 수 있습니다. 생성된 Business Process Choreographer 아티팩트는 비즈니스 오브젝트에 대한 XMLSchema 정의, WSDL 정의, BPEL 및 TEL 정의로 구성됩니다.

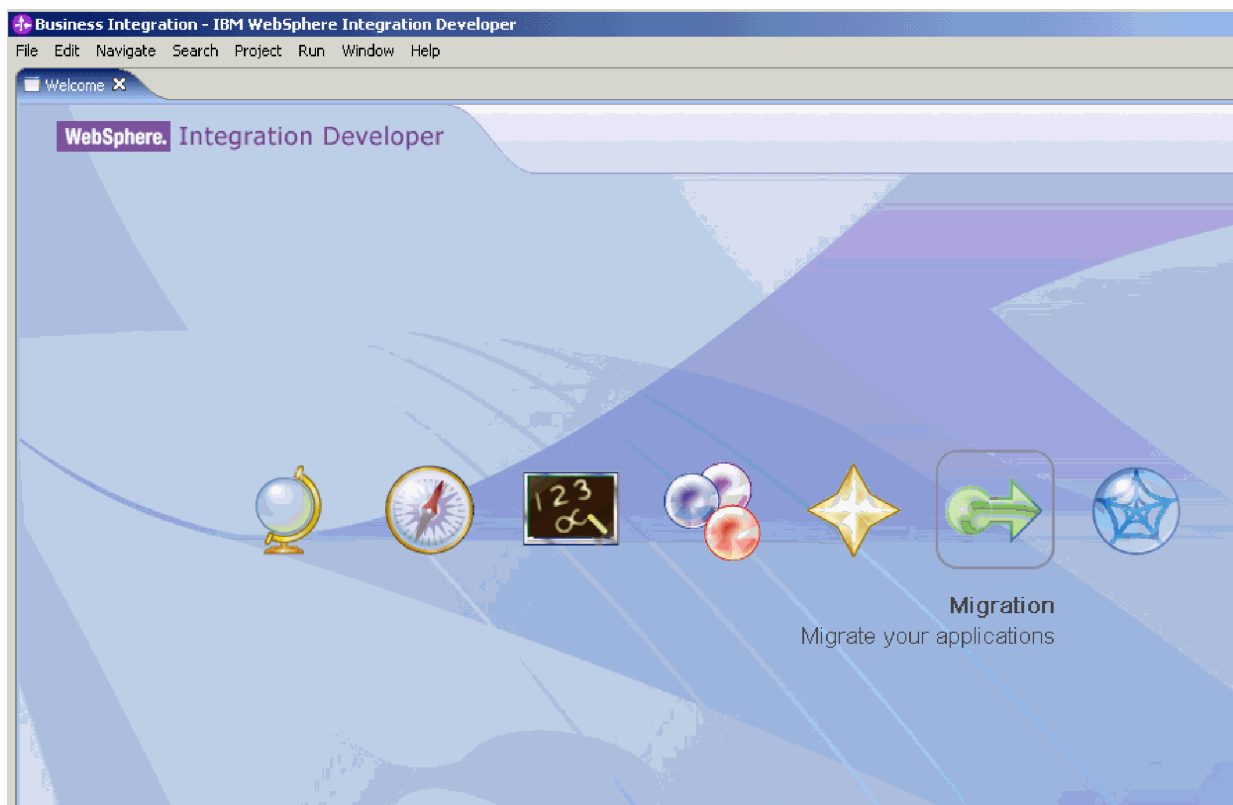
주: 이주 마법사는 다음의 이주를 처리하지 않습니다.

- WebSphere MQ Workflow 런타임 인스턴스
- WebSphere MQ Workflow 프로그램 실행 에이전트(PEA) 또는 WebSphere MQ Workflow 프로그램 실행 서버(z/OS용 PES)에서 호출한 프로그램 응용프로그램

이주 마법사를 사용하여 다음 단계에 따라 WebSphere MQ Workflow 아티팩트를 이주하십시오.



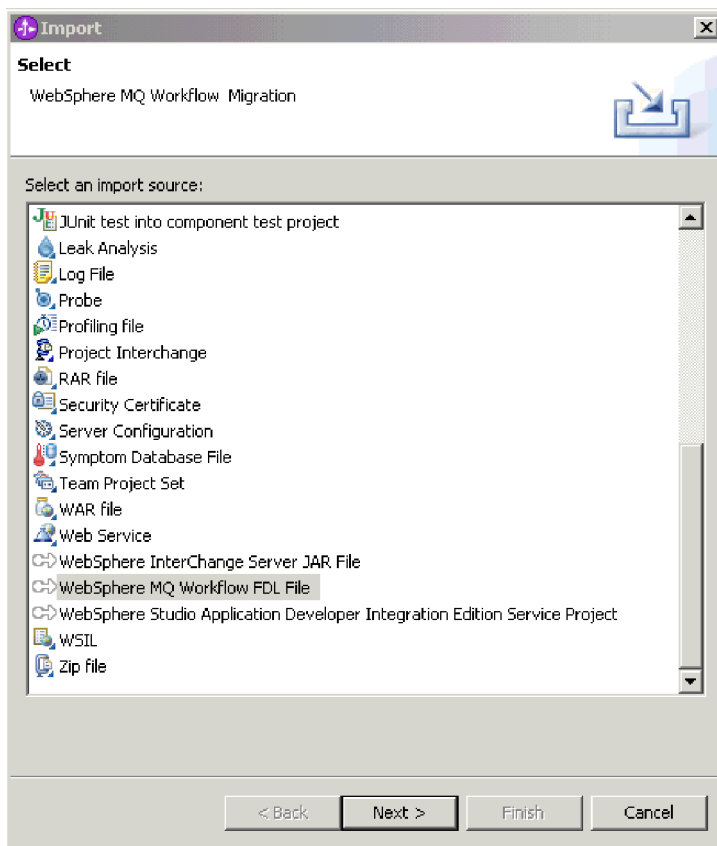
1. 환영 페이지에서 를 클릭하여 이주 페이지를 여십시오.



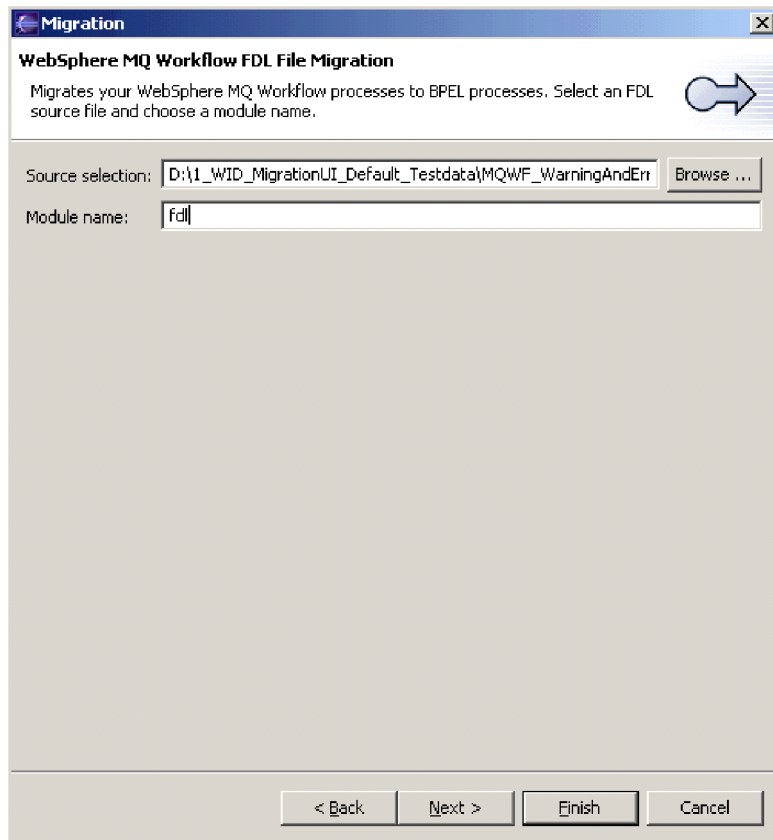
- 이주 페이지에서 WebSphere MQ Workflow 프로세스 이주 옵션을 선택하십시오.



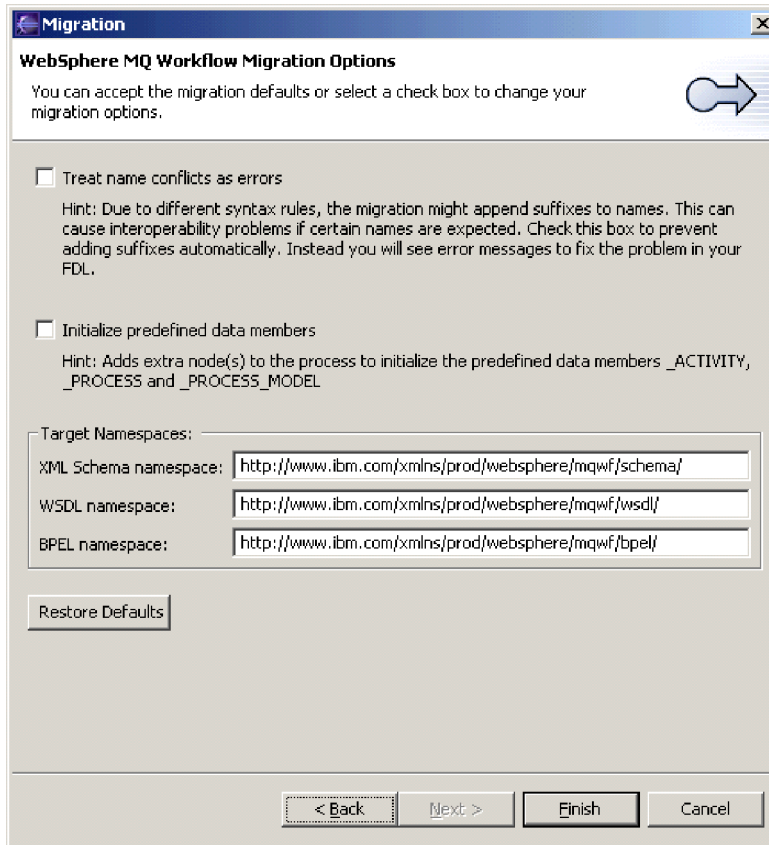
(참고: 파일 → 가져오기 → **WebSphere MQ Workflow FDL** 파일을 클릭하여 이주 마법사를 열 수도 있습니다.



- 이주 마법사가 열립니다. 찾아보기 단추를 클릭하고 파일을 탐색하여 소스 선택 필드에 .fdl 파일의 이름을 입력하십시오. 관련 필드에 모듈 이름을 입력하십시오. 다음을 클릭하십시오.



- 이주 옵션 페이지가 열립니다. 이 페이지에서 이주 기본값을 승인하거나 옵션을 변경하려면 선택란을 선택하십시오. 이름 충돌을 오류로 처리를 선택하면, 상호운용성 오류가 발생할 수 있는 접미부 자동 추가를 방지할 수 있습니다. 사전정의된 데이터 구성원 초기화 선택란은 프로세스에 노드를 추가하여 사전정의된 데이터 구성원을 초기화합니다.



Migration

WebSphere MQ Workflow Migration Options

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

☐ Treat name conflicts as errors

Hint: Due to different syntax rules, the migration might append suffixes to names. This can cause interoperability problems if certain names are expected. Check this box to prevent adding suffixes automatically. Instead you will see error messages to fix the problem in your FDL.

☐ Initialize predefined data members

Hint: Adds extra node(s) to the process to initialize the predefined data members _ACTIVITY, _PROCESS and _PROCESS_MODEL.

Target Namespaces:

XML Schema namespace:

WSDL namespace:

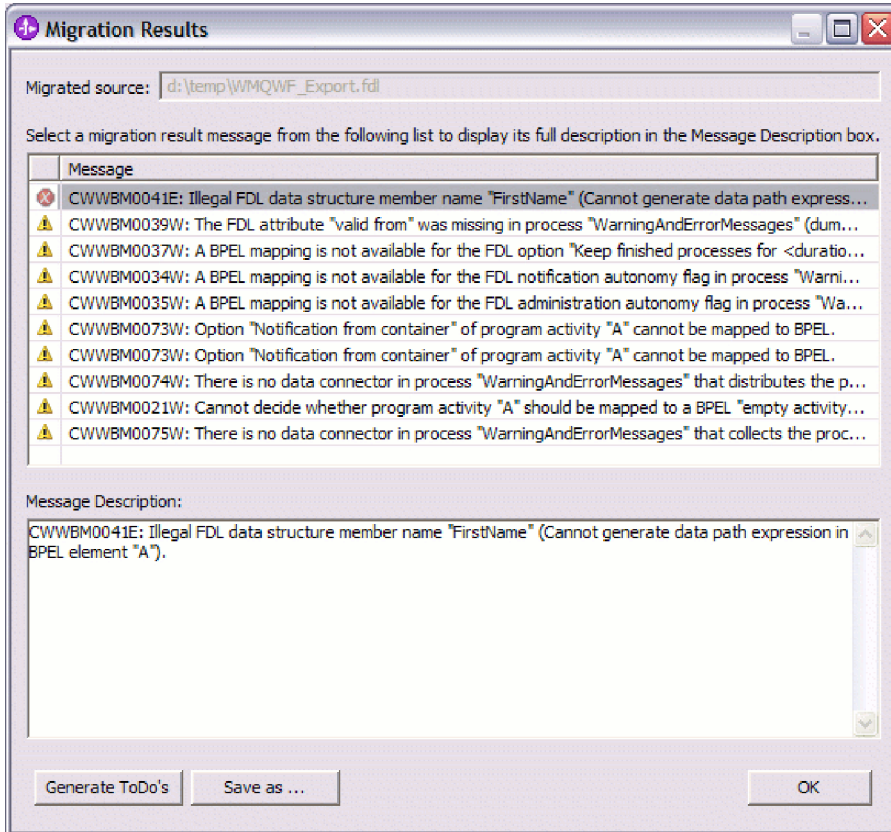
BPEL namespace:

완료를 클릭하십시오.

WebSphere MQ Workflow 이주 확인

이주 마법사가 완료되면 오류, 경고 및/또는 정보 메시지로 구성된 목록이 표시됩니다. 이 메시지를 통해 WebSphere MQ Workflow 이주를 확인할 수 있습니다.

이주 마법사가 완료되었을 때 다음 페이지가 표시됩니다.



이주 결과 창에서 이주 메시지 목록을 볼 수 있습니다. 메시지를 클릭하여 세부사항의 전체 설명을 보십시오. 필요하다면 이 메시지 목록에서, 수행할 작업 생성 단추를 클릭하여 수정 항목의 목록을 작성할 수 있습니다.

WebSphere MQ WorkFlow에서의 이주 프로세스 제한사항

WebSphere MQ Workflow 이주 프로세스에는 일정의 제한사항이 있습니다.

FDL의 이주는 UPES 활동 및 해당 WSDL에 대한 호출 활동을 생성합니다. 그러나 호출 메시지와 해당 응답을 상관시키는 데 사용되는 기법에서 런타임 환경은 IBM® WebSphere MQ Workflow와 IBM WebSphere Process Server 사이에 크게 다릅니다. 따라서 UPES 호환성 계층을 사용할 수 없는 경우 생성된 BPEL을 실행할 수 없습니다.

소스 아티팩트를 WebSphere Studio Application Developer Integration Edition에서 WebSphere Integration Developer로 이주

소스 아티팩트는 WebSphere Studio Application Developer Integration Edition에서 WebSphere Integration Developer로 이주할 수 있습니다. 응용프로그램에서의 소스 아티팩트 이주에는 새 기능/피처를 사용할 수 있도록 새 WebSphere Integration Developer 프로그래밍 모델로의 이주가 포함됩니다. 그런 다음 응용프로그램을 WebSphere Integration Developer 버전 6.0 서버에 다시 배치하고 설치할 수 있습니다.

WebSphere Business Integration Server Foundation 5.1에서 사용 가능한 많은 기능이 기본 WebSphere Application Server 6.0으로 이동되었습니다. 해당 기능의 이주에 대한 팁은 다음 사이트를 참조하십시오: http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.websphere.base.doc/info/aes/aerins_migratepme.html

WebSphere Studio Application Developer Integration Edition 서비스 프로젝트 전체를 이주할 경우 다음과 같은 두 개의 기본 작업을 완료해야 합니다.

1. 이주 마법사를 사용하여 자동으로 비즈니스 통합 모듈 프로젝트로 아티팩트를 이주하십시오.
2. WebSphere Integration Developer를 사용하여 수동으로 이주를 완료하십시오. 여기에는 자동으로 이주할 수 없는 모든 Java 코드 수정 및 이주된 아티팩트 재연결이 포함됩니다.

주: 런타임 이주(업그레이드 경로)는 WebSphere Process Server 6.0.0에서 제공되지 않으므로 이 소스 아티팩트 이주 경로는 6.0.0에서 WebSphere Studio Integration Edition 서비스 프로젝트 이주를 위한 유일한 옵션입니다.

소스 아티팩트 이주 시 지원되는 이주 경로

WebSphere Studio Application Developer Integration Edition에서 소스 아티팩트 이주를 시작하기 전에, WebSphere Integration Developer가 지원하는 지원 이주 경로를 검토해야 합니다.

이주 마법사는 한 번에 하나의 WebSphere Studio Application Developer Integration Edition 버전 5.1(이상) 서비스 프로젝트를 이주하는 기능을 제공합니다. 전체 작업공간을 이주하지 않습니다.

이주 마법사에서 응용프로그램 2진을 이주하지 않습니다. WebSphere Studio Application Developer Integration Edition 서비스 프로젝트에서 발견되는 소스 아티팩트만을 이주합니다.

이주 시 소스 아티팩트 준비

WebSphere Studio Application Developer Integration Edition에서 WebSphere Integration Developer로 소스 아티팩트를 이주하려면 먼저 환경을 올바르게 준비했는지 확인해야 합니다.

다음 단계에서는 소스 아티팩트를 WebSphere Studio Application Developer Integration Edition에서 WebSphere Integration Developer로 이주하기 전에 환경을 준비하는 방법을 설명합니다.

1. 이주를 시도하기 전에 전체 5.1 작업공간의 백업 사본이 있는지 확인하십시오.
2. Rational® Application Developer Information Center의 이주 섹션(<http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.rad.migration.doc/topics/tmigratefrom51x.html>)을 검토하여 작업공간의 WBI에 특정하지 않은 프로젝트를 이주하는 최상의 방법을 판별하십시오.
3. Rational Application Developer가 제공하는 웹 서비스 기능에 대한 백그라운드 정보에 대해 Rational Application Developer Information Center의 웹 서비스 섹션(<http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.webservices.rad.nav.doc/developingweb.html>)을 검토하십시오.
4. 적합한 모든 WebSphere Integration Developer 기능이 사용되는지 확인하십시오. 이들 기능이 사용되지 않는 경우 아래에서 설명하는 메뉴 옵션이 표시되지 않을 수 있습니다. 중요한 기능을 사용하려면 다음을 수행하십시오.

- WebSphere Integration Developer에서 창 메뉴 항목으로 이동하여 환경 설정을 선택하십시오.
- **Workbench**로 이동한 후 기능 카테고리를 선택하십시오.
- 다음 카테고리 아래의 모든 기능을 선택하십시오.
 - 고급 J2EE
 - Enterprise Java
 - 통합 개발자
 - Java 개발자
 - 웹 개발자(일반)
 - 웹 서비스 개발자
 - XML 개발자
- 확인을 클릭하십시오.

5. WebSphere Integration Developer의 새 작업공간 디렉토리를 사용하십시오. 먼저 WebSphere Integration Developer에서 읽을 수 있는 형식으로 이주해야 하므로 서비스 프로젝트를 포함하는 이전의 WebSphere Studio Application Developer Integration Edition 작업공간에서 WebSphere Integration Developer를 여는 것은 권장되지 않습니다. 이 경우 다음 단계를 수행하도록 권장합니다.

- 이전 작업공간에서 새 작업공간으로 모든 비서비스 프로젝트를 복사하십시오. 배치 코드가 5.1 서비스 프로젝트에 대해 생성될 때 작성된 5.1 EJB, 웹 및 EAR 프로젝트를 복사하지 마십시오. BI 모듈이 빌드될 때 새 6.0 배치 코드가 자동으로 다시 생성됩니다.
- 공백의 작업공간에서 WebSphere Integration Developer를 열고 파일 → 가져오기 → 작업공간으로 기존 프로젝트를 클릭하여 모든 비서비스 프로젝트를 가져온 다음 새 작업공간으로 복사된 프로젝트를 선택하십시오.
 - 프로젝트가 J2EE 프로젝트인 경우, Rational Application Developer 이주 마법사를 사용하여 1.4 레벨로 이주해야 합니다.
 - 1) 프로젝트를 마우스 오른쪽 단추로 클릭하고 이주 → **J2EE** 이주 마법사...를 선택하십시오.
 - 2) 첫 번째 페이지의 경고문을 검토하고 다음을 클릭하십시오.
 - 3) 프로젝트 목록에서 **J2EE** 프로젝트가 선택되었는지 확인하십시오. 프로젝트 구조 이주 및 **J2EE** 스펙 레벨 이주를 선택된 채로 두십시오. J2EE 버전 **1.4** 및 대상 서버 **WebSphere Process Server v6.0**을 선택하십시오.
 - 4) J2EE 프로젝트에 적합할 수 있는 다른 모든 옵션을 선택하고 완료를 클릭하십시오. 이 단계가 성공적으로 완료되는 경우 이주가 완료되었습니다라는 메시지가 표시됩니다.
 - 5) 이주 후에 J2EE 프로젝트에 오류가 있는 경우 v5 .jar 파일이나 라이브러리를 참조하는 모든 클래스 경로 항목을 제거하고 **JRE** 시스템 라이브러리 및 **WPS** 서버 대상 라이브러리를 대신 클래스 경로에 추가해야 합니다(아래에서 설명). 그러면 오류의 대부분(전체가 아닌)이 해결됩니다.
 - CMM(Extended Messaging) 또는 CMP/A(Container Managed Persistence over Anything)를 갖는 WebSphere Business Integration EJB 프로젝트의 경우, 5.1 프로젝트가 6.0 작업공간으로 이

주된 후에 IBM EJB Jar Extension 설명 파일이 이주되어야 합니다. 자세한 정보는 "WebSphere Business Integration EJB 프로젝트 이주"를 참조하십시오.

- 작업공간으로 가져온 각 비서비스 프로젝트의 클래스 경로를 수정하십시오. 클래스 경로에 JRE 및 WebSphere Process Server 라이브러리를 추가하려면 가져온 프로젝트에서 마우스 오른쪽 단추를 클릭하여 특성을 선택하십시오. **Java** 빌드 경로 항목으로 이동하여 라이브러리 탭을 선택하십시오. 그런 다음 다음을 수행하십시오.

1) 라이브러리 추가 → **JRE** 시스템 라이브러리 → 대체 **JRE - WPS Server v6.0 JRE** → 완료를 선택하십시오.

2) 그런 다음 라이브러리 추가 → **WPS** 서버 대상 → **WPS** 서버 클래스 경로 구성 → 완료를 선택하십시오.

6. 최상의 결과를 위해 이주 마법사를 실행하기 전에 프로젝트 메뉴 항목으로 이동하고 자동으로 빌드가 선택되지 않았는지 확인하십시오. WebSphere Integration Developer는 서비스 배치 옵션이 디자인 타임에 지정된다는 점에서 WebSphere Studio Application Developer Integration Edition과 다릅니다. 프로젝트를 빌드할 때, 배치 코드는 생성된 EJB 및 웹 프로젝트에서 자동으로 갱신되므로 더 이상 수동으로 배치 코드 생성 옵션은 제공되지 않습니다.

7. 서비스 프로젝트 내에서 .bpel 파일을 모두 이주하려면, .bpel 파일이 참조하는 모든 .wsdl 및 .xsd 파일이 새 작업공간에 있는 비즈니스 통합 프로젝트에 있는지 확인해야 합니다.

- .wsdl 및/또는 .xsd 파일이 .bpel 파일과 동일한 서비스 프로젝트에 있는 경우 추가 조치는 필요하지 않습니다.

- .wsdl 및/또는 .xsd 파일이 이주 중인 것과는 다른 서비스 프로젝트에 있는 경우 5.1 아티팩트가 이주 전에 WebSphere Studio Application Developer Integration Edition을 사용하여 재구성되어야 합니다. 그 이유는 비즈니스 통합 모듈 프로젝트가 아티팩트를 공유할 수 없다는 점입니다. 다음은 5.1 아티팩트 재구성을 위한 두 가지 옵션입니다.

- WebSphere Studio Application Developer Integration Edition에서, 모든 공통 아티팩트를 보유할 새 Java 프로젝트를 작성하십시오. 둘 이상의 서비스 프로젝트가 공유하는 모든 .wsdl 및 .xsd 파일을 이 새로운 Java 프로젝트에 넣으십시오. 이 새로운 Java 프로젝트의 종속성을 이들 공통 아티팩트를 사용하는 모든 서비스 프로젝트에 추가하십시오. WebSphere Integration Developer에서, 서비스 프로젝트 중 하나를 이주하기 전에 5.1 공유 Java 프로젝트와 동일한 이름을 갖는 새 비즈니스 통합 라이브러리 프로젝트를 작성하십시오. 5.1 공유 Java 프로젝트에서 이 새로운 BI 라이브러리 프로젝트 폴더로 이전 .wsdl 및 .xsd 파일을 수동으로 복사하십시오. BPEL 서비스 프로젝트를 이주하기 전에 수행해야 합니다.

- 다른 옵션은 이러한 공유 .wsdl 및 .xsd 아티팩트의 로컬 사본을 각 서비스 프로젝트에 보존하여 서비스 프로젝트 사이에 종속성이 없게 하는 것입니다.

- .wsdl 및/또는 .xsd 파일이 기타 프로젝트 유형(일반적으로 기타 Java 프로젝트)에 있을 경우, 5.1 프로젝트와 이름이 동일한 Business Integration Library 프로젝트를 작성해야 합니다. 또한 5.1 Java 프로젝트에서 항목을 추가하여(존재하는 경우) 새 라이브러리 프로젝트의 클래스 경로를 설정해야 합니다. 이 유형의 프로젝트는 공유 아티팩트 저장 시 유용합니다.

이제 이주 프로세스를 시작할 준비가 끝났습니다.

WebSphere Integration Developer 이주 마법사를 사용한 서비스 프로젝트 이주

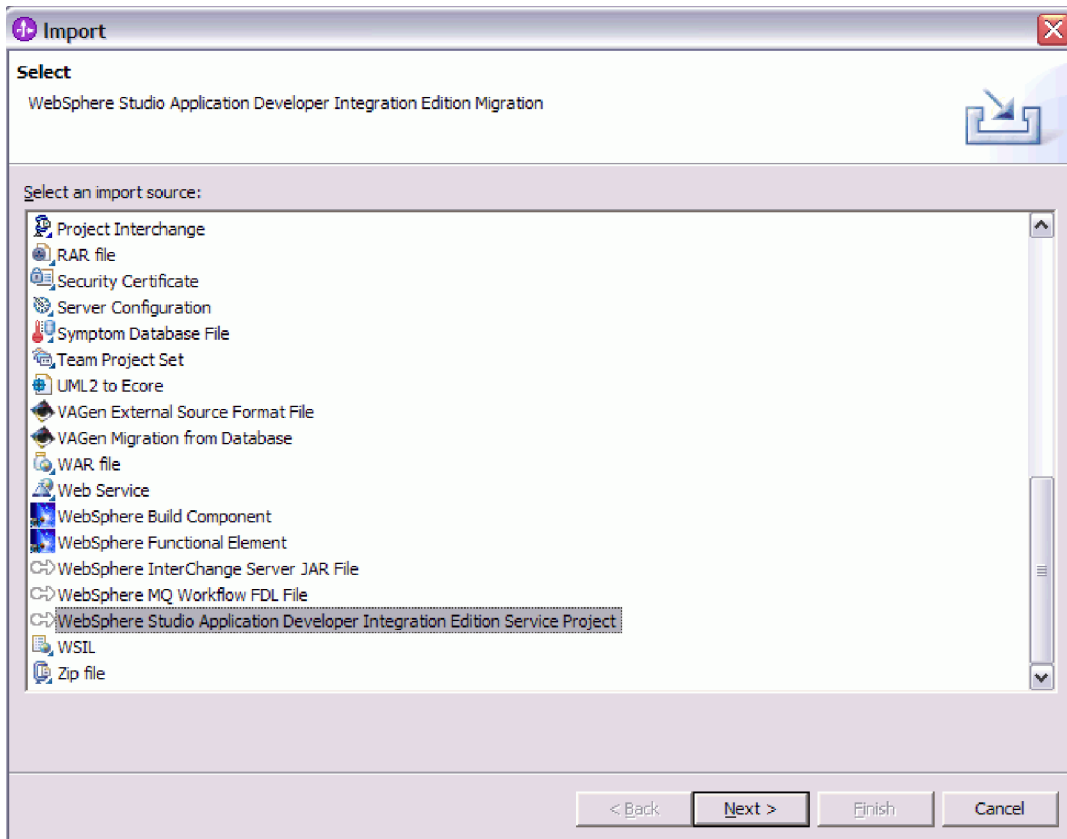
WebSphere Integration Developer 이주 마법사를 사용하면 서비스 프로젝트를 이주할 수 있습니다.

이주 마법사는 다음을 수행합니다.

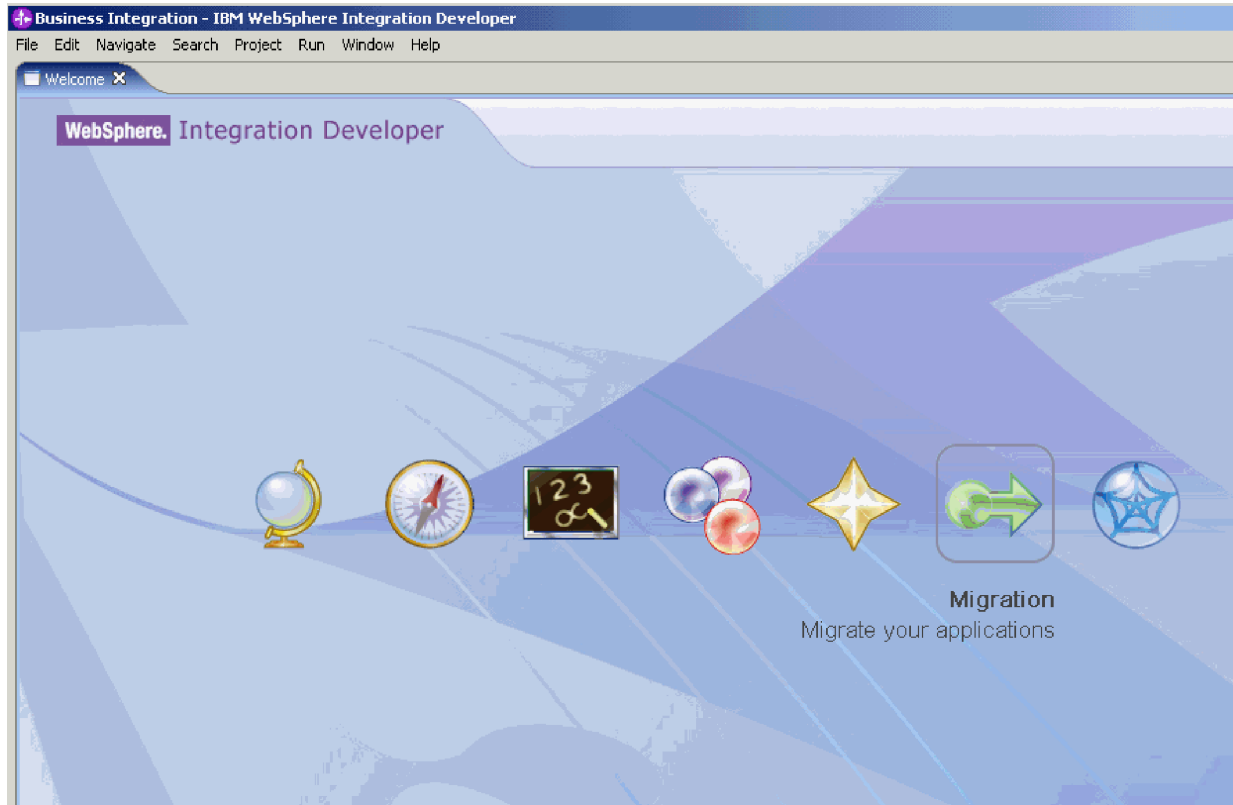
1. 새 비즈니스 통합 모듈 작성(모듈 이름은 사용자에 의해 정의됩니다.)
2. 서비스 프로젝트의 클래스 경로 항목을 새 모듈로 이주
3. 선택한 소스 프로젝트의 모든 WebSphere Business Integration Server Foundation 소스 아티팩트를 이 모듈로 복사
4. WSDL 파일에서 BPEL 확장 이주
5. 비즈니스 프로세스(.bpel 파일)를 BPEL4WS 버전 1.1 곧 출시될 WS-BPEL 버전 2.0 스펙의 주요 기능을 갖는 BPEL4WS 버전 1.1에서 빌드되는 WebSphere Process Server가 지원하는 새 레벨로 이주
6. 각 .bpel 프로세스에서 SCA 컴포넌트 작성
7. 필요한 경우 WebSphere Studio Application Developer Integration Edition에서 기본 모니터링 동작을 보존하도록 각 BPEL 프로세스에서 모니터링 .mon 파일 생성

이주 마법사를 사용하여 다음 단계에 따라 WebSphere Integration Developer 서비스 프로젝트를 이주하십시오.

1. 파일 → 가져오기 → **WebSphere Studio Application Developer Integration Edition** 서비스 프로젝트를 선택하여 마법사를 호출하십시오.



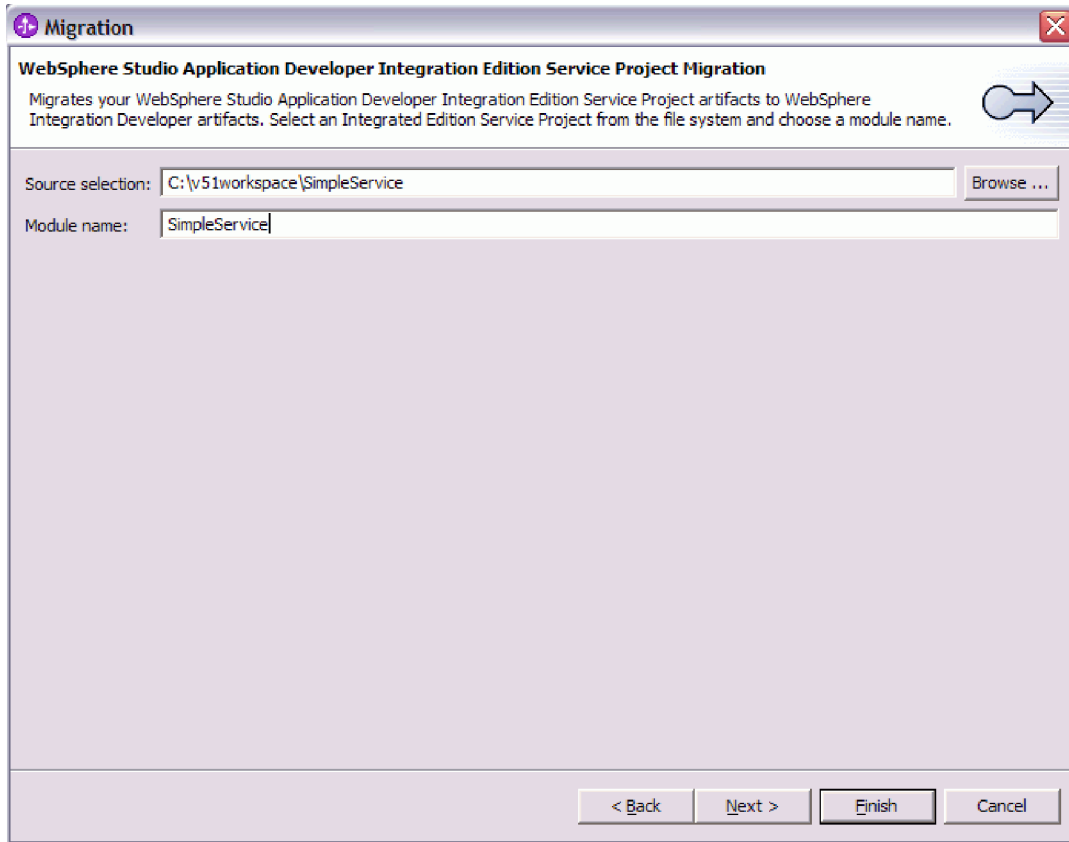
를 클릭하여 환영 페이지에서 이주 마법사를 열 수도 있습니다.



이주 페이지에서 Integration Edition 5.1 서비스 프로젝트 이주 옵션을 선택하십시오.

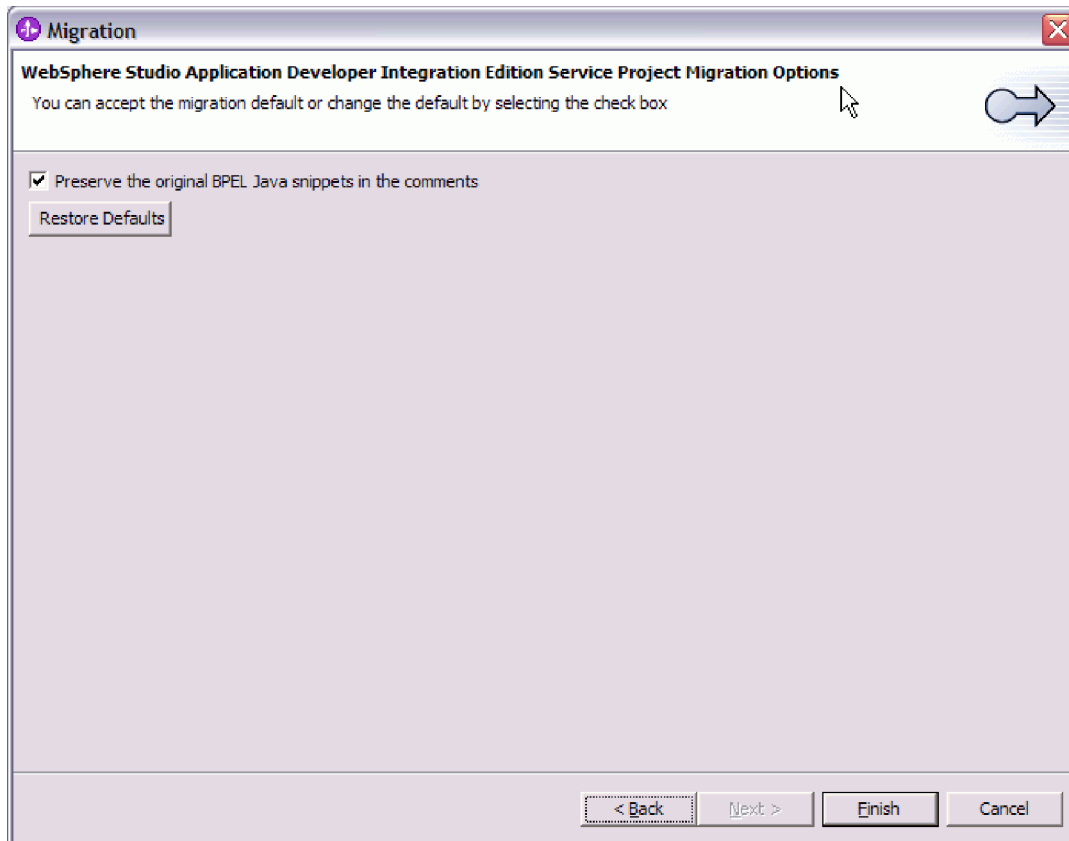


- 이주 마법사가 열립니다. 소스 선택에 대한 경로를 입력하거나 찾아보기 단추를 클릭하여 찾으십시오. 또한 이주할 WebSphere Studio Application Developer Integration Edition 서비스 프로젝트 위치의 모듈 이름을 입력하십시오.



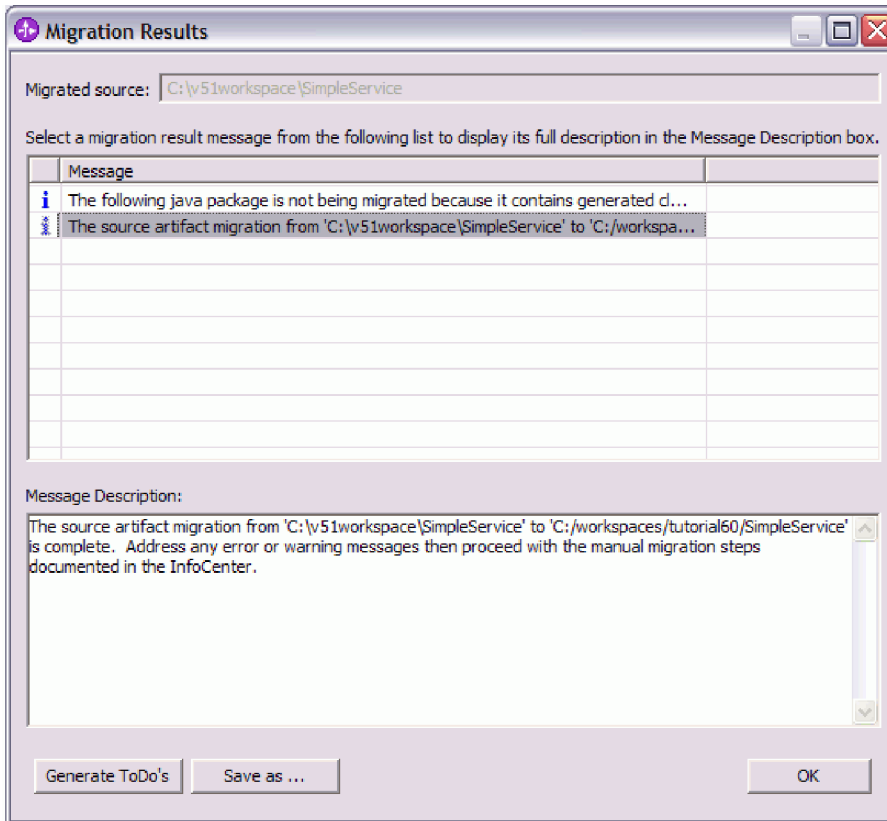
참고: 이 프로젝트에 종속되는 WebSphere Studio Application Developer Integration Edition 작업공간에 다른 프로젝트가 있는 경우 WebSphere Integration Developer로 프로젝트를 가져온 후 종속 프로젝트의 클래스 경로를 갱신할 필요가 없습니다. 모듈 이름으로 서비스 프로젝트의 이름을 선택하도록 권장합니다.

3. 이주 옵션에서, 원래 BPEL Java 스니펫을 설명에 보존 선택란을 선택하십시오.



완료를 클릭하십시오.

- 이주 프로세스가 완료되면 이주 결과 창이 열립니다.



이들 이주 메시지를 포함하는 로그 파일은 6.0 작업공간의 .metadata 폴더에 자동으로 생성됩니다. 로그 파일은 ".log"로 이름 지정됩니다.

이주 마법사를 완료한 후에는 작성된 Business Integration 모듈을 빌드하고 빌드 오류를 해결하십시오. 이주된 모든 .bpel 파일을 검사하십시오. 해당 파일을 모두 이주하여 WebSphere Integration Developer BPEL 편집기에서 열 수 있어야 합니다. 그러나 자동으로 이주할 수 없는 몇 가지 BPEL Java 스니펫도 있습니다. BPEL Java 스니펫에 오류가 표시되는 경우 오류 수정에 필요한 단계에 대해 "SCA 프로그래밍 모델로 이주"를 참조하십시오. 이주 마법사를 사용하여 서비스 프로젝트를 BI 모듈로 이주하는 경우, 모듈 종속성 편집기를 열어 종속성이 올바르게 설정되었는지 확인하십시오. 확인하려면 Business Integration Perspective로 전환하여 BI 모듈 프로젝트를 두 번 클릭하십시오. 여기에서 비즈니스 통합 라이브러리 프로젝트, Java 프로젝트 및 J2EE 프로젝트에 대한 종속성을 추가할 수 있습니다.

수동으로 응용프로그램 이주

이주 마법사에서 아티팩트를 새 비즈니스 통합 모듈로 이주한 후에는 SCA 모델을 준수하는 응용프로그램을 작성하도록 아티팩트를 함께 연결해야 합니다.

1. WebSphere Integration Developer를 열어 비즈니스 통합 Perspective로 전환하십시오. 이주 마법사에서 작성한 모듈(이주한 각 서비스 프로젝트에 대해 하나의 모듈)을 확인해야 합니다. 모듈 프로젝트에 나열된 첫 번째 아티팩트는 모듈의 어셈블리 파일(모듈과 이름이 동일함)입니다.
2. 어셈블리 파일을 두 번 클릭하여 어셈블리 편집기에서 여십시오. 이 편집기에서는 SCA 컴포넌트를 작성한 후 함께 연결하여 버전 5.1 응용프로그램과 유사한 기능을 얻을 수 있습니다. WebSphere Studio Application

Developer Integration Edition 서비스 프로젝트에 BPEL 프로세스가 있는 경우 이주 마법사는 해당 프로세스마다 기본 SCA 컴포넌트를 작성해야 합니다. 이때 컴포넌트는 어셈블리 편집기에 있습니다.

3. 컴포넌트를 선택하고 설명, 세부사항 및 구현 특성이 표시되고 편집 가능한 특성 보기로 이동하십시오.

다음 정보는 WebSphere Integration Developer에서 사용 가능한 도구를 사용하여 응용프로그램을 수동으로 재연결하는 방법에 대해 자세히 설명합니다.

재연결할 응용프로그램에 대해 서비스에 대한 SCA 컴포넌트 및 SCA 가져오기 작성:

모든 프로젝트는 5.1에서 사용되었던 방법으로 서비스를 다시 연결하기 위해 이주 후에 일부 재연결이 필요합니다. 예를 들어 이주된 모든 비즈니스 프로세스는 해당 비즈니스 파트너에 다시 연결해야 합니다. SCA 컴포넌트 또는 가져오기가 다른 모든 서비스 유형에 대해 작성되어야 합니다. 프로젝트 외부에 있는 시스템 또는 엔티티와 상호작용하는 WebSphere Studio Application Developer Integration Edition 서비스 프로젝트의 경우 이주된 프로젝트가 SCA 모델에 따라 엔티티를 서비스로 액세스하기 위해 SCA 가져오기를 작성할 수 있습니다.

프로젝트 내의 엔티티와 상호작용하는 WebSphere Studio Application Developer Integration Edition 서비스 프로젝트의 경우(예: 비즈니스 프로세스, 변환기 서비스 또는 Java 클래스), 이주된 프로젝트가 SCA 모델에 따라 엔티티를 서비스로 액세스하기 위해 SCA 가져오기를 작성할 수 있습니다.

다음 섹션은 이주되어야 하는 서비스의 유형에 따라 작성할 SCA 가져오기 또는 SCA 컴포넌트에 대한 세부사항을 제공합니다.

Java 서비스 이주:

Java 서비스를 SCA Java 컴포넌트로 이주할 수 있습니다.

WebSphere Studio Application Developer Integration Edition 서비스 프로젝트가 다른 Java 프로젝트에 종속된 경우, 기존 프로젝트를 새 작업공간 디렉토리에 복사하고 파일 → 가져오기 → 작업공간으로 기존 프로젝트 마법사를 사용하여 WebSphere Integration Developer로 가져오십시오.

WebSphere Studio Application Developer Integration Edition에서 새 Java 서비스를 기존 Java 클래스에서 생성한 경우 다음 옵션이 제공됩니다.

- 복합 데이터 유형의 XSD 스키마 작성
 - 인터페이스 WSDL 파일에서 작성
 - 각 데이터 유형의 새 파일로 작성
- 오류 처리 기능 지원
 - 결함 생성
 - 결함 생성하지 않음
- 생성할 서비스에 대한 기타 세부사항(예: 바인딩 및 서비스 이름)

6.0에는 데이터 맵핑, 인터페이스 중개, 비즈니스 상태 머신, 선택기, 비즈니스 규칙 등과 같은 새 기능을 제공하는 많은 새 컴포넌트가 있습니다. 먼저 이러한 새 컴포넌트 중 어느 것이 사용자 정의 Java 컴포넌트를 대체할 수 있는지 판별해야 합니다. 판별할 수 없는 경우 아래에서 설명하는 이주 경로를 따르십시오.

이주 마법사를 사용하여 서비스 프로젝트를 가져오십시오. 그러면 WSDL 메시지, PortTypes, 바인딩 및 서비스가 WebSphere Studio Application Developer Integration Edition에 생성되는 비즈니스 통합 모듈이 작성됩니다.

비즈니스 통합 Perspective에서 모듈을 펼쳐 콘텐츠를 보십시오. 모듈 프로젝트(프로젝트와 동일한 이름을 가짐) 아래의 첫 번째 항목을 두 번 클릭하여 어셈블리 편집기를 여십시오.

다음 옵션이 있습니다.

각 Java 서비스 재연결 옵션에 대한 장단점:

각 Java 서비스 재연결 옵션에 대한 장단점이 있습니다.

다음 목록은 두 가지 옵션 모두 또는 각 옵션에 대한 장단점에 대해 설명합니다.

- 웹 서비스 호출이 Java 컴포넌트 호출보다 더 느리기 때문에 첫 번째 옵션이 런타임 시에 더 좋은 성능을 제공할 수 있습니다.
- 첫 번째 옵션은 컨텍스트를 전파시킬 수 있는 반면 웹 서비스 호출은 동일한 방법으로 컨텍스트를 전파시키지 않습니다.
- 두 번째 옵션에서는 어떠한 사용자 정의 코드도 작성하지 않습니다.
- 두 번째 옵션은 Java 서비스 생성에 제한사항이 있기 때문에 일부 Java 인터페이스 정의에 불가능할 수 있습니다. Rational Application Developer 문서를 참조하십시오: (<http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.doc/ref/rlimit.html>)
- 두 번째 옵션은 인터페이스 변경이 발생할 수 있고 따라서 SCA 이용자에 대한 변경이 발생할 수 있습니다.
- 두 번째 옵션을 사용하려면 WebSphere Process Server 6.0 서버가 설치되어 있고 WebSphere Integration Developer와 함께 작동하도록 구성되어 있어야 합니다. WebSphere Integration Developer와 함께 작동하도록 구성된 설치된 런타임을 보려면 창 → 환경 설정 → 서버 → 설치된 런타임으로 이동하여 **WebSphere Process Server v6.0** 항목이 존재하는 경우 선택하고 제품이 설치된 위치를 가리키는지 확인하십시오. 서버가 존재하는 경우 이 항목이 선택되었는지 그리고, 이 서버가 실제로 설치되지 않은 경우에는 선택 취소되었는지 확인하십시오. 다른 서버를 추가하려는 경우 추가... 단추를 클릭할 수도 있습니다.
- Java 컴포넌트가 Java 스켈레톤이 WSDL에서 생성되는 하향식 접근 방법을 사용하여 WebSphere Studio Application Developer Integration Edition에 빌드된 경우, 이 Java 클래스를 입출력하는 매개변수는 서브 클래스 WSIFFormatPartImpl일 것입니다. 이 경우에 해당하면 옵션 1을 선택하여 원래의 WSDL/XSD에서 새 SCA 스타일 Java 스켈레톤을 생성하거나 옵션 2를 선택하여 원래 WSDL 인터페이스에서 새로운 일반 Java 스켈레톤(WSIF 또는 DataObject API에 종속되지 않음)을 생성하십시오.

사용자 정의 Java 컴포넌트 작성: 옵션 1:

권장되는 이주 기법은 Java 서비스를 SCA 컴포넌트로서 표시할 수 있는 WebSphere Integration Developer Java 컴포넌트 유형을 사용하는 것입니다. 이주 중에 SCA Java 인터페이스 스타일에서 기존의 Java 컴포넌트의 인터페이스 스타일로 변환하도록 Java 코드를 작성해야 합니다.

사용자 정의 Java 컴포넌트를 작성하려면 다음을 수행하십시오.

1. 모듈 프로젝트 아래에서 인터페이스를 펼치고 WebSphere Studio Application Developer Integration에서 이 Java 클래스에 대해 생성된 WSDL 인터페이스를 선택하십시오.
2. 이 인터페이스를 어셈블리 편집기로 끌어서 놓으십시오. 작성할 컴포넌트의 유형을 선택하는 대화 상자가 열립니다. 구현 유형이 없는 컴포넌트를 선택하고 확인을 클릭하십시오.
3. 일반 컴포넌트가 어셈블리 다이어그램에 표시됩니다. 해당 컴포넌트를 선택한 후 특성 보기로 이동하십시오.
4. 설명 탭에서 컴포넌트의 이름 및 표시 이름을 보다 구체적인 것으로 변경할 수 있습니다.
5. 세부사항 탭에서 어셈블리 편집기에 끌어서 놓은 하나의 인터페이스를 갖는 이 컴포넌트를 볼 수 있습니다.
6. 액세스하려고 시도 중인 Java 클래스가 서비스 프로젝트 자체에 포함되지 않는 경우 서비스 프로젝트의 클래스 경로에 있는지 확인하십시오.
7. 모듈 프로젝트를 마우스 오른쪽 단추로 클릭하고 **종속성 편집기 열기...**를 선택하십시오. **Java** 섹션 아래에서 이전 Java 클래스가 들어있는 프로젝트가 나열되는지 확인하십시오. 나열되지 않는 경우 **추가...**를 클릭하여 추가하십시오. 단추를 클릭하십시오.
8. 어셈블리 편집기로 돌아가서 방금 작성한 컴포넌트를 마우스 오른쪽 단추로 클릭하고 **구현 생성... → Java**를 선택하십시오. 그런 다음 Java 구현이 생성될 패키지를 선택하십시오. 그러면 복합 유형은 `commonj.sdo.DataObject`인 오브젝트로 표시되고 단순 유형은 동등한 Java 오브젝트로 표시되는 SCA 프로그래밍 모델에 따라 WSDL 인터페이스를 준수하는 스켈레톤 Java 서비스가 작성됩니다.

아래 코드 예는 다음을 보여줍니다.

1. 5.1 WSDL 인터페이스의 관련 정의
2. WSDL에 대응하는 WebSphere Studio Application Developer Integration Edition 5.1 Java 메소드
3. 동일한 WSDL에 대한 WebSphere Integration Developer 6.0 Java 메소드

아래 코드는 5.1 WSDL 인터페이스의 관련 정의를 보여줍니다.

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

```

</complexType>
</schema>
</types>
<message name="getStockInfoRequest">
<part name="symbol" type="xsd:string"/>
</message>
<message name="getStockInfoResponse">
<part name="result" type="xsd1:StockInfo"/>
</message>
<operation name="getStockInfo" parameterOrder="symbol">
<input message="tns:getStockInfoRequest"
name="getStockInfoRequest"/>
<output message="tns:getStockInfoResponse"
name="getStockInfoResponse"/>
</operation>

```

다음 코드는 WSDL에 해당하는 WebSphere Studio Application Developer Integration Edition 5.1 Java 메소드를 보여줍니다.

```

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

```

다음 코드는 동일한 WSDL에 대한 WebSphere Integration Developer 6.0 Java 메소드를 표시합니다.

```

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

```

이제 생성된 Java 구현 클래스에서 “//TODO” 태그가 표시되는 위치에 코드를 채워야 합니다. 다음 두 옵션이 있습니다.

1. 로직을 원래의 Java 클래스에서 이 클래스로 이동하고, DataObjects를 사용하도록 채택하십시오.
 - WebSphere Studio Application Developer Integration Edition에서 하향식 접근법을 선택했고 Java 컴포넌트가 DataObject 매개변수를 처리하려는 경우 권장되는 옵션입니다. 이 재작업은 WebSphere Studio Application Developer Integration Edition에서 WSDL 정의로부터 생성된 Java 클래스가 제거되어야 하는 WSIF 종속성을 갖기 때문에 필요합니다.
2. 이 생성된 Java 클래스 내부에 이전 Java 클래스의 개인용 인스턴스를 작성하고 다음을 수행하는 코드를 작성하십시오.
 - a. 생성된 Java 구현 클래스의 모든 매개변수를 이전 Java 클래스가 예상하는 매개변수로 변환
 - b. 변환된 매개변수로 이전 Java 클래스의 개인용 인스턴스 호출
 - c. 이전 Java 클래스의 리턴값을 생성된 Java 구현 메소드에 의해 선언되는 리턴값으로 변환

- d. 이 옵션은 WSIF 서비스 프록시가 새 6.0 스타일 Java 컴포넌트에 의해 이용되어야 하는 이용 시나리오에 권장됩니다.

위의 옵션 중 하나를 완료한 후 Java 서비스를 다시 연결해야 합니다. 어떤 참조도 없어야 하므로, Java 컴포넌트의 인터페이스를 다시 연결하면 됩니다.

- 이 서비스가 동일한 모듈의 비즈니스 프로세스에 의해 호출되는 경우 적합한 비즈니스 프로세스 참조에서 이 Java 컴포넌트 인터페이스로의 연결을 작성하십시오.
- 이 서비스가 다른 모듈의 비즈니스 프로세스에 의해 호출되는 경우 **SCA** 바인딩을 갖는 내보내기를 작성하고, 다른 모듈에서 이 내보내기를 해당 모듈의 어셈블리 편집기로 끌어 놓아서, 대응하는 **SCA** 바인딩을 갖는 내보내기를 작성하십시오. 적합한 비즈니스 프로세스 참조를 해당 가져오기로 연결하십시오.
- 이 서비스를 외부에 공개하기 위해 WebSphere Studio Application Developer Integration Edition에서 출력한 경우, 서비스 재출력 방법에 대한 지시사항은 "이주된 서비스를 액세스하기 위해 SCA 내보내기 작성" 섹션을 참조하십시오.

Java 웹 서비스 작성: 옵션 2:

고려할 대체 옵션은 Java 클래스 주위에 웹 서비스를 작성할 수 있게 하는 Rational Application Developer 웹 서비스 도구입니다.

주: 이 메소드를 사용하여 이주하려 시도하기 전에 사이트(<http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.was.creation.ui.doc/tasks/twsbeanw.html>)의 정보를 참조하십시오.

주: 이 옵션은 웹 서비스 마법사를 호출하기 전에 웹 서비스 런타임이 WebSphere Integration Developer를 통해 구성되어야 합니다.

WebSphere Studio Application Developer Integration Edition에서 하향식 접근법을 사용하여 Java 클래스 주위에 WSDL을 생성한 경우 다음 단계를 수행하십시오.

1. 새 웹 프로젝트를 작성하고 이 웹 프로젝트의 Java 소스 폴더에 서비스를 빌드하려는 Java 클래스를 복사하십시오.
2. 서비스를 작성하려는 Java 클래스에 대한 컨테이너인 엔터프라이즈 응용프로그램 프로젝트를 마우스 오른쪽 단추로 클릭하십시오.
3. 특성을 선택하고, 서버 특성으로 이동하여 대상 런타임이 **WebSphere Process Server v6.0**으로 설정되고 기본 서버가 설치된 **WebSphere Process Server v6.0**으로 설정되었는지 확인하십시오.
4. 테스트 서버를 시작하고 이 응용프로그램을 서버에 배치하고 성공적으로 시작하는지 확인하십시오.
5. 다음으로 서비스를 작성할 Java 클래스를 마우스 오른쪽 단추로 클릭하고 웹 서비스 → 웹 서비스 작성을 선택하십시오.
6. 웹 서비스 유형에 대해 **Java bean** 웹 서비스를 선택하고, 웹 서비스를 바로 배치하려는 경우가 아니면 웹 프로젝트에서 웹 서비스 시작 옵션을 선택 취소하십시오. 선택적으로 클라이언트 프록시도 생성하도록 선택할 수 있습니다. 다음을 클릭하십시오.
7. 마우스 오른쪽 단추를 클릭한 Java 클래스가 표시되면, 다음을 클릭하십시오.

8. 이제 서비스 배치 옵션을 구성해야 합니다. 편집... 단추를 클릭하십시오. 서버 유형에 대해 **WPS Server v6.0**을 선택하고 웹 서비스 런타임에 대해 **IBM WebSphere** 및 J2EE 버전 **1.4**를 선택하십시오. 이를 수행하여 올바른 조합을 선택할 수 없는 경우 "이주 준비" 섹션에서 J2EE 프로젝트를 v1.4 레벨로의 이주에 대한 정보를 참조하십시오. 확인을 클릭하십시오.
9. 서비스 프로젝트에 대해 웹 프로젝트의 이름을 입력하십시오. 또한 적합한 EAR 프로젝트를 선택하십시오. 다음을 클릭하십시오. 몇 분 정도 기다려야 할 수 있습니다.
10. 웹 서비스 Java Bean ID 패널에서 WSDL 정의를 포함할 WSDL 파일을 선택하십시오. 웹 서비스에서 공개하려는 메소드를 선택하고 적당한 스타일/인코딩(문서/리터럴, RPC/리터럴 또는 RPC/인코드)을 선택하십시오. 패키지에서 이름 공간으로 사용자 정의 �핑 정의 옵션을 선택하고 해당 Java 클래스의 인터페이스에 의해 사용되는 모든 Java 패키지에 대해 이주되는 Java 클래스에 고유한 이름 공간(기본 이름 공간은 동일한 Java 클래스를 사용하는 다른 웹 서비스를 작성할 경우 충돌을 일으킬 수도 있는 패키지 이름에 고유함.)을 선택하십시오. 적합한 경우 다른 매개변수를 완료하십시오.
11. 다음을 클릭하고 웹 서비스 패키지에서 이름 공간으로 �핑 패널에서 추가 단추를 클릭한 다음 작성된 행에서 Java Bean의 패키지 이름을 입력하고 이 Java 클래스를 고유하게 식별하는 사용자 정의 이름 공간을 추가하십시오. Java Bean 인터페이스에서 사용되는 모든 Java 패키지에 대해 �핑을 계속 추가하십시오.
12. 다음을 클릭하십시오. 몇 분 정도 기다려야 할 수 있습니다.
13. 완료를 클릭하십시오. 마법사를 완료한 후 서비스 프로젝트가 Java 서비스의 이용자인 경우 Java 서비스를 설명하는 생성된 WSDL 파일을 비즈니스 통합 모듈 프로젝트에 복사해야 합니다. WebContent/WEB-INF/wsdl 폴더 아래의 생성된 라우터 웹 프로젝트에서 파일을 찾을 수 있습니다. 비즈니스 통합 모듈 프로젝트를 새로 고치기/다시 빌드하십시오.
14. 비즈니스 통합 Perspective로 전환하고 모듈을 펼친 다음 웹 서비스 포트 논리 카테고리를 펼치십시오.
15. 이전 단계에서 작성된 포트를 선택하고 어셈블리 편집기로 끌어서 놓고 웹 서비스 바인딩을 갖는 가져오기를 작성하도록 선택하십시오. 프롬프트되는 경우 Java 클래스의 WSDL 인터페이스를 선택하십시오. 이제 5.1에서 Java 컴포넌트를 이용한 SCA 컴포넌트를 이 가져오기에 연결하여 수동 재연결 이주 단계를 완료할 수 있습니다.

인터페이스가 5.1 인터페이스와는 약간 다를 수 있으며 5.1 이용자와 새 가져오기 사이에 인터페이스 중개 컴포넌트를 삽입해야 할 수 있음을 주의하십시오. 이를 수행하려면 어셈블리 편집기에서 연결 도구를 클릭하고 SCA 소스 컴포넌트를 이 새로운 웹 서비스 바인딩을 갖는 가져오기에 연결하십시오. 인터페이스가 다르기 때문에 소스 및 대상 노드에 일치하는 인터페이스가 없습니다라고 프롬프트됩니다. 소스 및 대상 노드 사이에 인터페이스 �핑 작성을 선택하십시오. 어셈블리 편집기에서 작성된 �핑 컴포넌트를 두 번 클릭하십시오. �핑 편집기가 열립니다. 인터페이스 �핑 작성에 대한 지시사항은 Information Center를 참조하십시오.

WebSphere Studio Application Developer Integration Edition에서 하향식 접근법을 사용하고 WSDL 정의에서 Java 클래스를 생성한 후 다음 단계를 수행하십시오.

1. 새 웹 프로젝트를 작성하고 Java 스켈레톤을 생성하려는 WSDL 파일을 이 웹 프로젝트의 소스 폴더에 복사하십시오.

2. Java 스켈레톤을 생성하려는 PortType을 포함하는 WSDL 파일을 마우스 오른쪽 단추로 클릭하고 웹 서비스 → **Java Bean** 스켈레톤 생성을 선택하십시오.
3. 웹 서비스 유형 스켈레톤 **Java Bean** 웹 서비스를 선택하고 마법사를 완료하십시오.

마법사를 완료한 후 서비스 인터페이스를 구현하고 WSIF API에 종속되지 않는 Java 클래스가 있어야 합니다.

EJB 서비스 이주:

EJB 서비스를 Stateless 세션 Bean 바인딩을 갖는 SCA 가져오기로 이주할 수 있습니다.

WebSphere Studio Application Developer Integration Edition 서비스 프로젝트가 다른 EJB, EJB 클라이언트 또는 Java 프로젝트에 종속된 경우 파일 → 가져오기 → 기존 프로젝트 작업공간으로 마법사를 사용하여 기존 프로젝트를 가져오십시오. 일반적으로 서비스 프로젝트에서 EJB를 참조한 경우가 이에 해당합니다. 서비스 프로젝트에서 참조된 WSDL 또는 XSD 파일이 다른 유형의 프로젝트에 있는 경우 이전 비서비스 프로젝트와 이름이 동일한 새 비즈니스 통합 라이브러리를 작성하여 이 아티팩트 모두를 라이브러리로 복사하십시오.

이주 마법사를 사용하여 서비스 프로젝트를 가져오십시오. 그러면 WSDL 메시지, PortTypes, 바인딩 및 서비스가 WebSphere Studio Application Developer Integration Edition에 생성되는 비즈니스 통합 모듈이 작성됩니다.

비즈니스 통합 Perspective에서 모듈을 펼쳐 콘텐츠를 보십시오. 모듈 프로젝트(프로젝트와 동일한 이름을 가짐) 아래의 첫 번째 항목을 두 번 클릭하여 어셈블리 편집기를 여십시오.

다음 옵션이 있습니다.

각 EJB 서비스 재연결 옵션에 대한 장단점:

각 EJB 서비스 재연결 옵션에 대한 장단점이 있습니다.

다음 목록은 두 가지 옵션 모두 또는 각 옵션에 대한 장단점에 대해 설명합니다.

- 웹 서비스 호출이 EJB 호출보다 더 느리기 때문에 첫 번째 옵션이 런타임 시에 더 좋은 성능을 제공할 수 있습니다.
- 첫 번째 옵션은 컨텍스트를 전파시킬 수 있는 반면 웹 서비스 호출은 동일한 방법으로 컨텍스트를 전파시키지 않습니다.
- 두 번째 옵션에서는 어떠한 사용자 정의 코드도 작성하지 않습니다.
- 두 번째 옵션은 EJB 서비스 생성에 제한사항이 있기 때문에 일부 EJB 인터페이스 정의에 불가능할 수 있습니다. Rational Application Developer 문서를 참조하십시오: (<http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.doc/ref/rlimit.html>)
- 두 번째 옵션은 인터페이스 변경이 발생할 수 있고 따라서 SCA 이용자에 대한 변경이 발생할 수 있습니다.
- 두 번째 옵션을 사용하려면 WebSphere Process Server 6.0 서버가 설치되어 있고 WebSphere Integration Developer와 함께 작동하도록 구성되어 있어야 합니다. WebSphere Integration Developer와 함께 작동하도록 구성된 설치된 런타임을 보려면 창 → 환경 설정 → 서버 → 설치된 런타임으로 이동하여 **WebSphere**

Process Server v6.0 항목이 존재하는 경우 선택하고 제품이 설치된 위치를 가리키는지 확인하십시오. 서버가 존재하는 경우 이 항목이 선택되었는지 그리고, 이 서버가 실제로 설치되지 않은 경우에는 선택 취소되었는지 확인하십시오. 다른 서버를 추가하려는 경우 추가... 단추를 클릭할 수도 있습니다.

- Java 컴포넌트가 EJB 스켈레톤이 WSDL에서 생성되는 하향식 접근법을 사용하여 WebSphere Studio Application Developer Integration Edition에 빌드된 경우, 이 Java 클래스를 입출력하는 매개변수는 서브 클래스 WSIFFormatPartImpl일 것입니다. 이 경우에 해당하면 옵션 2를 선택하여 원래 WSDL 인터페이스에서 새로운 일반 EJB 스켈레톤(WSIF 또는 DataObject API에 종속되지 않음)을 생성하십시오.

사용자 정의 EJB 컴포넌트 작성: 옵션 1:

권장하는 이주 기법은 사용자가 Stateless 세션 EJB를 SCA 컴포넌트로서 호출하게 하는 Stateless 세션 바인딩을 갖는 WebSphere Integration Developer 가져오기를 사용하는 것입니다. 이주 중에 SCA Java 인터페이스 스타일에서 기존의 EJB 인터페이스 스타일로 변환하도록 Java 코드를 작성해야 합니다.

사용자 정의 EJB 컴포넌트를 작성하려면 다음을 수행하십시오.

1. 모듈 프로젝트 아래에서 인터페이스를 펼치고 WebSphere Studio Application Developer Integration에서 이 EJB에 대해 생성된 WSDL 인터페이스를 선택하십시오.
2. 이 인터페이스를 어셈블리 편집기로 끌어서 놓으십시오. 작성할 컴포넌트의 유형을 선택하는 대화 상자가 열립니다. 구현 유형이 없는 컴포넌트를 선택하고 확인을 클릭하십시오.
3. 일반 컴포넌트가 어셈블리 다이어그램에 표시됩니다. 해당 컴포넌트를 선택한 후 특성 보기로 이동하십시오.
4. 설명 탭에서 컴포넌트의 이름 및 표시 이름을 보다 구체적인 것으로 변경할 수 있습니다. EJB의 이름과 비슷한 이름을 선택하지만 “JavaMed” 같은 접미사를 붙이십시오. 이것이 WebSphere Studio Application Developer Integration에서 EJB에 대해 생성되는 WSDL 인터페이스와 EJB의 Java 인터페이스 사이를 중개하는 Java 컴포넌트가 되기 때문입니다.
5. 세부사항 탭에서 어셈블리 편집기에 끌어서 놓은 하나의 인터페이스를 갖는 이 컴포넌트를 볼 수 있습니다.
6. 어셈블리 편집기로 돌아가서 방금 작성한 컴포넌트를 마우스 오른쪽 단추로 클릭하고 구현 생성... → Java를 선택하십시오. 그런 다음 Java 구현이 생성될 패키지를 선택하십시오. 그러면 복합 유형은 commonj.sdo.DataObject인 오브젝트로 표시되고 단순 유형은 동등한 Java 오브젝트로 표시되는 SCA 프로그래밍 모델에 따라 WSDL 인터페이스를 준수하는 스켈레톤 Java 서비스가 작성됩니다.

아래 코드 예는 다음을 보여줍니다.

1. 5.1 WSDL 인터페이스의 관련 정의
2. WSDL에 대응하는 WebSphere Studio Application Developer Integration Edition 5.1 Java 메소드
3. 동일한 WSDL에 대한 WebSphere Integration Developer 6.0 Java 메소드

아래 코드는 5.1 WSDL 인터페이스의 관련 정의를 보여줍니다.

```

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

```

다음 코드는 WSDL에 해당하는 WebSphere Studio Application Developer Integration Edition 5.1 Java 메소드를 보여줍니다.

```

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

```

다음 코드는 동일한 WSDL에 대한 WebSphere Integration Developer 6.0 Java 메소드를 표시합니다.

```

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

```

따라서 생성된 Java 구현 클래스에서 “//TODO” 태그가 표시되는 위치에 실제 코드를 채워야 합니다. 먼저 이 Java 컴포넌트에서 실제 EJB로의 참조를 작성하여 SCA 프로그래밍 모델에 따라서 EJB에 액세스할 수 있게 해야 합니다.

1. 어셈블리 편집기를 계속 열어두고 J2EE Perspective로 전환하십시오. 서비스를 작성하려는 EJB가 들어있는 EJB 프로젝트를 찾으십시오.
2. 배치 설명자: <project-name> 항목을 펼치고 EJB를 찾으십시오. EJB를 어셈블리 편집기로 끌어서 놓으십시오. 프로젝트 종속성을 갱신하도록 경고하는 경우 모듈 종속성 편집기 열기... 선택란을 선택하고 확인을 클릭하십시오.
3. J2EE 섹션 아래에서 EJB 프로젝트가 나열되는지 확인하고 나열되지 않는 경우 추가...를 클릭하여 추가하십시오. 단추를 클릭하십시오.
4. 모듈 종속성을 저장하고 편집기를 닫으십시오. 새 가져오기가 어셈블리 편집기에서 작성되었음을 알 수 있습니다. 가져오기를 선택하고 설명 탭의 특성 보기로 이동하여 가져오기의 이름 및 표시 이름을 보다 구체적인 것으로 변경할 수 있습니다. 바인딩 탭에서 가져오기 유형이 자동으로 **Stateless** 세션 **Bean** 바인딩으로 설정되고 EJB의 JNDI 이름이 이미 적절하게 설정되었음을 알 수 있습니다.
5. 어셈블리 편집기의 팔레트에서 연결 도구를 선택하십시오.
6. Java 컴포넌트를 클릭하고 마우스를 놓으십시오.
7. 다음에 EJB 가져오기를 클릭하고 마우스를 놓으십시오.
8. 일치하는 참조가 소스 노드에 작성됩니다. 계속하시겠습니까? 표시되면 확인을 클릭하십시오. 두 컴포넌트 사이에 연결이 작성됩니다.
9. 어셈블리 편집기 및 세부사항 탭 아래의 특성 보기에서 Java 컴포넌트를 선택하고, 참조를 펼친 다음 방금 작성된 EJB에 대한 참조를 선택하십시오. 생성되는 이름이 매우 구체적이거나 적합하지 않은 경우 참조의 이름을 갱신할 수 있습니다. 나중에 사용할 수 있도록 이 참조의 이름을 기억하십시오.
10. 어셈블리 다이어그램을 저장하십시오.

SCA 프로그래밍 모델을 사용하여 생성된 Java 클래스로부터 EJB를 호출해야 합니다. 생성된 Java 클래스를 열고 다음 단계에 따라서 EJB 서비스를 호출하는 코드를 작성하십시오. 생성된 Java 구현 클래스에 대해 다음을 수행하십시오.

1. 개인용 변수(유형은 원격 EJB 인터페이스의 유형임)를 작성하십시오.

```
private YourEJBInterface ejbService = null;
```
2. 이들 변수가 사용자 EJB 인터페이스에서 복합 유형인 경우 BOFactory에 대한 개인용 변수도 작성하십시오.

```
private BOFactory boFactory = (BOFactory)
    ServiceManager.INSTANCE.locateService("com/ibm/websphere/bo/BOFactory");
```
3. Java 구현 클래스의 생성자에서, SCA API를 사용하여 EJB 참조를 해석하고(이전 단계에서 기록한 EJB 참조의 이름을 기억하기.) 개인용 변수를 이 참조와 동일하게 설정하십시오.

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

생성된 Java 구현 클래스의 각 “//TODO”에 대해,

1. 모든 매개변수를 EJB가 예상하는 매개변수 유형으로 변환하십시오.

2. SCA 프로그래밍 모델을 사용하여 EJB 참조에서 적당한 메소드를 호출하여 변환된 매개변수를 송신하십시오.
3. EJB의 리턴값을 생성된 Java 구현 메소드에서 선언한 리턴값으로 변환하십시오.

```
/**
 * Method generated to support the implementing WSDL port type named
 * "interface.MyBean".
 */
public BusObjImpl getStockInfo(String aString) {
    BusObjImpl boImpl = null;
    try {
        // invoke the EJB method
        StockInfo stockInfo = this.ejbService.getStockInfo(aString);
        // formulate the SCA data object to return.
        boImpl = (BusObjImpl)
            this.boFactory.createClass(StockInfo.class);
        // manually convert all data from the EJB return type into the
        // SCA data object to return
        boImpl.setInt("index", stockInfo.getIndex());
        boImpl.setString("symbol", stockInfo.getSymbol());
        boImpl.setDouble("price", stockInfo.getPrice());
    } catch (RemoteException e) {
        e.printStackTrace();
    }
    return boImpl;
}
/**
 * Method generated to support the implementing WSDL port type named
 * "interface.MyBean".
 */
public void setStockPrice(String symbol, Float newPrice) {
    try {
        this.ejbService.setStockPrice(symbol, newPrice.floatValue());
    } catch (RemoteException e) {
        e.printStackTrace();
    }
}
```

EJB 웹 서비스 작성: 옵션 2:

고려할 대체 옵션은 EJB 주위에 웹 서비스를 작성할 수 있게 하는 Rational Application Developer 웹 서비스 도구입니다.

주: 이 메소드를 사용하여 이주를 시도하기 전에 <http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.was.creation.ejb.ui.doc/tasks/twsejbw.html> 사이트의 정보를 참조하십시오.

주: 이 옵션은 웹 서비스 마법사를 호출하기 전에 웹 서비스 런타임이 WebSphere Integration Developer를 통해 구성되어야 합니다.

EJB 주위에 웹 서비스를 작성하려면 다음 단계를 수행하십시오.

1. 서비스를 작성 중인 EJB용 컨테이너인 엔터프라이즈 응용프로그램 프로젝트를 마우스 오른쪽 단추로 클릭하십시오.

2. 특성을 선택하고, 서버 특성으로 이동하여 대상 런타임이 **WebSphere Process Server v6.0**으로 설정되고 기본 서버가 설치된 **WebSphere Process Server v6.0**으로 설정되었는지 확인하십시오.
3. 테스트 서버를 시작하고 이 응용프로그램을 서버에 배치하고 성공적으로 시작하는지 확인하십시오.
4. J2EE Perspective에서 프로젝트 탐색기 보기의 **EJB** 프로젝트를 펼치십시오. 배치 설명자를 펼친 후 섹션 **Bean** 카테고리를 펼치십시오. 웹 서비스를 생성하려는 Bean을 선택하십시오.
5. 마우스 오른쪽 단추를 클릭하여 웹 서비스 → 웹 서비스 작성을 선택하십시오.
6. 웹 서비스 유형에 대해 **EJB** 웹 서비스를 선택하고 바로 웹 서비스를 배치하려는 경우가 아니면 웹 프로젝트에서 웹 서비스 시작 옵션을 선택 취소하십시오. 다음을 클릭하십시오.
7. 마우스 오른쪽 단추로 클릭한 EJB가 여기에 선택되어 있는지 확인하고 다음을 클릭하십시오.
8. 이제 서비스 배치 옵션을 구성해야 합니다. 편집... 단추를 클릭하십시오. 서버 유형에 대해 **WPS Server v6.0**을 선택하고 웹 서비스 런타임에 대해 **IBM WebSphere** 및 J2EE 버전 **1.4**를 선택하십시오. 이를 수행하여 올바른 조합을 선택할 수 없는 경우 "이주 준비" 섹션에서 J2EE 프로젝트를 v1.4 레벨로의 이주에 대한 정보를 참조하십시오. 확인을 클릭하십시오.
9. 서비스 프로젝트에 대해 EJB가 포함된 EJB 프로젝트의 이름을 입력하십시오. 또한 적합한 EAR 프로젝트를 선택하십시오. 다음을 클릭하십시오. 몇 분 정도 기다려야 할 수 있습니다.
10. 웹 서비스 EJB 구성 패널에서 사용할 적당한 라우터 프로젝트를 선택하십시오. (작성할 라우터 웹 프로젝트의 이름을 선택하십시오. 이 프로젝트는 원래의 EJB와 동일한 엔터프라이즈 응용프로그램에 추가됩니다.) 원하는 전송(**SOAP over HTTP** 또는 **SOAP over JMS**)을 선택하십시오. 다음을 클릭하십시오.
11. WSDL 정의를 포함할 WSDL 파일을 선택하십시오. 웹 서비스에서 공개하려는 메소드를 선택하고 적당한 스타일/인코딩(문서/리터럴, RPC/리터럴 또는 RPC/인코드)을 선택하십시오. 패키지에서 이름 공간으로 사용자 정의 �핑 정의 옵션을 선택하고 EJB에 의해 사용되는 모든 Java 패키지에 대해 이주되는 EJB에 고유한 이름 공간(기본 이름 공간은 동일한 Java 클래스를 사용하는 다른 웹 서비스를 작성할 경우 충돌을 일으킬 수도 있는 패키지 이름에 고유합니다.)을 선택하십시오. 적합한 경우 다른 매개변수를 완료하십시오. 각 스타일/인코딩 조합에 제한사항이 있습니다. 자세한 정보는 <http://publib.boulder.ibm.com/infocenter/rtnl0600/topic/com.ibm.etools.webservice.doc/ref/rlimit.html>의 제한사항을 참조하십시오.
12. 다음을 클릭하고 웹 서비스 패키지에서 이름 공간으로 �핑 패널에서 추가 단추를 클릭한 다음 작성된 행에서 해당 EJB의 패키지 이름과 이 EJB를 고유하게 식별하는 사용자 정의 이름 공간을 입력하십시오. EJB 인터페이스에서 사용되는 모든 Java 패키지에 대해 �핑을 계속 추가하십시오.
13. 다음을 클릭하십시오. 몇 분 정도 기다려야 할 수 있습니다.
14. 완료를 클릭하십시오. 마법사를 완료한 후 서비스 프로젝트가 EJB 서비스의 이용자인 경우 EJB 서비스를 설명하는 생성된 WSDL 파일을 비즈니스 통합 모듈 프로젝트에 복사해야 합니다. WebContent/WEB-INF/wsdl 폴더 아래의 생성된 라우터 웹 프로젝트에서 파일을 찾을 수 있습니다. 비즈니스 통합 모듈 프로젝트를 새로 고치기/다시 빌드하십시오.
15. 비즈니스 통합 Perspective로 전환하고 이주된 모듈을 펼친 후 웹 서비스 포트 논리 카테고리를 펼치십시오.

16. 이전 단계에서 생성된 포트를 선택하고 어셈블리 편집기로 끌어서 놓고 웹 서비스 바인딩을 갖는 가져오기를 작성할 것을 선택하십시오. 프롬프트되는 경우 EJB의 WSDL 인터페이스를 선택하십시오. 이제 5.1에서 EJB를 이용한 SCA 컴포넌트를 이 가져오기에 연결하여 수동 재연결 이주 단계를 완료하십시오.

WebSphere Studio Application Developer Integration Edition에서 하향식 접근 방식을 사용하여 WSDL 정의에서 EJB 스켈레톤을 생성한 후 다음 단계를 수행하십시오.

1. 새 웹 프로젝트를 작성하고 EJB 스켈레톤을 생성하려는 WSDL 파일을 이 웹 프로젝트의 소스 폴더에 복사하십시오.
2. EJB 스켈레톤을 생성하려는 PortType이 들어있는 WSDL 파일을 마우스 오른쪽 단추로 클릭하고 웹 서비스 → **Java Bean** 스켈레톤 생성을 선택하십시오.
3. 웹 서비스 유형 스켈레톤 **EJB** 웹 서비스를 선택하고 마법사를 완료하십시오.

마법사를 완료하면 서비스 인터페이스를 구현하며 WSIF API에 종속되지 않는 EJB가 있어야 합니다.

인터페이스가 5.1 인터페이스와는 약간 다를 수 있으며 5.1 이용자와 새 가져오기 사이에 인터페이스 중개 컴포넌트를 삽입해야 할 수 있음을 주의하십시오. 이를 수행하려면 어셈블리 편집기에서 연결 도구를 클릭하고 SCA 소스 컴포넌트를 이 새로운 웹 서비스 바인딩을 갖는 가져오기에 연결하십시오. 인터페이스가 다르기 때문에 소스 및 대상 노드에 일치하는 인터페이스가 없습니다라고 프롬프트됩니다. 소스 및 대상 노드 사이에 인터페이스 매핑 작성을 선택하십시오. 어셈블리 편집기에서 작성된 매핑 컴포넌트를 두 번 클릭하십시오. 매핑 편집기가 열립니다. 인터페이스 매핑 작성에 대한 지시사항은 Information Center를 참조하십시오.

이 단계를 완료한 후에 EJB 서비스를 다시 연결해야 합니다. 어떤 참조도 없어야 하므로, Java 컴포넌트의 인터페이스를 다시 연결하면 됩니다.

- 이 서비스가 동일한 모듈의 비즈니스 프로세스에 의해 호출되는 경우 적합한 비즈니스 프로세스 참조에서 이 EJB 컴포넌트로의 연결을 작성하십시오.
- 이 서비스가 다른 모듈의 비즈니스 프로세스에 의해 호출되는 경우 **SCA** 바인딩을 갖는 내보내기를 작성하고, 다른 모듈에서 이 내보내기를 해당 모듈의 어셈블리 편집기로 끌어 놓아서, 대응하는 **SCA** 바인딩을 갖는 내보내기를 작성하십시오. 적합한 비즈니스 프로세스 참조를 해당 가져오기로 연결하십시오.
- 이 서비스를 외부에 공개하기 위해 WebSphere Studio Application Developer Integration Edition에 출력된 경우, 서비스 재출력 방법에 대한 지시사항은 "인바운드 BPEL 이외 서비스 이주" 섹션을 참조하십시오.

비즈니스 프로세스를 비즈니스 프로세스 서비스 호출로 이주:

이 시나리오는 다른 비즈니스 프로세스를 호출하고 두 번째 비즈니스 프로세스는 WSIF 프로세스 바인딩을 사용하여 호출되는 비즈니스 프로세스에 적용됩니다. 이 섹션은 연결 또는 SCA 바인딩을 갖는 가져오기/내보내기를 사용하여 BPEL 서비스 호출로 BPEL을 이주하는 방법을 설명합니다.

아웃바운드 서비스에 대한 프로세스(BPEL) 바인딩 서비스 프로젝트를 이주하려면 다음 단계를 수행하십시오.

1. 비즈니스 통합 Perspective에서 모듈을 펼쳐 콘텐츠를 보십시오. 모듈 프로젝트(프로젝트와 동일한 이름을 가짐) 아래의 첫 번째 항목을 두 번 클릭하여 어셈블리 편집기를 여십시오.

2. BPEL 프로세스가 다른 BPEL 프로세스를 호출할 수 있는 여러가지 시나리오가 있습니다. 사용자 응용프로그램에 적용되는 아래 시나리오를 찾으십시오.
 - 호출되는 BPEL이 동일한 모듈에 있는 경우 첫 번째 BPEL 구성요소의 적당한 참조에서 대상 BPEL 구성요소의 적합한 인터페이스로의 연결을 작성하십시오.
 - 호출되는 BPEL이 다른 모듈에 있는 경우(다른 모듈은 이주된 서비스 프로젝트임):
 - a. 두 번째 비즈니스 프로세스의 모듈 어셈블리 다이어그램에서 해당 프로세스에 대한 SCA 바인딩을 갖는 내보내기를 작성하십시오.
 - b. 비즈니스 통합 보기의 네비게이터에서 두 번째 모듈의 어셈블리 아이콘을 펼치십시오. 방금 작성한 내보내기가 표시되어야 합니다.
 - c. 내보내기를 두 번째 모듈의 비즈니스 통합 보기에서 첫 번째 모듈의 열린 어셈블리 편집기 위로 끌어 놓으십시오. 그러면 첫 번째 모듈에 SCA 바인딩을 갖는 가져오기가 작성됩니다. 이 서비스를 외부에 공개하기 위해 WebSphere Studio Application Developer Integration Edition에서 출력한 경우, "이주된 서비스를 액세스하기 위해 SCA 내보내기 작성" 섹션을 참조하십시오.
 - d. 첫 번째 비즈니스 프로세스의 적합한 참조를 해당 모듈에 방금 작성한 가져오기로 연결하십시오.
 - e. 어셈블리 다이어그램을 저장하십시오.
 - 두 번째 비즈니스 프로세스를 호출할 때 늦은 바인딩을 수행하려면:
 - a. 첫 번째 비즈니스 프로세스 구성요소의 참조를 연결이 끊긴 상태로 유지하십시오. BPEL 편집기에서 첫 번째 프로세스를 열고 참조 파트너 섹션에서 늦은 바인딩을 사용하여 호출할 두 번째 BPEL 프로세스에 상응하는 파트너를 선택하십시오.
 - b. 설명 탭의 특성 보기에서 두 번째 비즈니스 프로세스의 이름을 프로세스 템플릿 필드에 입력하십시오.
 - c. 비즈니스 프로세스를 저장하십시오. 이제 늦은 바인딩 호출의 설정을 완료했습니다.

웹 서비스(SOAP/JMS) 이주:

웹 서비스(SOAP/JMS)를 웹 서비스 바인딩을 갖는 SCA 가져오기로 이주할 수 있습니다.

아웃바운드 서비스 이주를 위해 SOAP/JMS 서비스 프로젝트를 이주하려면 다음 단계를 수행하십시오.

1. 첫 번째, 이주 마법사를 사용하여 서비스 프로젝트를 가져와야 합니다. 그러면 WSDL 메시지, PortTypes, 바인딩 및 서비스가 WebSphere Studio Application Developer Integration Edition에 생성되는 비즈니스 통합 모듈이 작성됩니다. 이 응용프로그램에서 호출하는 IBM Web Service(SOAP/JMS)도 이주할 WebSphere Studio Application Developer Integration Edition 웹 서비스인 경우 이주 중에 해당 웹 서비스를 갱신해야 할 수도 있습니다. 이 경우에 해당하면 여기에서 웹 서비스의 이주된 WSDL 파일을 사용해야 합니다.
2. 비즈니스 통합 Perspective에서 해당 콘텐츠를 볼 수 있도록 모듈을 펼치십시오. 모듈 프로젝트(프로젝트와 동일한 이름을 가짐) 아래의 첫 번째 항목을 두 번 클릭하여 어셈블리 편집기를 여십시오.

3. 다음으로 응용프로그램이 SCA 프로그래밍 모델에 따라서 IBM Web Service와(SOAP/JMS를 통한) 상호작용하도록 허용하는 가져오기를 추가하십시오. WSDL 인터페이스, 바인딩 및 서비스 정의가 이주된 모듈 또는 이주된 모듈이 종속되는 라이브러리에 존재하는지 확인하십시오.
4. 비즈니스 통합 Perspective에서 이주된 모듈을 펼치고 어셈블리 편집기에서 어셈블리 다이어그램을 여십시오.
5. 웹 서비스 포트 논리 카테고리를 펼치고 호출하려는 서비스에 해당하는 포트를 어셈블리 편집기로 끌어서 놓으십시오.
6. 웹 서비스 바인딩을 갖는 가져오기 작성을 선택하십시오.
7. 가져오기를 작성한 후 어셈블리 편집기에서 선택하고 특성 보기로 이동하십시오. 바인딩 탭 아래에 가져오기가 바인드되는 포트와 서비스가 표시됩니다.
8. 어셈블리 다이어그램을 저장하십시오.

이 단계를 완료한 후 서비스를 다시 연결해야 합니다.

- 이 서비스가 동일한 모듈의 비즈니스 프로세스에 의해 호출되는 경우 적합한 비즈니스 프로세스 참조에서 이 가져오기로의 연결을 작성하십시오.
- 이 서비스가 다른 모듈의 비즈니스 프로세스에 의해 호출되는 경우 SCA 바인딩을 갖는 내보내기를 작성하고, 다른 모듈에서 이 내보내기를 해당 모듈의 어셈블리 편집기로 끌어 놓아서, 대응하는 SCA 바인딩을 갖는 내보내기를 작성하십시오. 적합한 비즈니스 프로세스 참조를 해당 가져오기로 연결하십시오.
- 어셈블리 다이어그램을 저장하십시오.

웹 서비스(SOAP/HTTP) 이주:

웹 서비스(SOAP/HTTP)를 웹 서비스 바인딩을 갖는 SCA 가져오기로 이주할 수 있습니다.

아웃바운드 서비스 이주를 위해 SOAP/HTTP 서비스 프로젝트를 이주하려면 다음 단계를 수행하십시오.

1. 첫 번째, 이주 마법사를 사용하여 서비스 프로젝트를 가져와야 합니다. 그러면 WSDL 메시지, PortTypes, 바인딩 및 서비스가 WebSphere Studio Application Developer Integration Edition에 생성되는 비즈니스 통합 모듈이 작성됩니다. 이 응용프로그램에서 호출하는 IBM Web Service(SOAP/HTTP)도 이주할 WebSphere Studio Application Developer Integration Edition 웹 서비스인 경우 이주 중에 해당 웹 서비스를 갱신해야 할 수도 있습니다. 이 경우에 해당하면 여기에서 웹 서비스의 이주된 WSDL 파일을 사용해야 합니다.
2. 비즈니스 통합 Perspective에서 해당 콘텐츠를 볼 수 있도록 모듈을 펼치십시오. 모듈 프로젝트(프로젝트와 동일한 이름을 가짐) 아래의 첫 번째 항목을 두 번 클릭하여 어셈블리 편집기를 여십시오.
3. 다음으로 응용프로그램이 SCA 프로그래밍 모델에 따라서 IBM Web Service와(SOAP/HTTP를 통한) 상호작용하도록 허용하는 가져오기를 추가하십시오. WSDL 인터페이스, 바인딩 및 서비스 정의가 이주된 모듈 또는 이주된 모듈이 종속되는 라이브러리에 존재하는지 확인하십시오.
4. 비즈니스 통합 Perspective에서 이주된 모듈을 펼치고 어셈블리 편집기에서 어셈블리 다이어그램을 여십시오.

5. 웹 서비스 포트 논리 카테고리를 펼치고 호출하려는 서비스에 해당하는 포트를 어셈블리 편집기로 끌어서 놓으십시오.
6. 웹 서비스 바인딩을 갖는 가져오기 작성을 선택하십시오.
7. 가져오기를 작성한 후 어셈블리 편집기에서 선택하고 특성 보기로 이동하십시오. 바인딩 탭 아래에 가져오기가 바인드되는 포트와 서비스가 표시됩니다.
8. 어셈블리 다이어그램을 저장하십시오.

이 단계를 완료한 후 서비스를 다시 연결해야 합니다.

- 이 서비스가 동일한 모듈의 비즈니스 프로세스에 의해 호출되는 경우 적합한 비즈니스 프로세스 참조에서 이 가져오기로의 연결을 작성하십시오.
- 이 서비스가 다른 모듈의 비즈니스 프로세스에 의해 호출되는 경우 **SCA** 바인딩을 갖는 내보내기를 작성하고, 다른 모듈에서 이 내보내기를 해당 모듈의 어셈블리 편집기로 끌어 놓아서, 대응하는 **SCA** 바인딩을 갖는 내보내기를 작성하십시오. 적합한 비즈니스 프로세스 참조를 해당 가져오기로 연결하십시오.
- 어셈블리 다이어그램을 저장하십시오.

JMS 서비스 이주:

JMS 서비스를 JMS 바인딩을 갖는 SCA 가져오기로 이주할 수 있습니다.

주: JMS 메시지가 WebSphere Business Integration 어댑터로 송신되고 있는 경우 "WebSphere Business Integration 어댑터와의 상호작용 이주" 섹션을 참조하십시오.

아웃바운드 서비스 이주를 위해 JMS 서비스 프로젝트를 이주하려면 다음 단계를 수행하십시오.

1. 첫 번째, 이주 마법사를 사용하여 서비스 프로젝트를 가져와야 합니다. 그러면 WSDL 메시지, PortTypes, 바인딩 및 서비스가 WebSphere Studio Application Developer Integration Edition에 생성되는 비즈니스 통합 모듈이 작성됩니다.
2. 비즈니스 통합 Perspective에서 해당 콘텐츠를 볼 수 있도록 모듈을 펼치십시오. 모듈 프로젝트(프로젝트와 동일한 이름을 가짐) 아래의 첫 번째 항목을 두 번 클릭하여 어셈블리 편집기를 여십시오.
3. 다음으로 응용프로그램이 SCA 프로그래밍 모델에 따라서 JMS 대기열과 상호작용할 가져오기를 추가하십시오.
4. 어셈블리 편집기에서 이주된 모듈 프로젝트를 펼치고 인터페이스 카테고리를 펼치고 응용프로그램이 호출할 웹 서비스를 설명하는 WSDL PortType을 찾으십시오. EJB를 어셈블리 편집기로 끌어서 놓으십시오.
5. 컴포넌트 작성 대화 상자에서 작성할 컴포넌트의 유형을 선택할 수 있습니다. 바인딩이 없는 가져오기를 선택하십시오.
6. 어셈블리 편집기에 새 가져오기가 작성되었음을 알 수 있으며 해당 가져오기를 선택하고 특성 보기로 이동하면 설명 탭에서 가져오기의 이름과 표시 이름을 보다 구체적인 것으로 변경할 수 있습니다.
7. 5.1 WSDL 바인딩 및 서비스 파일을 참조하면 이주하고 있는 JMS 서비스에 대한 세부사항을 찾을 수 있으며 이 내용으로 6.0 "JMS 바인딩으로 가져오기"의 세부사항을 채울 수 있습니다. 5.1 서비스 프로젝트 안에서 5.1 JMS 바인딩 및 서비스 WSDL 파일(이름은 대개 *JMSBinding.wsdl 및 *JMSService.wsdl임)

을 찾으십시오. 여기에서 캡처된 바인딩 및 서비스 정보를 검사하십시오. 바인딩에서 텍스트 또는 오브젝트 메시지가 사용되었는지 여부와 사용자 정의 데이터 형식 바인딩이 사용되었는지 여부를 확인할 수 있습니다. 사용된 항목이 있는 경우 6.0 "JMS 바인딩으로 가져오기"에 대해서도 사용자 정의 데이터 바인딩을 작성하는 것이 좋습니다. 서비스에서 초기 컨텍스트 팩토리, JNDI 연결 팩토리 이름, JNDI 대상 이름, 대상 스타일(큐) 등을 찾을 수 있습니다.

8. 가져오기를 마우스 오른쪽 단추로 클릭하고 바인딩 생성을 선택한 후 **JMS** 바인딩을 선택하십시오. 다음 매개변수를 입력하라는 프롬프트가 표시됩니다.

JMS 메시징 도메인 선택:

- 지점간
- 공개-등록
- 도메인 독립

데이터가 비즈니스 오브젝트와 JMS 메시지 사이에 직렬화되는 방법 선택:

- 텍스트
- 오브젝트
- 사용자 제공

사용자 제공이 선택되는 경우:

com.ibm.websphere.sca.jms.data.JMSDataBinding 구현 클래스의 완전한 이름을 지정하십시오. 응용 프로그램이 JMS Import 바인딩에서 정상적으로 사용할 수 없는 JMS 헤더 특성을 설정할 필요가 있는 경우 사용자 정의 데이터 바인딩을 지정해야 합니다. 이 경우 표준 JMS 데이터 바인딩 "com.ibm.websphere.sca.jms.data.JMSDataBinding"을 확장하는 사용자 정의 데이터 바인딩 클래스를 작성하고 JMSMessage에 직접 액세스할 수 있는 사용자 정의 코드를 추가할 수 있습니다. 아래 링크에서 "컴포넌트 가져오기 및 내보내기 바인딩 작성 및 수정"의 JMS 예제를 참조하십시오.

인바운드 연결에서 기본 JMS 가능 선택기 클래스를 사용 중임:

<selected> 또는 <deselected>

9. 방금 작성한 가져오기를 선택하십시오. 특성 보기에서 바인딩 탭으로 이동하십시오. 수동으로 나열되어 있는 모든 바인딩 정보를 전에 WebSphere Studio Application Developer Integration Edition에서 지정한 것과 동일한 값으로 채울 수 있습니다. 지정할 수 있는 바인딩 정보는 다음과 같습니다.
 - JMS 가져오기 바인딩(가장 중요함)
 - 연결
 - 자원 어댑터
 - JMS 대상
 - 메소드 바인딩

이 단계를 완료한 후 서비스를 다시 연결해야 합니다.

- 이 서비스가 동일한 모듈의 비즈니스 프로세스에 의해 호출되는 경우 적합한 비즈니스 프로세스 참조에서 이 가져오기로의 연결을 작성하십시오.
- 이 서비스가 다른 모듈의 비즈니스 프로세스에 의해 호출되는 경우 **SCA 바인딩**을 갖는 내보내기를 작성하고, 다른 모듈에서 이 내보내기를 해당 모듈의 어셈블리 편집기로 끌어 놓아서, 대응하는 **SCA 바인딩**을 갖는 내보내기를 작성하십시오. 적합한 비즈니스 프로세스 참조를 해당 가져오기로 연결하십시오.
- 어셈블리 다이어그램을 저장하십시오.

J2C-IMS 서비스 이주:

J2C-IMS 서비스를 EIS 바인딩을 갖는 내보내기 또는 웹 서비스 바인딩을 갖는 SCA 가져오기로 이주할 수 있습니다.

IMS 서비스를 위해 생성된 WebSphere Studio Application Developer Integration Edition 아티팩트를 사용하지 마십시오. WebSphere Integration Developer에서 사용 가능한 마법사를 사용하여 서비스를 다시 작성하고 응용프로그램을 수동으로 다시 연결해야 합니다.

주: 자동 빌드를 시작하거나 모듈을 수동으로 빌드하십시오.

다음 옵션이 있습니다.

주: 두 옵션 모두의 경우에 EIS 서비스에 의해 공개되는 인터페이스가 이전 5.1 인터페이스와 약간 다르기 때문에 BPEL 서비스가 이 IMS™ 서비스를 호출하는 경우 BPEL이 약간 변경되어야 함을 주의하십시오. 이를 수행하려면 BPEL 편집기를 열고 EIS 서비스에 해당하는 파트너 링크를 조정하고 위의 단계를 수행할 때 새 인터페이스(WSDL 파일)를 사용하십시오. EIS 서비스의 새 WSDL 인터페이스에 맞게 BPEL 활동을 필요한 대로 변경하십시오.

각 J2C-IMS 서비스 재연결 옵션에 대한 장단점:

각 J2C-IMS 서비스 재연결 옵션에 대한 장단점이 있습니다.

다음 목록은 두 가지 옵션 모두 또는 각 옵션에 대한 장단점에 대해 설명합니다.

- 첫 번째 옵션은 표준 SCA 컴포넌트를 사용하여 IMS 서비스를 호출합니다.
- 첫 번째 옵션은 다음의 몇 가지 제한사항을 갖습니다.
 - SDO 버전 1 스펙 API는 COBOL 또는 C 바이트 배열에 대한 액세스를 제공하지 않습니다. 이것은 IMS 다중 세그먼트에 대해 작업하는 고객에게 영향을 줍니다.
 - 직렬화에 대한 SDO 버전 1 스펙은 COBOL 재정의 또는 C 유니온을 지원하지 않습니다.
- 두 번째 옵션은 표준 JSR 109 접근을 사용하여 IMS 서비스에 연결합니다. 이 기능은 Rational Application Developer의 파트로서 사용 가능합니다.

IMS 서비스를 호출하는 SCA 가져오기 작성: 옵션 1:

DataObjects를 사용하여 IMS 시스템과 통신하기 위한 메시지데이터를 저장하는 EIS 바인딩을 갖는 SCA 가져오기를 작성할 수 있습니다.

IMS 서비스를 호출하기 위해 SCA 가져오기를 작성하려면 다음 단계를 수행하십시오.

1. 이 새로운 IMS 서비스를 보유할 새 비즈니스 통합 모듈 프로젝트를 작성하십시오.
2. EIS 서비스를 다시 작성하려면 파일 → 새로 작성 → 기타 → 비즈니스 통합 → 엔터프라이즈 서비스 발견으로 이동하십시오.
3. 이 마법사를 사용하여 EIS 시스템에서 서비스를 가져올 수 있습니다. 5.1에서 WSIF 기반 EIS 서비스를 작성한 WebSphere Studio Application Developer Integration Edition 마법사와 매우 비슷합니다. 이 마법사에서 새 J2C IMS 자원 어댑터를 가져올 수 있습니다. WebSphere Integration Developer가 설치된 디렉토리를 찾아서 자원 어댑터 → **ims15** → **imsico9102.rar**로 이동하십시오.

주: 특성 및 오퍼레이션 저장 패널 완료에 대한 자세한 정보는 Information Center를 참조하십시오. 엔터프라이즈 서비스 발견 마법사 중에 오퍼레이션을 추가할 때 오퍼레이션의 입력 또는 출력 데이터 유형에 대한 비즈니스 오브젝트를 작성할 수도 있습니다. 이 경우 WebSphere Studio Application Developer Integration Edition 마법사에서 사용한 C 또는 COBOL 소스 파일이 있어야 합니다. 사용자가 해당 위치의 소스 파일을 가리킬 수 있도록 이들 파일이 이전 서비스 프로젝트로 복사되었어야 합니다. 또한 독립 마법사인 파일 → 새로 작성 → 기타 → 비즈니스 통합 → 엔터프라이즈 데이터 발견을 사용하여 비즈니스 오브젝트를 가져올 수도 있습니다.

4. 마법사를 완료한 후 비즈니스 통합 Perspective를 열고 콘텐츠를 볼 수 있도록 모듈을 펼치십시오. 모듈의 데이터 유형 아래에 나열되는 새 비즈니스 오브젝트와 인터페이스 아래에 나열되는 새 인터페이스가 있어야 합니다.
5. 모듈 프로젝트(프로젝트와 동일한 이름을 가짐) 아래의 첫 번째 항목을 두 번 클릭하여 어셈블리 편집기를 여십시오. 가져오기가 캔버스 위에 존재하고, 이 가져오기가 EIS 바인딩을 갖고 있으며 방금 작성한 서비스를 표시해야 합니다.

해당 서비스를 이용자에게 공개하는 방법에 대한 지시사항은 “이주된 서비스를 액세스하기 위해 SCA 내보내기 작성” 섹션을 참조하십시오.

J2C 서비스에 웹 서비스 작성: 옵션 2:

J2C 웹 서비스를 작성할 수 있으며 서비스의 이용자가 SCA 컴포넌트인 경우 서비스를 IBM Web Service(SOAP/HTTP 또는 SOAP/JMS)로서 이용할 수 있습니다.

J2C 서비스에 웹 서비스를 작성하려면 다음 단계를 수행하십시오.

1. 파일 → 새로 작성 → **J2C** → **J2C Java Bean**을 클릭하여 J2C Java Bean을 작성하십시오.
2. **IMS Connector for Java**의 1.5 버전을 선택하고 다음을 클릭하십시오.
3. 관리 연결을 선택하고 JNDI 찾아보기 이름을 입력하십시오. 다음을 클릭하십시오.
4. 새 Java Bean에 대한 프로젝트, 패키지 및 이름을 지정하십시오. Bean은 인터페이스와 구현 클래스로서 구성됩니다. 다음을 클릭하십시오.
5. EIS에서 액세스하려는 각 기능 또는 서비스에 대해 Java 메소드를 추가하십시오. 추가 메소드는 나중에 스니펫 보기를 통해 Java 소스 편집기에서 추가할 수 있습니다. 추가... 단추를 클릭할 때 메소드 이름을 선택하고 다음을 클릭하십시오.

6. 이제 **찾아보기...**를 선택하여 기존 유형을 다시 사용하거나 **새로 작성...**을 사용하여 입력 및 출력 데이터 유형에 대해 CICS/IMS Java 데이터 바인딩 마법사(여기에서 COBOL 또는 C 소스 파일을 참조할 수 있음)를 실행할 수 있습니다.
7. Java 메소드 작성을 완료한 후 다음을 클릭하십시오.
8. 이 마법사의 나머지 단계를 완료하여 J2C Java Bean을 작성하십시오.
9. 파일 → 새로 작성 → **J2C** → **J2C Java Bean**에서 웹 페이지, 웹 서비스 또는 **EJB**를 클릭하여 웹 서비스를 작성하여 J2C Java Bean에 웹 서비스를 작성하십시오.
10. 마법사를 완료하십시오.

이 서비스의 이용자는 이제 이 마법사에 의해 생성되는 WSDL 서비스를 사용하여 IMS 서비스를 호출할 수 있습니다.

J2C-CICS ECI 서비스 이주:

J2C-CICS ECI 서비스를 EIS 바인딩을 갖는 내보내기 또는 웹 서비스 바인딩을 갖는 SCA 가져오기로 이주할 수 있습니다.

"J2C-IMS 서비스 프로젝트 이주" 주제의 지시사항을 따르십시오. 그러나 반드시 IMS RAR 파일 대신 다음 RAR 파일을 가져오십시오.

- WebSphere Integration Developer가 설치된 디렉토리를 찾아보고 자원 어댑터 → **cics15** → **cicseci.rar**로 이동하십시오.

두 번째 옵션에 따라서 J2C 웹 서비스를 작성하는 경우 J2C Java Bean 작성 마법사의 두 번째 패널에서 v1.5 **ECIResourceAdapter**를 선택하십시오.

또한 "J2C-IMS 서비스 이주" 주제도 참조하십시오.

J2C-CICS EPI 서비스 이주:

WebSphere Integration Developer에서 J2C-CICS EPI 서비스에 대한 직접적인 지원은 없습니다. SCA 모듈에서 이 서비스에 액세스하려면 **이용 시나리오**를 사용하여 이주해야 합니다.

이 서비스 유형을 WebSphere Integration Developer로 이주하기 위한 지시사항은 "서비스 이주를 위한 이용 시나리오" 주제를 참조하십시오.

J2C-HOD 서비스 이주:

WebSphere Integration Developer에서 J2C-HOD 서비스에 대한 직접적인 지원은 없습니다. SCA 모듈에서 이 서비스에 액세스하려면 **이용 시나리오**를 사용하여 이주해야 합니다.

이 서비스 유형을 WebSphere Integration Developer로 이주하기 위한 지시사항은 "서비스 이주를 위한 이용 시나리오" 주제를 참조하십시오.

변환기 서비스 이주:

가능한 경우 변환기 서비스를 SCA 데이터 맵 및 인터페이스 맵으로 이주할 수 있습니다. 또한 이용 시나리오를 사용하여 SCA 모듈에서 이 서비스에 액세스할 수도 있습니다.

데이터 맵 및 인터페이스 맵 컴포넌트는 버전 6.0의 새로운 사항입니다. 5.1의 변환기 서비스와 비슷한 기능을 제공하지만 완전한 XSL 변환 기능을 갖지는 않습니다. 변환기 서비스를 이들 컴포넌트 중 하나로 대체할 수 없는 경우, WebSphere Integration Developer에 변환기 서비스에 대한 직접 지원이 없으므로 이용 시나리오를 사용하여 이주해야 합니다. SCA 모듈에서 이 서비스에 액세스하려면 "서비스 이주를 위한 이용 시나리오" 섹션에 있는 단계를 수행하십시오.

서비스 이주에 대한 이용 시나리오:

WebSphere Studio Application Developer Integration Edition 서비스 유형에 대한 직접 상대방이 없는 경우, WebSphere Integration Developer에서 응용프로그램을 다시 디자인할 때 이전 WebSphere Studio Application Developer Integration Edition 서비스를 있는 그대로 이용하기 위해 이용 시나리오가 필요합니다.

다음은 이주 마법사를 호출하기 전에 WebSphere Studio Application Developer Integration Edition에서 수행할 단계입니다.

1. 이 클라이언트 프록시 코드를 보유할 새 Java 프로젝트를 작성하십시오. 5.1 스타일에서 생성한 메시지와 Java Bean 클래스가 서비스 프로젝트를 이주하는 자동 이주 마법사에 의해 생략되기 때문에 서비스 프로젝트에 이 클라이언트 프록시 코드를 넣지 마십시오.
2. WebSphere Studio Application Developer Integration Edition을 열고 변환기 바인딩과 서비스를 포함하는 WSDL 파일을 마우스 오른쪽 단추로 클릭하고 엔터프라이즈 서비스 → 서비스 프록시 생성을 선택하십시오. 사용자에게 작성할 프록시의 유형을 묻지만, **WSIF(Web Services Invocation Framework)**만 사용 가능합니다. 다음을 클릭하십시오.
3. 이제 작성할 서비스 프록시 Java 클래스의 패키지 및 이름을 지정할 수 있습니다. (현재 서비스 프로젝트에서 프록시를 작성합니다.) 다음을 클릭하십시오.
4. 이제 프록시 스타일을 지정하고, 클라이언트 스텝을 선택하고, 프록시에 포함시킬 원하는 오퍼레이션을 선택하고 완료를 클릭할 수 있습니다. 이는 WebSphere Studio Application Developer Integration Edition 서비스와 동일한 메소드를 공개하는 Java 클래스를 작성하며, 여기에서 Java 메소드에 대한 인수가 소스 WSDL 메시지의 파트입니다.

이제 WebSphere Integration Developer로 이주할 수 있습니다.

1. 클라이언트 프록시 Java 프로젝트를 새 작업공간에 복사하고 파일 → 가져오기 → 작업공간으로 기존 프로젝트로 이동하여 가져오십시오.
2. 이주 마법사를 사용하여 서비스 프로젝트를 가져오십시오. 그러면 WSDL 메시지, PortTypes, 바인딩 및 서비스가 WebSphere Studio Application Developer Integration Edition에 생성되는 비즈니스 통합 모듈이 작성됩니다.
3. 비즈니스 통합 Perspective에서 해당 콘텐츠를 볼 수 있도록 모듈을 펼치십시오. 모듈 프로젝트(프로젝트와 동일한 이름을 가짐) 아래의 첫 번째 항목을 두 번 클릭하여 어셈블리 편집기를 여십시오.

4. 사용자 정의 Java 컴포넌트를 작성하려면, 모듈 프로젝트 아래에서 인터페이스를 펼치고, WebSphere Studio Application Developer Integration Edition의 이 변환기 서비스에 대해 생성된 WSDL 인터페이스를 선택하십시오.
5. 이 인터페이스를 어셈블리 편집기로 끌어서 놓으십시오. 작성할 컴포넌트의 유형을 선택하는 대화 상자가 열립니다. 구현 유형이 없는 컴포넌트를 선택하고 확인을 클릭하십시오.
6. 일반 컴포넌트가 어셈블리 다이어그램에 표시됩니다. 해당 컴포넌트를 선택한 후 특성 보기로 이동하십시오.
7. 설명 탭에서 컴포넌트의 이름과 표시 이름을 보다 구체적인 것으로 변경할 수 있습니다. (이 경우에 EJB의 이름 등으로 이름을 지정하지만, "JavaMed"와 같은 접미어를 첨부하십시오. 이것이 WebSphere Studio Application Developer Integration Edition에서 변환기 서비스에 대해 생성되는 WSDL 인터페이스와 변환기 클라이언트 프록시의 Java 인터페이스 사이를 중개하는 Java 컴포넌트가 되기 때문입니다.)
8. 세부사항 탭에서 어셈블리 편집기에 끌어서 놓은 하나의 인터페이스를 갖는 이 컴포넌트를 볼 수 있습니다.
9. 어셈블리 편집기로 돌아가서 방금 작성한 컴포넌트를 마우스 오른쪽 단추로 클릭하고 구현 생성... → Java를 선택하십시오. 그런 다음 Java 구현이 생성될 패키지를 선택하십시오. 그러면 복합 유형은 `commonj.sdo.DataObject`인 오브젝트로 표시되고 단순 유형은 동등한 Java 오브젝트로 표시되는 SCA 프로그래밍 모델에 따라 WSDL 인터페이스를 준수하는 스켈레톤 Java 서비스가 작성됩니다.

이제 생성된 Java 구현 클래스에서 “//TODO” 태그가 표시되는 위치에 코드를 채워야 합니다. 다음 두 옵션이 있습니다.

1. 원래의 Java 클래스에서 이 클래스로 로직을 이동하여 새 데이터 구조를 채택하십시오.
2. 이 생성된 Java 클래스 내부에 이전 Java 클래스의 개인용 인스턴스를 작성하고 다음을 수행하는 코드를 작성하십시오.
 - a. 생성된 Java 구현 클래스의 모든 매개변수를 이전 Java 클래스가 예상하는 매개변수로 변환
 - b. 변환된 매개변수로 이전 Java 클래스의 개인용 인스턴스 호출
 - c. 이전 Java 클래스의 리턴값을 생성된 Java 구현 메소드에 의해 선언되는 리턴값으로 변환

위의 옵션을 완료한 후 클라이언트 프록시를 다시 연결해야 합니다. 어떤 “참조”도 없어야 하므로, Java 컴포넌트의 인터페이스를 다시 연결하면 됩니다.

- 이 서비스가 동일한 모듈의 비즈니스 프로세스에 의해 호출되는 경우 적합한 비즈니스 프로세스 참조에서 이 Java 컴포넌트 인터페이스로의 연결을 작성하십시오.
- 이 서비스가 다른 모듈의 비즈니스 프로세스에 의해 호출되는 경우 SCA 바인딩을 갖는 내보내기를 작성하고, 다른 모듈에서 이 내보내기를 해당 모듈의 어셈블리 편집기로 끌어 놓아서, 대응하는 SCA 바인딩을 갖는 내보내기를 작성하십시오. 적합한 비즈니스 프로세스 참조를 해당 가져오기로 연결하십시오.
- 이 서비스를 외부에 공개하기 위해 WebSphere Studio Application Developer Integration Edition에서 출력한 경우, 서비스 재출력 방법에 대한 지시사항은 “이주된 서비스를 액세스하기 위해 SCA 내보내기 작성” 섹션을 참조하십시오.

이주된 서비스에 액세스하기 위해 SCA 내보내기 작성:

WebSphere Studio Application Developer Integration Edition 서비스 프로젝트에 배치 코드를 생성한 모든 서비스에 대한 SCA 모델에 따라 이주된 서비스를 외부 이용자에게 사용 가능하도록 만들려면 SCA 내보내기를 작성해야 합니다. 여기에는 응용프로그램 외부의 클라이언트에 의해 호출되는 모든 서비스가 포함됩니다.

WebSphere Studio Application Developer Integration Edition에서 BPEL 프로세스나 기타 서비스 WSDL을 마우스 오른쪽 단추로 클릭하고 엔터프라이즈 서비스 → 배치 코드 생성을 선택했으면, 아래 수동 이주 단계를 수행해야 합니다. WebSphere Integration Developer는 모든 배치 옵션을 저장하는 점에서 WebSphere Studio Application Developer Integration Edition과 다르다는 것을 참고하십시오. 프로젝트를 빌드할 때, 배치 코드는 생성된 EJB 및 웹 프로젝트에서 자동으로 갱신되므로 더 이상 수동의 배치 코드 생성 옵션이 제공되지 않습니다.

다섯 개의 바인딩 옵션이 BPEL 배치 코드 생성 마법사의 파트너 섹션용 인터페이스 아래에서 제공됩니다. 다음 인바운드 BPEL 서비스 이주 정보에서는 WebSphere Studio Application Developer Integration Edition에서 선택한 배치 바인딩 유형에 기반하여 작성하도록 내보내기 유형 및 특성에 대한 세부사항을 제공합니다.

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

EJB 및 EJB 프로세스 바인딩 이주:

EJB 및 EJB 프로세스 바인딩을 권장 SCA 생성으로 이주할 수 있습니다.

WebSphere Studio Application Developer Integration Edition에서 이 바인딩 유형을 통해 클라이언트가 EJB를 호출하여 BPEL 프로세스 또는 기타 서비스 유형과 통신할 수 있습니다. 이 바인딩 유형은 마이크로 프로세스에 대한 옵션이 아니라는 점을 참고하십시오. 기타 바인딩 유형에서 생성된 EJB를 내부적으로 사용하는 경우 이 유형은 항상 선택됩니다.

생성된 EJB의 JNDI 이름은 BPEL 이름, 대상 이름 공간 및 올바른 시작 시간 소인을 결합하여 자동으로 생성됩니다. 예를 들어 이 속성은 설명 및 서버 콘텐츠 탭에서 BPEL 편집기의 BPEL 프로세스 특성을 점검하여 찾을 수 있습니다.

표 3. 생성된 이름 공간

프로세스 이름	MyService
대상 이름 공간	http://www.example.com/process87787141/
올바른 시작 시간	Jan 01 2003 02:03:04

이 예제에 대해 생성된 이름 공간은 com/example/www/process87787141/MyService20030101T020304입니다.

WebSphere Studio Application Developer Integration Edition에서 EJB 바인딩이 배치 유형으로 선택되었을 때 제공되는 옵션은 없었습니다.

WebSphere Studio Application Developer Integration Edition 프로세스 바인딩의 이주에는 4가지 옵션이 있습니다. 서비스에 액세스하는 클라이언트 유형에 따라 아래의 수행할 이주 옵션을 결정하십시오.

주: 수동 이주 단계를 완료한 후, 클라이언트도 새 프로그래밍 모델로 이주해야 합니다. 다음 클라이언트 유형에 대한 해당 주제를 참조하십시오.

표 4. 클라이언트 이주에 대한 추가 정보

클라이언트 유형	추가 정보 참조 위치
생성된 세션 Bean을 호출하는 EJB 클라이언트. 이러한 클라이언트는 호출할 때 BPEL 오퍼레이션에 해당하는 EJB 메소드를 호출합니다.	EJB 클라이언트 이주
EJB 프로세스 바인딩을 사용하는 WSIF 클라이언트	EJB 프로세스 바인딩 클라이언트 이주
일반 BPC(business process choreographer) EJB API	BPC(business process choreographer) 일반 EJB API 클라이언트 이주
일반 BPC(business process choreographer) 메시징 API	BPC(business process choreographer) 일반 메시징 API 클라이언트 이주
동일 모듈에 있는 다른 BPEL 프로세스	해당 없음: 어셈블리 편집기를 사용하여 BPEL 컴포넌트를 함께 연결
다른 모듈에 있는 다른 BPEL 프로세스	해당 없음: 참조하는 모듈에서 SCA 바인딩을 사용한 가져오기를 작성하고, 아래 옵션 1에서 작성하는 SCA 바인딩을 사용한 내보내기를 가리키도록 바인딩을 구성하십시오.

비즈니스 프로세스가 모듈 밖으로(서비스 참조를 통해) 자신에 대한 참조를 전달하는 경우 항상 아래의 옵션 1에 따라서(항상 이들 옵션의 둘 이상을 수행할 수 있음) 비즈니스 프로세스에 대한 SCA 바인딩을 갖는 내보내기를 작성해야 함을 주의해야 합니다. 내보내기가 모듈의 기본 내보내기로 표시되어야 하므로 모듈당 하나의 비즈니스 프로세스만이 해당 서비스 참조를 모듈 밖으로 전달할 수 있습니다. 이는 다음에서와 같이 내보내기의 "default"라는 속성에 대해 "true"를 지정하여 수행됩니다.

Default endpoint reference

비즈니스 통합 보기에서 내보내기를 마우스 오른쪽 단추로 클릭하고 연결 프로그램을 선택한 후 문서 편집기를 선택하여 수동으로 이 비즈니스 프로세스의 내보내기를 기본값으로 표시해야 합니다.

EJB 및 EJB 프로세스 바인딩에 대한 이주 옵션 1:

WebSphere Studio Application Developer Integration Edition EJB 프로세스 바인딩을 위한 첫 번째 이주 옵션은 동일 모듈에 있는 다른 컴포넌트에 액세스할 수 있는 비즈니스 프로세스를 작성하는 것입니다.

어셈블리 편집기에서, 다른 컴포넌트를 BPEL 컴포넌트에 연결하십시오.

1. 도구 모음에서 연결 항목을 선택하십시오.
2. 기타 컴포넌트를 클릭하여 연결 소스로 선택하십시오.
3. BPEL SCA 컴포넌트를 클릭하여 연결의 대상으로 선택하십시오.

4. 어셈블리 다이어그램을 저장하십시오.

EJB 및 EJB 프로세스 바인딩에 대한 이주 옵션 2:

WebSphere Studio Application Developer Integration Edition EJB 프로세스 바인딩을 위한 두 번째 이주 옵션은 기타 SCA 모듈 및 클라이언트에 액세스할 수 있는 비즈니스 프로세스를 작성하는 것입니다.

주: 이 단계는 일반 BPC(Business Process choreographer) API가 비즈니스 프로세스를 호출하는 데 사용될 경우 필수입니다.

SCA 바인딩을 사용한 내보내기에서는 기타 SCA 모듈에서 액세스할 수 있는 SCA 컴포넌트를 작성합니다. SCA 바인딩을 사용한 내보내기를 작성하려면 다음을 수행하십시오.

1. 이주 마법사가 작성한 모듈을 어셈블리 편집기로 여십시오.
2. WebSphere Studio Application Developer Integration Edition에서 EJB 바인딩이 생성된 각 BPEL 프로세스 인터페이스에 대해 SCA 바인딩을 사용하여 내보내기를 작성하십시오.
 - a. 어셈블리 편집기에서 BPEL 컴포넌트를 마우스 오른쪽 단추로 클릭하십시오.
 - b. 내보내기...를 선택하십시오.
 - c. SCA 바인딩을 선택하십시오.
 - d. 프로세스에 대해 다중의 인터페이스가 있는 경우, 이 바인딩 유형으로 내보낼 인터페이스를 선택하십시오.
 - e. SCA 내보내기를 작성한 후, 어셈블리 편집기에서 내보내기를 선택하고 특성 보기에서 설명 콘텐츠 분할창을 선택하십시오. 내보내기 이름 및 설명이 나열되고 필요에 따라 수정할 수 있습니다.
 - f. 어셈블리 다이어그램을 저장하십시오.

EJB 및 EJB 프로세스 바인딩에 대한 이주 옵션 3:

WebSphere Studio Application Developer Integration Edition EJB 프로세스 바인딩을 위한 세 번째 이주 옵션은 비SCA 엔티티(예를 들면 JSP 또는 Java 클라이언트)가 액세스할 수 있는 모듈을 작성하는 것입니다.

독립형 참조에서는 외부 클라이언트가 액세스할 수 있는 SCA 컴포넌트를 작성합니다. 독립형 참조를 작성하려면 다음을 수행하십시오.

1. 이주 마법사가 작성한 모듈에 대한 어셈블리 편집기를 여십시오.
2. WebSphere Studio Application Developer Integration Edition에서 EJB 바인딩이 생성된 각 BPEL 프로세스 인터페이스에 대해 독립형 참조를 작성하십시오.
 - a. 도구 모음에서 독립형 참조 항목을 선택하십시오.
 - b. 어셈블리 편집기 캔버스를 클릭하여 독립형 참조 SCA 엔티티를 작성하십시오.
 - c. 도구 모음에서 연결 항목을 선택하십시오.
 - d. 독립형 참조 엔티티를 클릭하여 연결의 소스로 선택하십시오.
 - e. BPEL SCA 컴포넌트를 클릭하여 연결의 대상으로 선택하십시오.
 - f. 일치하는 참조가 소스 노드에서 작성됩니다. 계속하시겠습니까? 경고가 표시되면 확인을 클릭하십시오.

- g. 방금 작성한 독립형 참조 엔티티를 선택하고 특성 보기에서 설명 콘텐츠 분할창을 선택하십시오.
- h. 참조 링크를 펼치고 방금 작성한 참조를 선택하십시오. 참조 이름 및 설명이 나열되고 필요에 따라 수정할 수 있습니다.
- i. 프로세스에 대해 다중의 인터페이스가 있는 경우, 이 바인딩 유형으로 내보낼 인터페이스를 선택하십시오.
- j. 어셈블리 다이어그램을 저장하십시오.

EJB 및 EJB 프로세스 바인딩에 대한 이주 옵션 4:

WebSphere Studio Application Developer Integration Edition EJB 프로세스 바인딩을 위한 네 번째 이주 옵션은 웹 서비스 클라이언트에서 액세스할 수 있는 비즈니스 프로세스를 작성하는 것입니다.

웹 서비스 바인딩을 사용한 내보내기에서는 외부 웹 서비스 클라이언트에서 액세스할 수 있는 SCA 컴포넌트를 작성합니다. 웹 서비스 바인딩을 사용한 내보내기를 작성하려면 다음을 수행하십시오.

1. 이주 마법사가 작성한 모듈을 어셈블리 편집기로 여십시오.
2. WebSphere Studio Application Developer Integration Edition에서 EJB 바인딩이 생성된 각 BPEL 프로세스 인터페이스에 대해 SCA 바인딩을 사용하여 내보내기를 작성하십시오.
 - a. 어셈블리 편집기에서 BPEL 컴포넌트를 마우스 오른쪽 단추로 클릭하십시오.
 - b. 내보내기...를 선택하십시오.
 - c. 웹 서비스 바인딩을 선택하십시오.
 - d. 프로세스에 대해 다중의 인터페이스가 있는 경우, 이 바인딩 유형으로 내보낼 인터페이스를 선택하십시오.
 - e. **soap/http** 또는 **soap/jms** 전송을 선택하십시오.
 - f. 웹 서비스 내보내기를 작성한 후, 어셈블리 편집기에서 내보내기를 선택하고 특성 보기에서 설명 콘텐츠 분할창을 선택하십시오. 내보내기 이름 및 설명이 나열되고 필요에 따라 수정할 수 있습니다.
 - g. 어셈블리 다이어그램을 저장하십시오.

JMS 및 JMS 프로세스 바인딩 이주:

JMS 및 JMS 프로세스 바인딩을 권장 SCA 생성으로 이주할 수 있습니다.

WebSphere Studio Application Developer Integration Edition에서, 이러한 바인딩 유형을 통해 클라이언트는 MDB에 메시지를 전송하여 BPEL 프로세스 또는 기타 서비스와 통신할 수 있습니다. 이 바인딩 유형은 장기 실행 프로세스의 경우 옵션이 아니며 항상 선택된다는 점을 주목하십시오. 실제로 이 바인딩 유형은 장기 실행 프로세스의 요청-응답 인터페이스에서 허용되는 유일한 바인딩 유형입니다. 기타 서비스 유형의 경우 MDB가 생성되며 적당한 서비스를 호출합니다.

JMS 바인딩에서 사용하는 JNDI 이름은 BPEL 이름, 대상 이름 공간 및 올바른 시작 시간 소인을 결합하여 구성됩니다.

WebSphere Studio Application Developer Integration Edition에서 BPEL 프로세스의 배치 유형으로 JMS 바인딩을 선택한 경우 다음 옵션이 제공됩니다.

- **JNDI 연결 팩토리** - 기본값은 jms/BPECF입니다. 이 이름은 대상 비즈니스 프로세스 컨테이너의 대기열 연결 팩토리에 해당하는 JNDI 이름입니다.
- **JNDI 대상 대기열** - 기본값은 jms/BPEIntQueue입니다. 이 이름은 대상 비즈니스 프로세스 컨테이너의 내부 대기열에 해당하는 JNDI 이름입니다.
- **JNDI 제공자 URL:** 서버 제공 또는 사용자 정의 - 주소를 입력해야 합니다. 기본값은 iiop://localhost:2809입니다.

WebSphere Studio Application Developer Integration Edition JMS 프로세스 바인딩의 이주에는 5가지 옵션이 있습니다. 서비스에 액세스하는 클라이언트 유형에 따라 아래의 수행할 이주 옵션을 결정하십시오.

주: 수동 이주 단계를 완료한 후, 클라이언트도 새 프로그래밍 모델로 이주해야 합니다. 다음 클라이언트 유형에 대한 해당 주제를 참조하십시오.

표 5. 클라이언트 이주에 대한 추가 정보

클라이언트 유형	추가 정보 참조 위치
JMS 프로세스 바인딩을 사용하는 WSIF 클라이언트	BPC(Business Process Choreographer) 일반 메시징 API 클라이언트 및 JMS 프로세스 바인딩 클라이언트 이주
일반 BPC(business process choreographer) EJB API	BPC(business process choreographer) 일반 EJB API 클라이언트 이주
일반 BPC(Business Process Choreographer) 메시징 API 비즈니스 이주	BPC(business process choreographer) 일반 메시징 API 클라이언트 이주
동일 모듈에 있는 다른 BPEL 프로세스	해당 없음: 어셈블리 편집기를 사용하여 BPEL 컴포넌트를 함께 연결
다른 모듈에 있는 다른 BPEL 프로세스	해당 없음: 참조하는 모듈에서 'SCA 바인딩을 사용한 가져오기'를 작성하고 아래 옵션 1에서 작성하는 'SCA 바인딩을 사용한 내보내기'를 지적하도록 바인딩을 구성하십시오.

비즈니스 프로세스가 모듈 밖으로(서비스 참조를 통해) 자신에 대한 참조를 전달하는 경우 항상 아래의 옵션 1에 따라서(항상 이들 옵션의 둘 이상을 수행할 수 있음) 비즈니스 프로세스에 대한 SCA 바인딩을 갖는 내보내기를 작성해야 함을 주의해야 합니다. 내보내기가 모듈의 기본 내보내기로 표시되어야 하므로 모듈당 하나의 비즈니스 프로세스만이 해당 서비스 참조를 모듈 밖으로 전달할 수 있습니다. 이는 다음에서와 같이 내보내기의 "default"라는 속성에 대해 "true"를 지정하여 수행됩니다.

Default endpoint reference

비즈니스 통합 보기에서 내보내기를 마우스 오른쪽 단추로 클릭하고 연결 프로그램을 선택한 후 문서 편집기를 선택하여 수동으로 이 비즈니스 프로세스의 내보내기를 기본값으로 표시해야 합니다.

JMS 및 JMS 프로세스 바인딩에 대한 이주 옵션 1:

WebSphere Studio Application Developer Integration Edition JMS 프로세스 바인딩을 위한 첫 번째 이주 옵션은 동일 모듈에 있는 다른 컴포넌트에 액세스할 수 있는 비즈니스 프로세스를 작성하는 것입니다.

어셈블리 편집기에서, 다른 컴포넌트를 BPEL 컴포넌트에 연결하십시오.

1. 도구 모음에서 연결 항목을 선택하십시오.
2. 기타 컴포넌트를 클릭하여 연결 소스로 선택하십시오.
3. **BPEL SCA** 컴포넌트를 클릭하여 연결의 대상으로 선택하십시오.
4. 어셈블리 다이어그램을 저장하십시오.

JMS 및 JMS 프로세스 바인딩에 대한 이주 옵션 2:

WebSphere Studio Application Developer Integration Edition JMS 프로세스 바인딩을 위한 두 번째 이주 옵션은 기타 SCA 모듈 및 클라이언트에 액세스할 수 있는 비즈니스 프로세스를 작성하는 것입니다.

SCA 바인딩을 사용한 내보내기에서는 기타 SCA 모듈에서 액세스할 수 있는 SCA 컴포넌트를 작성합니다. SCA 바인딩을 사용한 내보내기를 작성하려면 다음을 수행하십시오.

1. 이주 마법사가 작성한 모듈을 어셈블리 편집기로 여십시오.
2. WebSphere Studio Application Developer Integration Edition에서 JMS 바인딩이 생성된 각 BPEL 프로세스 인터페이스에 대해 SCA 바인딩을 사용하여 내보내기를 작성하십시오.
 - a. 어셈블리 편집기에서 BPEL 컴포넌트를 마우스 오른쪽 단추로 클릭하십시오.
 - b. 내보내기...를 선택하십시오.
 - c. **SCA** 바인딩을 선택하십시오.
 - d. 프로세스에 대해 다중의 인터페이스가 있는 경우, 이 바인딩 유형으로 내보낼 인터페이스를 선택하십시오.
 - e. SCA 내보내기를 작성한 후, 어셈블리 편집기에서 내보내기를 선택하고 특성 보기에서 설명 콘텐츠 분할창을 선택하십시오. 내보내기 이름 및 설명이 나열되고 필요에 따라 수정할 수 있습니다.
 - f. 어셈블리 다이어그램을 저장하십시오.

JMS 및 JMS 프로세스 바인딩에 대한 이주 옵션 3:

WebSphere Studio Application Developer Integration Edition JMS 프로세스 바인딩을 위한 세 번째 이주 옵션은 비SCA 엔티티(예를 들면 JSP 또는 Java 클라이언트)가 액세스할 수 있는 비즈니스 프로세스를 작성하는 것입니다.

독립형 참조에서는 외부 클라이언트가 액세스할 수 있는 SCA 컴포넌트를 작성합니다. 독립형 참조를 작성하려면 다음을 수행하십시오.

1. 이주 마법사가 작성한 모듈을 어셈블리 편집기로 여십시오.
2. WebSphere Studio Application Developer Integration Edition에서 JNB 바인딩이 생성된 각 BPEL 프로세스 인터페이스에 대해 독립형 참조를 작성하십시오.
 - a. 도구 모음에서 독립형 참조 항목을 선택하십시오.
 - b. 어셈블리 편집기 캔버스를 클릭하여 독립형 참조 SCA 엔티티를 작성하십시오.
 - c. 도구 모음에서 연결 항목을 선택하십시오.
 - d. 독립형 참조 엔티티를 클릭하여 연결의 소스로 선택하십시오.

- e. **BPEL SCA** 컴포넌트를 클릭하여 연결의 대상으로 선택하십시오.
- f. 일치하는 참조가 소스 노트에서 작성됩니다. 계속하시겠습니까? 경고가 표시되면 확인을 클릭하십시오.
- g. 방금 작성한 독립형 참조 엔티티를 선택하고 특성 보기에서 설명 콘텐츠 분할창을 선택하십시오.
- h. 참조 링크를 펼치고 방금 작성한 참조를 선택하십시오. 참조 이름 및 설명이 나열되고 필요에 따라 수정할 수 있습니다.
- i. 프로세스에 대해 다중의 인터페이스가 있는 경우, 이 바인딩 유형으로 내보낼 인터페이스를 선택하십시오.
- j. 어셈블리 다이어그램을 저장하십시오.

JMS 및 JMS 프로세스 바인딩에 대한 이주 옵션 4:

WebSphere Studio Application Developer Integration Edition JMS 프로세스 바인딩을 위한 네 번째 이주 옵션은 웹 서비스 클라이언트에서 액세스할 수 있는 비즈니스 프로세스를 작성하는 것입니다.

웹 서비스 바인딩을 사용한 내보내기에서는 외부 웹 서비스 클라이언트에서 액세스할 수 있는 SCA 컴포넌트를 작성합니다. 웹 서비스 바인딩을 사용한 내보내기를 작성하려면 다음을 수행하십시오.

1. 이주 마법사가 작성한 모듈을 어셈블리 편집기로 여십시오.
2. WebSphere Studio Application Developer Integration Edition에서 JMS 바인딩이 생성된 각 BPEL 프로세스 인터페이스에 대해 SCA 바인딩을 사용하여 내보내기를 작성하십시오.
 - a. 어셈블리 편집기에서 BPEL 컴포넌트를 마우스 오른쪽 단추로 클릭하십시오.
 - b. 내보내기...를 선택하십시오.
 - c. 웹 서비스 바인딩을 선택하십시오.
 - d. 프로세스에 대해 다중의 인터페이스가 있는 경우, 이 바인딩 유형으로 내보낼 인터페이스를 선택하십시오.
 - e. **soap/http** 또는 **soap/jms** 전송을 선택하십시오.
 - f. 웹 서비스 내보내기를 작성한 후, 어셈블리 편집기에서 내보내기를 선택하고 특성 보기에서 설명 콘텐츠 분할창을 선택하십시오. 내보내기 이름 및 설명이 나열되고 필요에 따라 수정할 수 있습니다.
 - g. 어셈블리 다이어그램을 저장하십시오.

JMS 및 JMS 프로세스 바인딩에 대한 이주 옵션 5:

WebSphere Studio Application Developer Integration Edition JMS 프로세스 바인딩을 위한 다섯 번째 이주 옵션은 JMS 클라이언트가 액세스할 수 있는 비즈니스 프로세스를 작성하는 것입니다.

JMS 바인딩을 사용한 내보내기에서는 외부 JMS 클라이언트가 액세스할 수 있는 SCA 컴포넌트를 작성합니다. JMS 바인딩을 사용한 내보내기를 작성하려면 다음을 수행하십시오.

1. BPEL 서비스의 경우 5.1 JMS 프로세스 바인딩은 표준 5.1 JMS 바인딩과 완전히 다르기 때문에 새 쿼리 자원을 작성 및 참조해야 합니다. 비 BPEL 서비스의 경우 생성된 EJB 프로젝트의 **ejbModule/META-INF** 폴더 아래에 있는 해당 패키지에서 **JMSBinding.wsdl** 및 **JMSService.wsdl**이라는 이름의 WSDL 파일을 검색하고 여기서 캡처한 바인딩 및 서비스 정보를 검사하는 방법으로 WebSphere Studio Application

Developer Integration Edition 5.1에서 JMS 배치 코드에 대해 선택한 값을 찾을 수 있습니다. 바인딩에서 텍스트 또는 오브젝트 메시지가 사용되었는지 여부와 사용자 정의 데이터 형식 바인딩이 사용되었는지 여부를 확인할 수 있습니다. 사용된 항목이 있는 경우 6.0 JMS 바인딩으로 내보내기에 대해서도 사용자 정의 데이터 바인딩을 작성하는 것이 좋습니다. 서비스에서 초기 컨텍스트 팩토리, JNDI 연결 팩토리 이름, JNDI 대상 이름, 대상 스타일(큐) 등을 찾을 수 있습니다.

2. 이주 마법사가 작성한 모듈을 어셈블리 편집기로 여십시오.
3. 어셈블리 편집기에서 BPEL 컴포넌트를 마우스 오른쪽 버튼으로 클릭하여 WebSphere Studio Application Developer Integration Edition에서 JMS 바인딩을 생성한 각 BPEL 프로세스 인터페이스에 대해 JMS 바인딩을 사용한 내보내기를 작성하십시오.
4. 내보내기...를 선택하십시오.
5. JMS 바인딩을 선택하십시오.
6. 프로세스에 대해 다중의 인터페이스가 있는 경우, 이 바인딩 유형으로 내보낼 인터페이스를 선택하십시오.
7. 다음 패널(JMS 내보내기 바인딩 속성)에서 JMS 메시징 도메인을 선택하십시오. 이 속성을 지점간으로 정의하십시오.
8. 비즈니스 오브젝트 및 JMS 메시지 간 데이터 직렬화 방식을 선택하고 다음 값(대개는 XML인 텍스트는 런타임에 독립적이고 이중 시스템 간에 서비스 통합을 가능하게 하므로 오브젝트 대신 텍스트를 선택하도록 권장)을 입력하십시오.
 - a. 텍스트의 경우, 기본 JMS 기능 선택기를 사용하도록 선택하거나 FunctionSelector 구현 클래스의 완전한 이름을 입력하십시오.
 - b. 오브젝트의 경우, 기본 JMS 기능 선택기를 사용하도록 선택하거나 FunctionSelector 구현 클래스의 완전한 이름을 입력하십시오.
 - c. 사용자 제공의 경우, JMSSDataBinding 구현 클래스의 완전한 이름을 입력하십시오. 응용프로그램이 JMS Import 바인딩에서 아직 사용할 수 없는 JMS 헤더 특성에 액세스할 필요가 있는 경우 사용자 제공을 선택해야 합니다. 이 경우 표준 JMS 데이터 바인딩 **com.ibm.websphere.sca.jms.data.JMSSDataBinding**을 확장하는 사용자 정의 데이터 바인딩 클래스를 작성하고 JMSMessage에 직접 액세스할 수 있는 사용자 정의 코드를 추가해야 합니다. 그런 다음 이 필드에 대한 사용자 정의 클래스의 이름을 제공합니다. 아래 링크에서 "컴포넌트 가져오기 및 내보내기 바인딩 작성 및 수정"의 JMS 예제를 참조하십시오.
 - d. 사용자 제공의 경우 기본 JMS 기능 선택기를 사용하도록 선택하거나 FunctionSelector 구현 클래스의 완전한 이름을 입력하십시오.
9. JMS 바인딩을 사용한 내보내기를 작성한 후, 어셈블리 편집기에서 내보내기를 선택하고 특성 보기에서 설명 콘텐츠 분할창을 선택하십시오. 내보내기 이름 및 설명이 나열되고 필요에 따라 수정할 수 있습니다.
10. 바인딩 콘텐츠 분할창을 선택하면 좀 더 많은 옵션이 표시됩니다.
11. 어셈블리 다이어그램을 저장하십시오.

IBM Web Service 바인딩(SOAP/JMS) 이주:

BPEL 프로세스 또는 기타 서비스 유형에 대한 IBM Web Service 바인딩(SOAP/JMS)은 권장 SCA 생성으로 이주할 수 있습니다.

WebSphere Studio Application Developer Integration Edition에서, 이 바인딩 유형을 통해 클라이언트가 IBM Web Service를 호출하여 BPEL 프로세스 또는 기타 서비스 유형과 통신할 수 있습니다. 여기서 통신 프로토콜은 JMS이고 메시지는 SOAP 인코딩 규칙을 준수합니다.

다음은 5.1 BPEL 서비스에 대해 IBM Web Service(SOAP/JMS)를 생성할 때 사용되는 규칙의 예입니다. 생성된 IBM Web Service의 JNDI 이름은 BPEL 이름, 대상 이름 공간 및 올바른 시작 시간소인 뿐만 아니라 인터페이스 이름(배치 코드가 생성되는 WSDL 포트 유형)을 결합하여 구성됩니다. 예를 들어, 이 속성은 설명 및 서버 콘텐츠 탭에서 BPEL 편집기의 BPEL 프로세스 특성을 점검하여 찾을 수 있습니다.

표 6. 생성된 이름 공간

프로세스 이름	MyService
대상 이름 공간	http://www.example.com/process87787141/
올바른 시작 시간	Jan 01 2003 02:03:04
인터페이스	ProcessPortType

이 예제에 대해 생성된 이름 공간은 com/example/www/process87787141/MyService20030101T020304_ProcessPortTypePT입니다.

WebSphere Studio Application Developer Integration Edition에서 IBM Web Service 바인딩(SOAP/JMS)을 BPEL 프로세스 또는 기타 서비스 유형에 대한 배치 유형으로 선택할 경우, 다음 옵션이 제공됩니다.

- 문서 스타일의 경우, 기본값은 문서 / 기타 옵션은 **RPC**입니다.
- 문서 사용의 경우, 기본값은 리터럴 / 기타 옵션은 인코드입니다.
- JNDI 제공업체 URL의 경우, 서버 제공 또는 사용자 정의입니다(주소를 입력해야 하며, 기본값은 iiop://localhost:2809임).
- 대상 스타일의 경우, 기본값은 대기열 / 기타 옵션은 주제입니다.
- JNDI 연결 팩토리의 경우, 기본값은 **jms/qcf**입니다. (이는 생성된 MDB 대기열의 대기열 연결 팩토리에 대한 JNDI 이름임)
- JNDI 대상 대기열의 경우, 기본값은 **jms/queue**입니다. (이는 생성된 MDB 대기열의 JNDI 이름임)
- MDB 리스너 포트의 경우, 기본값은 <서비스 프로젝트 이름>**MdbListenerPort**입니다.

IBM Web Service SOAP/JMS 바인딩 및 서비스를 지정하는 WSDL 파일은 생성된 EJB 프로젝트에서 작성되지만 서비스 프로젝트 자체에서는 작성되지 않습니다. 이는 IBM Web Service 클라이언트 코드가 변경되어서는 안되는 경우 해당 파일을 수동으로 찾아 이를 비즈니스 통합 모듈 프로젝트로 복사해야 한다는 의미입니다. 기본적으로 이 WSDL 파일은 ejbModule/META-INF/wsdl/<business process name>_<business process interface port type name>_JMS.wsdl의 EJB 프로젝트에서 작성됩니다.

비즈니스 프로세스 인터페이스의 WSDL PortType 및 메시지는 실제로 서비스 프로젝트에서 정의된 기존 WSDL PortType 및 메시지를 참조하지 않고 대신 이 WSDL 파일로 복사됩니다.

IBM Web Service 클라이언트 코드가 이주 후 변경되지 않은 채로 남아 있어야 하는 경우, 이 파일에 있는 정보는 아래 수동 이주 단계에 필요합니다.

WebSphere Studio Application Developer Integration Edition SOAP/JMS 프로세스 바인딩의 이주에는 두 가지 옵션이 있습니다. 클라이언트를 SCA 프로그래밍 모델로 이주할 것인지 또는 웹 서비스 클라이언트로서 남겨둘 것인지를 결정해야 합니다.

주: 수동 이주 단계를 완료한 후, 클라이언트도 새 프로그래밍 모델로 이주해야 합니다. 다음 클라이언트 유형에 대한 해당 주제를 참조하십시오.

표 7. 클라이언트 이주에 대한 추가 정보

클라이언트 유형	추가 정보 참조 위치
IBM Web Service 클라이언트	IBM Web Service(SOAP/JMS) 클라이언트 이주

IBM Web Service 바인딩(SOAP/JMS)을 위한 이주 옵션 1:

WebSphere Studio Application Developer Integration Edition SOAP/JMS 바인딩을 위한 첫 번째 이주 옵션은 서비스를 웹 서비스 클라이언트에서 액세스할 수 있도록 만드는 것입니다.

웹 서비스 바인딩을 사용한 내보내기에서는 외부 웹 서비스 클라이언트에서 액세스할 수 있는 SCA 컴포넌트를 작성합니다. 웹 서비스 바인딩을 사용한 내보내기를 작성하려면 다음을 수행하십시오.

1. 이주 마법사가 작성한 모듈을 어셈블리 편집기로 여십시오.
2. WebSphere Studio Application Developer Integration Edition에서 IBM Web Service(SOAP/JMS) 바인딩이 생성된 각 서비스 인터페이스에 대해 SCA 바인딩을 갖는 내보내기를 작성하십시오.
 - a. 어셈블리 편집기에서 SCA 컴포넌트를 마우스 오른쪽 단추로 클릭하십시오.
 - b. 내보내기...를 선택하십시오.
 - c. 웹 서비스 바인딩을 선택하십시오.
 - d. 컴포넌트에 대해 다중의 인터페이스가 있는 경우, 이 바인딩 유형으로 내보낼 인터페이스를 선택하십시오.
 - e. **soap/jms** 전송을 선택하십시오.
3. 웹 서비스 내보내기를 작성한 후, 어셈블리 편집기에서 내보내기를 선택하고, 특성 보기에서 설명 콘텐츠를 분할창을 선택하십시오. 내보내기 이름 및 설명이 나열되고 필요에 따라 수정할 수 있습니다.
4. 어셈블리 다이어그램을 저장하십시오.
5. 바인딩 콘텐츠 분할창을 선택하면 IBM Web Service WSDL 바인딩 및 서비스가 모듈의 프로젝트 폴더에서 직접 생성되었음을 알 수 있습니다. 그 이름은 *component-that-was-exported* Export WSDL PortType name Jms_Service.wsdl로 지정됩니다. 해당 파일을 조사하는 경우 문서/리터럴 랩핑된 바인딩이 6.0에서 선호되는 스타일이기 때문에 기본적으로 사용됨을 알 수 있습니다. 이는 IBM Web Service 클라이언트가 서비스를 호출할 때 사용하는 WSDL입니다.
6. 클라이언트 코드 보존이 바람직한 경우 다음 단계를 수행하여 새 웹 서비스 바인딩 및 서비스를 생성하십시오.

- a. ejbModule/META-INF/wsdl/business process name/business process interface port type nameJMS.wsdl의 5.1 생성 EJB 프로젝트에서 비즈니스 통합 모듈 프로젝트로 5.1 WSDL 파일을 복사하십시오.
- b. 파일을 복사하고 모듈을 다시 빌드한 후 웹 서비스가 사용하는 XML 스키마 유형, WSDL 메시지 및 WSDL 포트 유형이 5.1에 있는 IBM Web Service WSDL 파일에서 중복되기 때문에 오류 메시지가 표시될 수 있습니다. 이를 수정하려면 IBM Web Service 바인딩/서비스 WSDL에서 중복 정의를 삭제하고 해당 위치에 실제 인터페이스 WSDL에 대한 WSDL 가져오기를 추가하십시오. 참고: WebSphere Studio Application Developer Integration Edition이 IBM Web Service 배치 코드를 생성했을 때 일부의 경우에 스키마 정의를 수정했음을 주의해야 합니다. 이는 IBM Web Service WSDL을 사용하는 기존 클라이언트에 대해 불일치를 유발할 수 있습니다. 예를 들어 원래의 스키마 정의가 규정되지 않은 경우, "elementFormDefault" 스키마 속성이 IBM Web Service WSDL에서 생성된 인라인 스키마에서 "qualified"로 설정되었습니다. 이는 런타임 중에 WWS3047E: Error: Cannot deserialize element 오류가 생성되도록 합니다.
- c. 방금 비즈니스 통합 모듈에 복사한 이 WSDL 파일을 마우스 오른쪽 단추로 클릭하고 연결 프로그램을 선택한 후 **WSDL 편집기**를 선택하십시오.
- d. 소스 탭으로 이동하십시오. 이 파일에 정의된 모든 WSDL PortTypes 및 메시지를 삭제하십시오.
- e. 다음 오류가 표시됩니다. '<binding>' 바인딩에 대해 '<portType>' 포트 유형이 정의되지 않았습니다. 이를 수정하려면 그래프 탭의 WSDL 편집기에서 가져오기 섹션을 마우스 오른쪽 단추로 클릭하고 가져오기 추가를 선택하십시오.
- f. 일반 탭의 특성 보기에서 위치 필드의 오른쪽에 있는 ... 단추를 클릭하십시오. WSDL 메시지 및 포트 유형 정의가 위치하는 인터페이스 WSDL을 찾아보고 확인을 클릭하여 인터페이스 WSDL을 서비스/바인딩 WSDL로 가져오십시오.
- g. WSDL 파일을 저장하십시오.
- h. 프로젝트를 새로 고치기/다시 빌드하십시오. 비즈니스 통합 Perspective로 전환하십시오. 어셈블리 편집기에서 모듈의 어셈블리 다이어그램을 여십시오.
- i. 프로젝트 탐색기 보기에서 이주 중인 모듈을 펼치고 웹 서비스 포트 논리 카테고리를 펼치십시오. 바인딩/서비스 WSDL 목록에 존재하는 포트가 보여야 합니다. 어셈블리 편집기로 끌어서 놓으십시오.
- j. 웹 서비스 바인딩을 갖는 내보내기 작성을 선택하고 적합한 포트 이름을 선택하십시오. 기존 웹 서비스 클라이언트가 변경할 필요가 없도록 이전 바인딩/서비스를 사용하는 내보내기가 작성됩니다. 방금 어셈블리 편집기에서 작성한 내보내기를 선택하고 특성 보기로 이동하는 경우, 바인딩 탭에서 5.1 포트 및 서비스 이름이 채워졌음을 알아야 합니다.
- k. 모든 변경을 저장하십시오.
- l. 응용 프로그램을 배치하기 직전에 5.1 서비스 주소와 일치하도록 생성된 웹 프로젝트의 구성을 변경할 수 있습니다. (이 파일을 다시 생성하게 하는 SCA 모듈을 변경할 때마다 변경해야 합니다.) 재사용 중인 IBM Web Service WSDL 서비스 정의를 보는 경우, 다음과 같은 5.1 클라이언트가 코드화된 서비스 주소가 표시됩니다. <wsdlsoap:address location="http://localhost:9080/MyServiceWeb/services/MyServicePort"/>

- m. 6.0 생성 웹 프로젝트 아티팩트가 이 이전 서비스 주소와 일치하게 하기 위해 생성된 웹 프로젝트의 배치 설명자를 수정해야 합니다. WebSphere Integration Developer에서 배치 설명자를 열고, Servlet 탭에서 해당 내보내기에 대한 기존 URL 맵핑과 매우 비슷하고 동일한 Servlet 이름을 갖지만 다른 URL 패턴을 갖는 추가 URL 맵핑을 추가하십시오.
- n. 또한 이 웹 프로젝트의 컨텍스트 루트를 수정하여 원래 서비스 주소의 컨텍스트 루트와 일치하도록 해야 하는 경우(이 예제에서 컨텍스트 루트는 "MyServiceWeb"), 이 웹 프로젝트가 있는 J2EE Enterprise Application에 대한 배치 설명자를 열고 해당 웹 모듈의 컨텍스트 루트가 이전 서비스 주소의 컨텍스트 루트와 일치하도록 변경할 수 있습니다. 다음 오류가 표시될 수 있지만 무시할 수 있습니다. CHKJ3017E: Web Project: <WEB PROJ NAME> is mapped to an invalid Context root: <NEW CONTEXT ROOT> in EAR Project: <APP NAME>.

IBM Web Service 바인딩(SOAP/JMS)을 위한 이주 옵션 2:

WebSphere Studio Application Developer Integration Edition SOAP/JMS 프로세스 바인딩을 위한 두 번째 이주 옵션은 비SCA 엔티티(예를 들면 JSP 또는 Java 클라이언트)에 액세스할 수 있는 비즈니스 프로세스를 작성하는 것입니다.

독립형 참조에서는 외부 클라이언트가 액세스할 수 있는 SCA 컴포넌트를 작성합니다. 독립형 참조를 작성하려면 다음을 수행하십시오.

1. 이주 마법사가 작성한 모듈을 어셈블리 편집기로 여십시오.
2. WebSphere Studio Application Developer Integration Edition에서 IBM Web Service(SOAP/JMS) 바인딩이 생성된 각 BPEL 프로세스 인터페이스에 대해 독립형 참조를 작성하십시오.
 - a. 도구 모음에서 독립형 참조 항목을 선택하십시오.
 - b. 어셈블리 편집기 캔버스를 클릭하여 독립형 참조 SCA 엔티티를 작성하십시오.
 - c. 도구 모음에서 연결 항목을 선택하십시오.
 - d. 독립형 참조 엔티티를 클릭하여 연결의 소스로 선택하십시오.
 - e. BPEL SCA 컴포넌트를 클릭하여 연결의 대상으로 선택하십시오.
 - f. 일치하는 참조가 소스 노드에서 작성됩니다. 계속하시겠습니까? 경고가 표시되면 확인을 클릭하십시오.
 - g. 방금 작성한 독립형 참조 엔티티를 선택하고 특성 보기에서 설명 콘텐츠 분할창을 선택하십시오.
 - h. 참조 링크를 펼치고 방금 작성한 참조를 선택하십시오. 참조 이름 및 설명이 나열되고 필요에 따라 수정할 수 있습니다.
 - i. 프로세스에 대해 다중의 인터페이스가 있는 경우, 이 바인딩 유형으로 내보낼 인터페이스를 선택하십시오.
 - j. 어셈블리 다이어그램을 저장하십시오.

IBM Web Service 바인딩(SOAP/HTTP) 이주:

BPEL 프로세스 또는 기타 서비스 유형에 대한 IBM Web Service 바인딩(SOAP/HTTP)은 권장 SCA 생성으로 이주할 수 있습니다.

WebSphere Studio Application Developer Integration Edition에서, 이 바인딩 유형을 통해 클라이언트가 IBM Web Service를 호출하여 BPEL 프로세스 또는 기타 서비스 유형과 통신할 수 있습니다. 여기서 통신 프로토콜은 HTTP이고 메시지는 SOAP 인코딩 규칙을 준수합니다.

다음은 5.1 BPEL 서비스에 대해 IBM Web Service(SOAP/HTTP)를 생성할 때 사용되는 규칙의 예입니다. 생성된 IBM Web Service의 JNDI 이름은 BPEL 이름, 대상 이름 공간 및 올바른 시작 시간소인 뿐만 아니라 인터페이스 이름(배치 코드가 생성되는 WSDL 포트 유형)을 결합하여 구성됩니다. 예를 들어, 이 속성은 설명 및 서버 콘텐츠 탭에서 BPEL 편집기의 BPEL 프로세스 특성을 점검하여 찾을 수 있습니다.

표 8. 생성된 이름 공간

프로세스 이름	MyService
대상 이름 공간	http://www.example.com/process87787141/
올바른 시작 시간	Jan 01 2003 02:03:04
인터페이스	ProcessPortType

이 예제에 대해 생성된 이름 공간은 com/example/www/process87787141/MyService20030101T020304_ProcessPortTypePT입니다.

WebSphere Studio Application Developer Integration Edition에서 IBM Web Service 바인딩(SOAP/HTTP)을 BPEL 프로세스 또는 기타 서비스 유형에 대한 배치 유형으로 선택할 경우, 다음 옵션이 제공됩니다.

- 문서 스타일의 경우, 기본값은 문서 / 기타 옵션은 **RPC**입니다.
- 문서 사용의 경우, 기본값은 리터럴 / 기타 옵션은 인코드입니다.
- 라우터 주소의 경우, 기본값은 http://localhost:9080입니다.

IBM Web Service SOAP/HTTP 바인딩 및 서비스를 지정하는 WSDL 파일은 생성된 웹 및 EJB 프로젝트에서 작성되지만 서비스 프로젝트 자체에서는 작성되지 않습니다. 이는 IBM Web Service 클라이언트 코드가 변경되어서는 안되는 경우 해당 파일을 수동으로 찾아 이를 비즈니스 통합 모듈 프로젝트로 복사해야 한다는 의미입니다. 기본적으로 이 WSDL 파일은 WebContent/WEB-INF/wsdl/<business process name>_<business process interface port type name>_HTTP.wsdl의 웹 프로젝트에서 작성됩니다.

비즈니스 프로세스 인터페이스의 WSDL PortType 및 메시지는 실제로 서비스 프로젝트에서 정의된 기존 WSDL PortType 및 메시지를 참조하지 않고 대신 이 WSDL 파일로 복사됩니다.

IBM Web Service 클라이언트 코드가 이주 후 변경되지 않은 채로 남아 있어야 하는 경우, 이 파일에 있는 정보는 아래 수동 이주 단계에 필요합니다.

WebSphere Studio Application Developer Integration Edition SOAP/HTTP 프로세스 바인딩의 이주에는 두 가지 옵션이 있습니다. 클라이언트를 SCA 프로그래밍 모델로 이주할 것인지 또는 웹 서비스 클라이언트로서 남겨둘 것인지를 결정해야 합니다.

주: 수동 이주 단계를 완료한 후, 클라이언트도 새 프로그래밍 모델로 이주해야 합니다. 다음 클라이언트 유형에 대한 해당 주제를 참조하십시오.

표 9. 클라이언트 이주에 대한 추가 정보

클라이언트 유형	추가 정보 참조 위치
IBM Web Service 클라이언트	IBM Web Service(SOAP/HTTP) 클라이언트 이주

IBM Web Service(SOAP/HTTP) 바인딩을 위한 이주 옵션 1:

WebSphere Studio Application Developer Integration Edition SOAP/HTTP 프로세스 바인딩을 위한 첫 번째 이주 옵션은 웹 서비스 클라이언트에 액세스할 수 있는 비즈니스 프로세스를 작성하는 것입니다.

웹 서비스 바인딩을 사용한 내보내기에서는 외부 웹 서비스 클라이언트에서 액세스할 수 있는 SCA 컴포넌트를 작성합니다. 웹 서비스 바인딩을 사용한 내보내기를 작성하려면 다음을 수행하십시오.

1. 이주 마법사가 작성한 모듈에 대한 어셈블리 편집기를 여십시오.
2. 어셈블리 편집기에서 BPEL 컴포넌트를 마우스 오른쪽 버튼으로 눌러 WebSphere Studio Application Developer Integration Edition에서 IBM Web Service(SOAP/HTTP) 바인딩을 생성한 각 BPEL 프로세스 인터페이스에 대해 SCA 바인딩을 사용한 내보내기를 작성하십시오.
3. 내보내기...를 선택하십시오.
4. 웹 서비스 바인딩을 선택하십시오.
5. 컴포넌트에 대해 다중의 인터페이스가 있는 경우, 이 바인딩 유형으로 내보낼 인터페이스를 선택하십시오.
6. **soap/http** 전송을 선택하십시오.
7. 웹 서비스 내보내기를 작성한 후, 어셈블리 편집기에서 내보내기를 선택하고, 특성 보기에서 설명 콘텐츠를 분할창을 선택하십시오. 내보내기 이름 및 설명이 나열되고 필요에 따라 수정할 수 있습니다.
8. 어셈블리 다이어그램을 저장하십시오.
9. 클라이언트 코드 보존이 바람직한 경우 다음 단계를 수행하여 새 웹 서비스 바인딩 및 서비스를 생성하십시오.
 - a. `ejbModule/META-INF/wsdl/business process name/business process interface port type name_HTTP.wsdl`의 5.1 생성 EJB 프로젝트에서 비즈니스 통합 모듈 프로젝트로 5.1 WSDL 파일을 복사하십시오.
 - b. 파일을 복사하고 모듈을 다시 빌드한 후 웹 서비스가 사용하는 XML 스키마 유형, WSDL 메시지 및 WSDL 포트 유형이 5.1에 있는 IBM Web Service WSDL 파일에서 중복되기 때문에 오류 메시지가 표시될 수 있습니다. 이를 수정하려면 IBM Web Service 바인딩/서비스 WSDL에서 중복 정의를 삭제하고 해당 위치에 실제 인터페이스 WSDL에 대한 WSDL 가져오기를 추가하십시오. 참고: WebSphere Studio Application Developer Integration Edition이 IBM Web Service 배치 코드를 생성했을 때 일부의 경우에 스키마 정의를 수정했음을 주의해야 합니다. 이는 IBM Web Service WSDL을 사용하는 기존 클라이언트에 대해 불일치를 유발할 수 있습니다. 예를 들어 원래의 스키마 정의가 규정되지 않은 경우, "elementFormDefault" 스키마 속성이 IBM Web Service WSDL에서 생성된 인라인 스키마에서 "qualified"로 설정되었습니다. 이는 런타임 중에 WWS3047E: Error: Cannot deserialize element 오류가 생성되도록 합니다.

- c. 방금 비즈니스 통합 모듈에 복사한 이 WSDL 파일을 마우스 오른쪽 단추로 클릭하고 **연결 프로그램**을 선택한 후 **WSDL 편집기**를 선택하십시오.
- d. 소스 탭으로 이동하십시오. 이 파일에 정의된 모든 WSDL PortTypes 및 메시지를 삭제하십시오.
- e. 다음 오류가 표시됩니다. '<binding>' 바인딩에 대해 '<portType>' 포트 유형이 정의되지 않았습니다. 이를 수정하려면 그래프 탭의 WSDL 편집기에서 가져오기 섹션을 마우스 오른쪽 단추로 클릭하고 **가져오기 추가**를 선택하십시오.
- f. 일반 탭의 특성 보기에서 위치 필드의 오른쪽에 있는 ... 단추를 클릭하십시오. WSDL 메시지 및 포트 유형 정의가 위치하는 인터페이스 WSDL을 찾아보고 **확인**을 클릭하여 인터페이스 WSDL을 서비스/바인딩 WSDL로 가져오십시오.
- g. WSDL 파일을 저장하십시오.
- h. 프로젝트를 새로 고치기/다시 빌드하십시오. 비즈니스 통합 Perspective로 전환하십시오. 어셈블리 편집기에서 모듈의 어셈블리 다이어그램을 여십시오.
- i. 프로젝트 탐색기 보기에서 아주 중인 모듈을 펼치고 웹 서비스 포트 논리 카테고리를 펼치십시오. 바인딩/서비스 WSDL 목록에 존재하는 포트가 보여야 합니다. 어셈블리 편집기로 끌어서 놓으십시오.
- j. 웹 서비스 바인딩을 갖는 내보내기 작성을 선택하고 적합한 포트 이름을 선택하십시오. 기존 웹 서비스 클라이언트가 변경할 필요가 없도록 이전 바인딩/서비스를 사용하는 내보내기가 작성됩니다. 방금 어셈블리 편집기에서 작성한 내보내기를 선택하고 특성 보기로 이동하는 경우, 바인딩 탭에서 5.1 포트 및 서비스 이름이 채워졌음을 알아야 합니다.
- k. 모든 변경을 저장하십시오.
- l. 응용 프로그램을 배치하기 직전에 5.1 서비스 주소와 일치하도록 생성된 웹 프로젝트의 구성을 변경할 수 있습니다. (이 파일을 다시 생성하게 하는 SCA 모듈을 변경할 때마다 변경해야 합니다.) 재사용 중인 IBM Web Service WSDL 서비스 정의를 보는 경우, 다음과 같은 5.1 클라이언트가 코드화된 서비스 주소가 표시됩니다. <wsdlsoap:address location="http://localhost:9080/MyServiceWeb/services/MyServicePort"/>
- m. 6.0 생성 웹 프로젝트 아티팩트가 이 이전 서비스 주소와 일치하게 하기 위해 생성된 웹 프로젝트의 배치 설명자를 수정해야 합니다. WebSphere Integration Developer에서 배치 설명자를 열고, Servlet 탭에서 해당 내보내기에 대한 기존 URL 맵핑과 매우 비슷하고 동일한 Servlet 이름을 갖지만 다른 URL 패턴을 갖는 추가 URL 맵핑을 추가하십시오.
- n. 또한 이 웹 프로젝트의 컨텍스트 루트를 수정하여 원래 서비스 주소의 컨텍스트 루트와 일치하도록 해야 하는 경우(이 예제에서 컨텍스트 루트는 "MyServiceWeb"), 이 웹 프로젝트가 있는 J2EE Enterprise Application에 대한 배치 설명자를 열고 해당 웹 모듈의 컨텍스트 루트가 이전 서비스 주소의 컨텍스트 루트와 일치하도록 변경할 수 있습니다. 다음 오류가 표시될 수 있지만 무시할 수 있습니다. CHKJ3017E: Web Project: <WEB PROJ NAME> is mapped to an invalid Context root: <NEW CONTEXT ROOT> in EAR Project: <APP NAME>.

IBM Web Service(SOAP/HTTP) 바인딩을 위한 이주 옵션 2:

WebSphere Studio Application Developer Integration Edition SOAP/HTTP 프로세스 바인딩을 위한 두 번째 이주 옵션은 비SCA 엔티티(예를 들면 JSP 또는 Java 클라이언트)에 액세스할 수 있는 비즈니스 프로세스를 작성하는 것입니다.

독립형 참조에서는 외부 클라이언트가 액세스할 수 있는 SCA 컴포넌트를 작성합니다. 독립형 참조를 작성하려면 다음을 수행하십시오.

1. 이주 마법사가 작성한 모듈을 어셈블리 편집기로 여십시오.
2. WebSphere Studio Application Developer Integration Edition에서 IBM Web Service(SOAP/HTTP) 바인딩이 생성된 각 BPEL 프로세스 인터페이스에 대해 독립형 참조를 작성하십시오.
 - a. 도구 모음에서 독립형 참조 항목을 선택하십시오.
 - b. 어셈블리 편집기 캔버스를 클릭하여 독립형 참조 SCA 엔티티를 작성하십시오.
 - c. 도구 모음에서 연결 항목을 선택하십시오.
 - d. 독립형 참조 엔티티를 클릭하여 연결의 소스로 선택하십시오.
 - e. SCA 컴포넌트를 클릭하여 연결의 대상으로 선택하십시오.
 - f. 일치하는 참조가 소스 노드에서 작성됩니다. 계속하시겠습니까? 경고가 표시되면 확인을 클릭하십시오.
 - g. 방금 작성한 독립형 참조 엔티티를 선택하고 특성 보기에서 설명 콘텐츠 분할창을 선택하십시오.
 - h. 참조 링크를 펼치고 방금 작성한 참조를 선택하십시오. 참조 이름 및 설명이 나열되고 필요에 따라 수정할 수 있습니다.
 - i. 프로세스에 대해 다중의 인터페이스가 있는 경우, 이 바인딩 유형으로 내보낼 인터페이스를 선택하십시오.
 - j. 어셈블리 다이어그램을 저장하십시오.

Apache 웹 서비스 바인딩(SOAP/HTTP) 이주:

BPEL 프로세스 또는 기타 서비스 유형에 대한 Apache 웹 서비스 바인딩(SOAP/HTTP)은 권장 SCA 생성으로 이주할 수 있습니다.

WebSphere Studio Application Developer Integration Edition에서는 이 바인딩 유형을 통해 클라이언트가 Apache 웹 서비스를 호출하여 BPEL 프로세스 또는 기타 서비스 유형과 통신할 수 있습니다.

WebSphere Studio Application Developer Integration Edition에서 Apache 웹 서비스 바인딩을 BPEL 프로세스 또는 기타 서비스 유형에 대한 배치 유형으로 선택하면 다음 옵션이 제공됩니다.

- 문서 스타일의 경우, **RPC**입니다(기타 옵션은 사용할 수 없음).
- SOAP 조치의 경우, **URN:WSDL PortType** 이름입니다.
- 주소의 경우, **http://localhost:9080/서비스 프로젝트 이름Web/servlet/rpcrouter**입니다.
- 인코딩 사용의 경우, 기본값은 예입니다. (예인 경우, 인코딩 유형은 **http://schemas.xmlsoap.org/soap/encoding/**으로 설정됩니다.)

Apache SOAP 바인딩 및 서비스를 지정하는 WSDL 파일은 서비스 프로젝트에서 작성됩니다. 기본적으로 `<business process name>_<business process interface port type name>_SOAP.wsdl` 이름으로 랩핑되는 서비스와 동일한 디렉토리에서 작성됩니다. 비즈니스 프로세스 인터페이스의 WSDL PortType 및 메시지는 이 바인딩 및 서비스에서 직접 사용합니다. 이주 후, 버전 6에서 생성할 새 WSDL에 있는 동일한 이름 공간, 포트 및 서비스 이름을 제외하고 다른 것에 대해 이 WSDL을 사용해서는 안 됩니다.

WebSphere Studio Application Developer Integration Edition 웹 서비스 프로세스 바인딩을 이주하는 데에는 두 가지 옵션이 있습니다. 클라이언트를 SCA 프로그래밍 모델로 이주할 것인지 또는 IBM Web Service 프로그래밍 모델로서 남겨둘 것인지를 결정해야 합니다. 6 SCA 프로그래밍 모델에서는 더 이상 Apache 웹 서비스(SOAP/HTTP) 바인딩과 동등한 바인딩이 없습니다.

IBM Web Service 엔진을 사용하려면 이 Apache 웹 서비스를 이주시켜야 합니다. 해당 이주를 수행하고 IBM Web Service(SOAP/HTTP)를 작성하는 방법에 대한 지시사항은 "IBM Web Service(SOAP/HTTP) 바인딩 이주" 주제를 참조하십시오.

SCA 프로그래밍 모델로 이주:

WebSphere Studio Application Developer Integration Edition 서비스와 상호작용하는 자유 양식 Java 코드의 경우 이 섹션에서는 WSIF 프로그래밍 모델을 새 SCA 프로그래밍 모델(응용프로그램에서의 데이터 플로우가 Eclipse SDO(Service Data Object)에 저장됨)로 이주하는 방법을 설명합니다. 이 섹션은 또한 가장 공통적인 클라이언트 유형을 새 프로그래밍 모델로 수동으로 이주하는 방법을 설명합니다.

Java 스니펫을 포함하는 모든 BPEL 프로세스에 대해 이 섹션은 이전 Java 스니펫 API에서 응용프로그램을 통해 이동하는 데이터가 Eclipse SDO(Service Data Object)에 저장되는 새 Java 스니펫 API로 이주하는 방법을 설명합니다. 가능한 경우 스니펫은 이주 마법사를 통해 자동으로 이주되지만 이주 마법사가 전체를 이주할 수 없는 스니펫도 있습니다. 즉, 이주를 완료하려면 수동 단계가 필요합니다.

다음은 프로그래밍 모델 변경사항의 요약입니다.

V5.1 프로그래밍 모델

1. WSIF 및 WSDL 기반
2. 서비스를 위해 프록시 생성
3. 유형에 대한 Bean 및 형식 핸들러

V6.0 프로그래밍 모델(보다 Java 중심적)

1. doclet 태그를 갖는 SDO 기반 SCA 서비스
2. 서비스를 위한 인터페이스 바인딩
3. 유형에 대한 SDO 및 데이터 바인딩

WSIFMessage API 호출을 SDO API로 이주:

다음 섹션은 응용프로그램을 통해 이동하는 데이터가 강하게 유형화된, 생성된 인터페이스를 갖는 WSIFMessage 오브젝트로 표시되는 이전 WebSphere Business Integration Server Foundation 버전 5.1 프로그래밍 모델로

부터 데이터가 SDO(Service Data Object)로서 표시되고 강하게 유형화된 인터페이스가 생성되지 않는 새 WebSphere Process Server 버전 6.0 프로그래밍 모델로 이주하는 방법에 대해 자세히 설명합니다.

표 10. SDO API로 WSIFMessage API 호출 이주를 위한 변경 및 솔루션

변경	솔루션
WSIFMessage 기반 래퍼 클래스가 더 이상 WSDL 메시지 유형에 대해 생성되지 않으며, 복합 스키마 유형에 대해 Java Bean 헬퍼 클래스가 생성되지 않습니다.	SCA 서비스와 상호작용하는 코드를 작성할 때 일반 SDO API를 사용하여 응용프로그램을 통해 이동하는 데이터를 보유하는 <code>commonj.sdo.DataObject</code> 메시지를 조작해야 합니다. 하나의 단순 입력 파트를 갖는 WSDL 메시지 정의는 이제 실제 데이터 주위에 래퍼를 갖는 대신 파트를 직접 표시하는 단순 Java 유형으로 표시됩니다. 단일 메시지 파트가 복합 유형인 경우 데이터는 복합 유형 정의를 준수하는 <code>DataObject</code> 로서 표시됩니다. 다중 파트를 갖는 WSDL 메시지 정의는 이제 모든 메시지 파트에 대한 특성을 갖는 <code>DataObject</code> 에 대응하는 반면, <code>complexType</code> 은 <code>getDataObject</code> 및 <code>setDataObject</code> 메소드를 통해 액세스할 수 있는 상위 <code>DataObject</code> 의 'reference-type' 특성으로 표시됩니다.
WSIFMessage 파트에 대한 강하게 유형화된 Getter 메소드 및 생성된 Java Bean은 사용해서는 안됩니다.	<code>DataObject</code> 특성을 가져오려면 약하게 유형화된 SDO API를 사용해야 합니다.
BPEL 변수의 메시지 파트에 대해 강하게 유형화된 Setter 메소드는 더 이상 사용할 수 없습니다.	<code>DataObject</code> 특성을 설정하려면 약하게 유형화된 SDO API를 사용해야 합니다.
WSIFMessage 특성에 대해 약하게 유형화된 Getter 메소드는 더 이상 사용되지 않습니다.	<code>DataObject</code> 특성을 설정하려면 약하게 유형화된 SDO API를 사용해야 합니다.
WSIFMessage 특성에 대해 약하게 유형화된 Setter 메소드는 더 이상 사용되지 않습니다.	<code>DataObject</code> 특성을 설정하려면 약하게 유형화된 SDO API를 사용해야 합니다.
모든 WSIFMessage API 호출은 가능하면 SDO API로 이주되어야 합니다.	가능하면 호출을 동등한 SDO API 호출로 이주하십시오. 불가능한 경우 로직을 다시 디자인하십시오.

WebSphere Business Integration Server Foundation 클라이언트 코드 이주:

이 섹션은 WebSphere Business Integration Server Foundation 5.1 서비스 유형에 대해 가능했던 다양한 클라이언트 유형을 이주하는 방법을 보여줍니다.

EJB 클라이언트 이주:

이 주제에서는 EJB 인터페이스를 사용하여 서비스를 호출하는 클라이언트를 이주하는 방법을 보여 줍니다.

1. 이주된 모듈에서 새 모듈의 어셈블리 편집기로 SCA 바인딩이 있는 내보내기를 끌어서 놓으십시오. 그러면 SCA 바인딩이 있는 가져오기가 작성됩니다. 클라이언트에서 이 가져오기를 참조로 얻으려면 독립형 참조를 작성해야 합니다.
2. 팔레트에서 독립형 참조 항목을 선택하십시오. 어셈블리 편집기 캔버스를 한 번 클릭하여 이 새 모듈의 새 독립형 참조를 작성하십시오.
3. 연결 도구를 선택하고 서비스 참조를 클릭한 후 가져오기를 클릭하십시오.
4. 일치하는 참조가 소스 노드에서 작성되었음을 경고하면 확인을 클릭하십시오.
5. 다음과 같은 질문이 표시됩니다: **Java 클라이언트가 이 참조와 함께 Java 인터페이스를 사용하는 것이 더 쉽습니다. WSDL 참조를 호환 가능한 Java 참조로 변환하시겠습니까?**

- a. 클라이언트가 이 서비스를 찾고 Java 인터페이스를 사용하여 호출하기 위해 Java 클래스로서 캐스트하려는 경우 예로 응답하십시오. 새 Java 인터페이스에서는 WSDL PortType 이름(인터페이스 패키지가 WSDL WSDL PortType의 이름 공간에서 파생됨)을 사용합니다. WSDL PortType에 정의된 각 오퍼레이션에서 정의한 메소드가 있으며 각 WSDL 메시지 파트는 인터페이스 메소드의 인수로 표시됩니다.
 - b. 클라이언트가 이 서비스를 찾고 호출 오퍼레이션을 일반 SCA 서비스로 사용하여 호출하기 위해 일반 com.ibm.websphere.sca.Service 인터페이스를 사용하려는 경우 아니오를 응답하십시오.
6. 해당하는 경우 어셈블리 편집기에서 독립형 참조 컴포넌트를 선택하여 보다 유용한 이름으로 독립형 참조 이름을 바꾸십시오. 특성 보기, 세부사항 탭으로 이동하여 방금 작성한 참조를 끌어서 놓은 후 해당 참조를 작성하여 이름을 수정하십시오. com.ibm.websphere.sca.ServiceManager 인스턴스의 locateService 메소드를 호출할 때 클라이언트가 이 이름을 사용해야 하기 때문에 이 참조에 대해 선택하는 이름을 기억하십시오.
 7. 저장을 클릭하여 어셈블리 다이어그램을 저장하십시오.

서버에서 실행 중인 이주된 EJB 모듈에 액세스하도록 클라이언트는 해당 로컬 클래스 경로에 이 새 모듈을 배치합니다.

다음은 유형 "CustomerInfo"의 서비스의 경우 클라이언트 코드의 모습을 보여줍니다.

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

클라이언트는 반드시 메시지 생성 방식을 변경해야 합니다. 메시지가 WSIFMessage 클래스를 기반으로 했으나 이제는 commonj.sdo.DataObject 클래스를 기반으로 해야 합니다.

EJB 프로세스 바인딩 클라이언트 이주:

이 주제에서는 WSIF EJB 프로세스 바인딩을 사용하여 BPEL 서비스에 액세스하는 클라이언트를 이주하는 방법을 보여 줍니다.

EJB 프로세스 바인딩을 사용하여 비즈니스 프로세스를 호출한 클라이언트는 이제 SCA API를 사용하여 서비스를 호출하거나(이주된 비즈니스 프로세스가 SCA 바인딩을 갖는 내보내기를 가져야 함) IBM Web Service 클라이언트 API를 사용하여 서비스를 호출해야 합니다. (이주된 비즈니스 프로세스는 웹 서비스 바인딩을 갖는 내보내기를 가져야 합니다.)

해당 클라이언트 생성에 대한 자세한 정보는 "EJB 클라이언트 이주", "IBM Web Service (SOAP/JMS) Client", 또는 "IBM Web Service(SOAP/HTTP) 클라이언트" 주제를 참조하십시오.

IBM Web Service(SOAP/JMS) 클라이언트 이주:

이 주제에서는 웹 서비스 API(SOAP/JMS)를 사용하여 서비스를 호출하는 클라이언트를 이주하는 방법을 보여 줍니다.

이주 중에 기존 클라이언트에 대한 이주가 필요없습니다. 수동으로 생성된 웹 프로젝트를 수정(새 Servlet 맵핑을 작성)해야 하며 때로는 엔터프라이즈 응용프로그램 배치 설명자에 있는 웹 프로젝트의 컨텍스트 루트를 수정하여 서비스를 WebSphere Business Integration Server Foundation에 출력된 정확하게 동일한 주소에 출력해야 할 수 있음을 주의하십시오. "IBM Web Service 바인딩(SOAP/JMS) 이주" 주제를 참조하십시오.

WSIF 또는 RPC 클라이언트 프록시가 생성될 수 있는 5.1에서와는 달리, 6.0에서는 RPC가 WSIF API에서 6.0이 선호하는 API이기 때문에 도구가 RPC 클라이언트 생성만을 지원함을 주의해야 합니다.

참고: WebSphere Integration Developer에서 새 클라이언트 프록시를 생성하려면 WebSphere Process Server 또는 WebSphere Application Server가 설치되어 있어야 합니다.

1. WebSphere Process Server 또는 WebSphere Application Server가 설치되어 있어야 합니다.
2. 자원 또는 Java Perspective에서, 웹 서비스 바인딩을 갖는 내보내기에 대응하는 WSDL 파일을 선택한 후 마우스 오른쪽 단추로 클릭하고 웹 서비스 → 클라이언트 생성을 선택하십시오(이 마법사는 5.1 마법사와 매우 유사합니다).
3. 클라이언트 프록시 유형에 대해 **Java** 프록시를 선택하고 다음을 클릭하십시오.
4. WSDL의 위치를 채워야 합니다. 다음을 클릭하십시오.
5. 다음으로 적당한 옵션을 선택하여 웹 서비스 런타임 및 서버, J2EE 버전, 클라이언트 유형(Java, EJB, 웹, 응용프로그램 클라이언트)를 포함한 클라이언트 환경 구성을 지정해야 합니다. 다음을 클릭하십시오.
6. 남은 단계를 완료하여 클라이언트 프록시를 생성하십시오.

IBM Web Service(SOAP/HTTP) 클라이언트 이주:

이 주제에서는 웹 서비스 API(SOAP/HTTP)를 사용하여 서비스를 호출하는 클라이언트를 이주하는 방법을 보여 줍니다.

이주 중에 기존 클라이언트에 대한 이주가 필요없습니다. 수동으로 생성된 웹 프로젝트를 수정(새 Servlet 맵핑을 작성)해야 하며 때로는 엔터프라이즈 응용프로그램 배치 설명자에 있는 웹 프로젝트의 컨텍스트 루트를 수정하여 서비스를 WebSphere Business Integration Server Foundation에 출력된 정확하게 동일한 주소에 출력해야 할 수 있음을 주의하십시오. "IBM Web Service 바인딩(SOAP/HTTP) 이주" 주제를 참조하십시오.

디자인 변경이 발생했고 새 클라이언트 프록시를 생성하려는 경우 다음 단계가 해당 작업의 수행 방법을 설명합니다. WSIF 또는 RPC 클라이언트 프록시가 생성될 수 있는 5.1에서와는 달리, 6.0에서는 RPC가 WSIF API에서 6.0이 선호하는 API이기 때문에 도구가 RPC 클라이언트 생성만을 지원함을 주의해야 합니다.

참고: WebSphere Integration Developer에서 새 클라이언트 프록시를 생성하려면 WebSphere Process Server 또는 WebSphere Application Server가 설치되어 있어야 합니다.

1. WebSphere Process Server 또는 WebSphere Application Server가 설치되어 있어야 합니다.
2. 웹 서비스 바인딩을 갖는 내보내기에 대응하는 WSDL 파일을 선택한 후 마우스 오른쪽 단추로 클릭하고 웹 서비스 → 클라이언트 생성을 선택하십시오. (이 마법사는 5.1 마법사와 매우 유사합니다.)

3. 클라이언트 프록시 유형에 대해 **Java** 프록시를 선택하고 다음을 클릭하십시오.
4. WSDL의 위치를 채워야 합니다. 다음을 클릭하십시오.
5. 다음으로 적당한 옵션을 선택하여 웹 서비스 런타임 및 서버, J2EE 버전, 클라이언트 유형(Java, EJB, 웹, 응용프로그램 클라이언트)를 포함한 클라이언트 환경 구성을 지정해야 합니다. 다음을 클릭하십시오.
6. 남은 단계를 완료하여 클라이언트 프록시를 생성하십시오.

Apache 웹 서비스(SOAP/HTTP) 클라이언트 이주:

Apache 웹 서비스 클라이언트 API는 WebSphere Integration Developer 서비스 호출에 적합하지 않습니다. IBM Web Service(SOAP/HTTP) 클라이언트 API를 사용하도록 클라이언트 코드를 이주해야 합니다.

자세한 정보는 "IBM Web Service(SOAP/HTTP) 클라이언트" 섹션을 참조하십시오.

5.1에서는 클라이언트 프록시가 자동으로 생성된 경우 해당 프록시는 WSIF API를 사용하여 서비스와 상호작용했습니다. 6.0에서는 RPC가 WSIF API에서 6.0이 선호하는 API이기 때문에 RPC 클라이언트 생성만을 지원합니다.

참고: WebSphere Integration Developer에서 새 클라이언트 프록시를 생성하려면 WebSphere Process Server 또는 WebSphere Application Server가 설치되어 있어야 합니다.

1. WebSphere Process Server 또는 WebSphere Application Server가 설치되어 있어야 합니다.
2. 웹 서비스 바인딩을 갖는 내보내기에 대응하는 WSDL 파일을 선택한 후 마우스 오른쪽 단추로 클릭하고 웹 서비스 → 클라이언트 생성을 선택하십시오. (이 마법사는 5.1 마법사와 매우 유사합니다.)
3. 클라이언트 프록시 유형에 대해 **Java** 프록시를 선택하고 다음을 클릭하십시오.
4. WSDL의 위치를 채워야 합니다. 다음을 클릭하십시오.
5. 다음으로 적당한 옵션을 선택하여 웹 서비스 런타임 및 서버, J2EE 버전, 클라이언트 유형(Java, EJB, 웹, 응용프로그램 클라이언트)를 포함한 클라이언트 환경 구성을 지정해야 합니다. 다음을 클릭하십시오.
6. 남은 단계를 완료하여 클라이언트 프록시를 생성하십시오.

JMS 클라이언트 이주:

JMS API(JMS 메시지를 큐에 전송함)를 통해 5.1 서비스와 통신한 클라이언트는 수동 이주가 필요할 수도 있습니다. 이 주제에서는 JMS API(JMS 메시지를 큐에 전송함)를 사용하여 서비스를 호출하는 클라이언트를 이주하는 방법을 보여 줍니다.

이전 단계에서 작성한 **JMS** 바인딩으로 내보내기가 이 텍스트 또는 오브젝트 메시지를 아무 변경 없이 수신할 수 있어야 합니다. 이를 위해서는 사용자 정의 데이터 바인딩을 작성해야 합니다. 자세한 정보는 "JMS 및 JMS 프로세스 바인딩 이주" 섹션을 참조하십시오.

클라이언트는 반드시 메시지 생성 방식을 변경해야 합니다. 이전 메시지는 WSIFMessage 클래스에 기반했지만 현재 메시지는 commonj.sdo.DataObject 클래스에 기반해야 합니다. 이 수동 이주를 수행하는 방법에 대한 자세한 내용은 "WSIFMessage API 호출을 SDO API로 이주" 섹션을 참조하십시오.

BPC(business process choreographer) 일반 EJB API 클라이언트 이주:

이 주제에서는 5.1 Process Choreographer 일반 EJB API를 사용하여 BPEL 서비스를 호출하는 클라이언트를 이주하는 방법을 보여 줍니다.

자체 메시지 형식으로 DataObject를 사용하는 일반 EJB API의 새 버전이 있습니다. 클라이언트는 반드시 메시지 생성 방식을 변경해야 합니다. 이전 메시지는 WSIFMessage 클래스에 기반했지만 현재 메시지는 commonj.sdo.DataObject 클래스에 기반해야 합니다. ClientObjectWrapper가 특정 메시지 형식에서 메시지 랩퍼를 계속 제공하므로 일반 EJB API는 많이 변경되지 않았다는 점을 참조하십시오.

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

WSIFMessage 오브젝트를 취하는 이전 일반 EJB의 JNDI 이름은 다음과 같습니다.

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

사용자 태스크가 독립 EJB로서 이제 사용 가능하기 때문에 6.0에는 두 개의 일반 EJB가 있습니다. 이들 일반 EJB의 JNDI 이름은 다음과 같습니다.

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

BPC(Business Process Choreographer) 일반 메시징 API 클라이언트 및 JMS 프로세스 바인딩 클라이언트 이주:

WebSphere Process Server 6.0에는 일반 메시징 API가 없습니다. "JMS 및 JMS 프로세스 바인딩 이주" 섹션을 참조하여 이용자에게 비즈니스 프로세스를 공개하고 선택한 바인딩에 따라 클라이언트를 재작성하는 다른 방법을 선택하십시오.

BPC(Business Process Choreographer) 웹 클라이언트 이주:

이 주제에서는 5.1 Process Choreographer 웹 클라이언트 설정 및 사용자 정의 JSP를 이주하는 방법을 보여 줍니다.

이주 마법사에서는 5.1 웹 클라이언트 설정을 유지하며 사용자 태스크 편집기에서는 이 설정을 편집할 수 없습니다. WebSphere Integration Developer 6.0을 사용하여 새 웹 클라이언트 설정 및 JSP를 작성해야 합니다.

웹 클라이언트 수정 이주

5.1에서 사용자는 해당 JSP **Header.jsp** 및 스타일시트 **dwc.css**를 수정하여 Struts 기반 웹 클라이언트의 룩앤필을 수정할 수 있습니다.

6.0 웹 클라이언트(**Business Process Choreographer** 탐색기로 이름 변경됨)가 Struts 대신 JSF(Java Server Face)를 기반으로 하기 때문에 웹 클라이언트 수정의 자동 이주가 불가능합니다. 따라서 이 응용프로그램의 6.0 버전 사용자 정의에 대한 자세한 내용은 "Business Process Choreographer 탐색기" 문서를 참조할 것을 권장합니다.

비즈니스 프로세스 및 스텝 활동에 대해 사용자 정의 JSP를 정의할 수 있습니다. 웹 클라이언트는 이들 JSP를 사용하여 프로세스 및 활동에 대한 입력 및 출력 메시지를 표시합니다.

이들 JSP는 특히 다음 경우에 유용합니다.

1. 메시지에는 메시지의 데이터 구조 가용성을 향상시키기 위해 기본요소가 아닌 파트가 포함되어 있습니다.
2. 웹 클라이언트의 기능을 펼치기 원합니다.

6.0 프로세스에 대한 웹 클라이언트 설정을 지정할 때 사용 가능한 추가 및 다른 옵션이 있으므로, WebSphere Integration Developer를 사용하여 이주된 프로세스 및 활동에 대한 웹 클라이언트 설정을 다시 디자인해야 합니다.

1. 프로세스 캔버스 또는 프로세스의 활동을 선택하십시오.
2. 특성 보기에서 클라이언트 탭을 선택하여 웹 클라이언트 설정을 다시 디자인하십시오.
3. 수동으로 모든 사용자 정의 JSP를 이주하십시오.
 - a. 프로그래밍 모델 변경사항에 대해서는 "SCA 프로그래밍 모델로 이주" 섹션을 참조하십시오.
 - b. 웹 클라이언트는 일반 API를 사용하여 비즈니스 프로세스와 대화합니다. 이들 일반 API로 호출을 이주하는 방법을 표시하는 섹션을 참조하십시오.
4. 프로세스에 대한 6.0 웹 클라이언트 설정에 새 JSP의 이름을 지정하십시오.

주: DataObjects에 어떤 사용자 정의 맵핑도 필요하지 않기 때문에 JSP 맵핑이 6.0 Business Process Choreographer 탐색기에서 필요없습니다.

WebSphere Business Integration Server Foundation BPEL Java 스니펫 이주:

Java 스니펫을 포함하는 모든 BPEL 프로세스에 대해 이 섹션은 이전 Java 스니펫 API에서 응용프로그램을 통해 이동하는 데이터가 Eclipse SDO(Service Data Object)로 저장되는 새 Java 스니펫 API로 이주하는 방법을 설명합니다.

WSIFMessage에서 SDO 상태 전이에 특정하게 수행하는 이주 단계에 대해 "WSIFMessage API 호출에서 SDO API로 이주" 섹션을 참조하십시오.

가능한 경우 스니펫은 이주 마법사를 통해 자동으로 이주되지만 이주 마법사가 전체를 이주할 수 없는 스니펫이 있습니다. 이주를 완료하려면 추가 수동 단계가 필요합니다. 수동으로 이주되어야 하는 Java 스니펫의 유형에 대한 세부사항은 제한사항 주제를 참조하십시오. 이들 스니펫 중 하나가 발생할 때마다 이주 마법사는 자동으로 이주될 수 없는 이유를 설명하고 경고 또는 오류 메시지를 발행합니다.

다음 표는 Process Choreographer 버전 5.1에서 6.0으로 BPEL Java 스니펫 프로그래밍 모델 및 API의 변경사항을 자세히 설명합니다.

표 11. WebSphere Business Integration Server Foundation BPEL Java 스니펫 이주에 대한 변경사항 및 솔루션

변경사항	솔루션
WSIFMessage 기반 래퍼 클래스가 더 이상 WSDL 메시지 유형에 대해 생성되지 않으며, 복합 스키마 유형에 대해 Java Bean 헬퍼 클래스가 생성되지 않습니다.	<p>BPEL 변수는 이름으로 직접 액세스할 수 있습니다. WSDL 메시지 정의가 단일 파트를 갖는 BPEL 변수의 경우 이제 이 변수는 실제 데이터 주위의 래퍼를 갖는 대신 파트를 직접 표시합니다. 메시지 유형이 복수 파트를 갖는 변수는 파트 주위에 DataObject 래퍼를 갖습니다. (WebSphere Application Developer Integration Edition의 래퍼는 WSIFMessage입니다.)</p> <p>BPEL 변수를 6.0 스니펫에서 직접 사용할 수 있기 때문에 5.1 보다 로컬 변수에 대한 필요성이 적습니다.</p> <p>BPEL 변수에 대해 강하게 유형화된 Getter는 내재적으로 메시지 파트 주변에 WSIFMessage 래퍼 오브젝트를 초기화했습니다. WSDL 메시지 정의가 단일 파트만을 갖는 BPEL 변수의 경우 '래퍼' 오브젝트가 없습니다. 이 경우에 BPEL 변수가 직접 파트를 표시합니다(단일 파트가 XSD 단순 유형인 경우에 BPEL 변수는 java.lang.String, java.lang.Integer 등과 같은 Java 오브젝트 래퍼 유형으로 표시됩니다). 다중 파트 WSDL 메시지 정의를 갖는 BPEL 변수는 다르게 처리됩니다. 여전히 파트 주위에 래퍼가 있으며 이 DataObject 래퍼는 이전 오퍼레이션에 의해 이미 설정되지 않은 경우 6.0 Java 스니펫 코드에서 명시적으로 초기화되어야 합니다.</p> <p>5.1 스니펫의 모든 로컬 변수가 BPEL 변수와 동일한 이름이었던 경우 충돌이 있을 수 있으므로 가능하면 이 상황을 개선하십시오.</p>
WSIFMessage 오브젝트는 더 이상 BPEL 변수를 표시하는 데 사용되지 않습니다.	Java 스니펫에서 호출된 임의의 사용자 정의 Java 클래스가 WSIFMessage 매개변수를 갖는 경우 DataObject를 승인/리턴하도록 이주되어야 합니다.
BPEL 변수에 대해 강하게 유형화된 Getter 메소드는 더 이상 사용할 수 없습니다.	변수는 이름으로 직접 액세스할 수 있습니다. WSDL 메시지 정의가 단일 파트를 갖는 BPEL 변수의 경우 이제 실제 데이터 주위의 래퍼를 갖는 대신 파트를 직접 표시함을 참고하십시오. 메시지 유형이 복수 파트를 갖는 변수는 파트 주위에 DataObject 래퍼를 갖습니다. (WebSphere Application Developer Integration Edition의 래퍼는 WSIFMessage입니다.)
BPEL 변수에 대해 강하게 유형화된 Setter 메소드는 더 이상 사용할 수 없습니다.	변수는 이름으로 직접 액세스할 수 있습니다. WSDL 메시지 정의가 단일 파트를 갖는 BPEL 변수의 경우 이제 이 변수는 실제 데이터 주위의 래퍼를 갖는 대신 파트를 직접 표시합니다. 메시지 유형이 복수 파트를 갖는 변수는 파트 주위에 DataObject 래퍼를 갖습니다. (WebSphere Application Developer Integration Edition의 래퍼는 WSIFMessage입니다.)

표 11. WebSphere Business Integration Server Foundation BPEL Java 스니펫 이주에 대한 변경사항 및 솔루션 (계속)

변경사항	솔루션
WSIFMessage를 리턴하는 BPEL 변수에 대해 약하게 유형화된 Getter 메소드는 더 이상 사용할 수 없습니다.	<p>변수는 이름으로 직접 액세스할 수 있습니다. WSDL 메시지 정의가 단일 파트를 갖는 BPEL 변수의 경우 이제 이 변수는 실제 데이터 주위의 랩퍼를 갖는 대신 파트를 직접 표시합니다. 메시지 유형이 복수 파트를 갖는 변수는 파트 주위에 DataObject 랩퍼를 갖습니다. (WebSphere Application Developer Integration Edition의 랩퍼는 WSIFMessage입니다.)</p> <p>getVariableAsWSIFMessage 메소드의 두 가지 변이가 있었음을 주의하십시오.</p> <pre>getVariableAsWSIFMessage(String variableName) getVariableAsWSIFMessage(String variableName, boolean forUpdate)</pre> <p>Java 스니펫 활동의 경우 기본 액세스는 읽기/쓰기입니다. 이 설정은 스니펫의 주석에 있는 변수의 이름 목록을 사용하여 @bpe.readOnlyVariables를 지정하면 읽기 전용으로 바꿀 수 있습니다. 예를 들어, 변수 B 및 D를 다음과 같이 읽기 전용으로 설정할 수 있습니다.</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>또한 조건에 Java 스니펫이 있으면 변수는 기본적으로 읽기 전용이지만 @bpe.readWriteVariables...를 지정하면 이 설정을 읽기/쓰기로 바꿀 수 있습니다.</p>
BPEL 변수에 대해 약하게 유형화된 Setter 메소드는 더 이상 사용할 수 없습니다.	<p>변수는 이름으로 직접 액세스할 수 있습니다. WSDL 메시지 정의가 단일 파트를 갖는 BPEL 변수의 경우 이제 이 변수는 실제 데이터 주위의 랩퍼를 갖는 대신 파트를 직접 표시합니다. 메시지 유형이 복수 파트를 갖는 변수는 파트 주위에 DataObject 랩퍼를 갖습니다. (WebSphere Application Developer Integration Edition의 랩퍼는 WSIFMessage입니다.)</p>
BPEL 변수 메시지 파트에 대해 약하게 유형화된 Getter 메소드는 단일 파트 메시지에 적합하지 않으며 다중 파트 메시지를 위해 변경되었습니다.	<p>BPEL 변수(DataObject의) 특성에 대해 약하게 유형화된 Getter 메소드로 이주하십시오.</p> <p>WSDL 메시지 정의가 단일 파트를 갖는 BPEL 변수의 경우 BPEL 변수가 직접 파트를 표시하며 변수는 Getter 메소드를 사용하지 않고 직접 액세스해야 합니다.</p> <p>getVariablePartAsObject 메소드의 다음 두 가지 변이가 있었습니다.</p> <pre>getVariablePartAsObject(String variableName, String partName) getVariablePartAsObject(String variableName, String partName, boolean forUpdate)</pre> <p>다중 파트 메시지의 경우 6.0에서 다음 메소드에 의해 동등한 기능이 제공됩니다.</p> <pre>getVariableProperty(String variableName, QName propertyName);</pre> <p>6.0에서는 읽기 전용 액세스(5.1에서 forUpdate=forUpdate='false'를 갖는 두 번째 메소드뿐 아니라 위의 첫 번째 메소드의 경우입니다)를 위해 변수를 사용하는 표기법이 없습니다. 변수는 6.0 스니펫에서 직접 사용되며 항상 갱신할 수 있습니다.</p>

표 11. WebSphere Business Integration Server Foundation BPEL Java 스니펫 이주에 대한 변경사항 및 솔루션 (계속)

변경사항	솔루션
BPEL 변수의 메시지 파트에 대해 약하게 유형화된 Setter 메소드는 단일 파트 메시지에 적합하지 않으며 다중 파트 메시지를 위해 변경되었습니다.	<p>BPEL 변수(DataObject)의 특성에 대해 약하게 유형화된 Setter 메소드로 이주하십시오.</p> <p>WSDL 메시지 정의가 단일 파트를 갖는 BPEL 변수의 경우 BPEL 변수가 직접 파트를 표시하며 변수는 Setter 메소드를 사용하지 않고 직접 액세스해야 합니다.</p> <p>다음 메소드에 대한 호출이 이주되어야 합니다.</p> <pre>setVariableObjectPart(String variableName, String partName, Object data)</pre> <p>다중 파트 메시지의 경우 6.0에서 다음 메소드에 의해 동등한 기능이 제공됩니다.</p> <pre>setVariableProperty(String variableName, QName propertyName, Serializable value);</pre>
BPEL 파트너 링크에 대해 강하게 유형화된 Getter 메소드는 더 이상 사용할 수 없습니다.	BPEL 파트너 링크에 대해 약하게 유형화된 Getter 메소드로 이주하십시오.
BPEL 파트너 링크에 대해 강하게 유형화된 Setter 메소드는 더 이상 사용할 수 없습니다.	BPEL 파트너 링크에 대해 약하게 유형화된 Setter 메소드로 이주하십시오.
BPEL 상관 세트에 대해 강하게 유형화된 Getter 메소드는 더 이상 사용할 수 없습니다.	<p>V5.1 스니펫:</p> <pre>String corrSetPropStr = getCorrelationSetCorrSetAPropertyCustomerName(); int corrSetPropInt = getCorrelationSetCorrSetBPropertyCustomerId();</pre> <p>V6.0 스니펫:</p> <pre>String corrSetPropStr = (String) getCorrelationSetProperty("CorrSetA", new QName("CustomerName")); int corrSetPropInt = ((Integer) getCorrelationSetProperty ("CorrSetB", new QName("CustomerId"))).intValue();</pre>
BPEL 활동 사용자 정의 정의 특성에 대해 약하게 유형화된 Getter 메소드에 필요한 추가 매개변수.	<p>V5.1 스니펫:</p> <pre>String val = getActivityCustomProperty("propName");</pre> <p>V6.0 스니펫:</p> <pre>String val = getActivityCustomProperty ("name-of-current-activity", "propName");</pre>

변경사항	솔루션
BPEL 활동 사용자 정의 특성에 대해 약하게 유형화된 Setter 메소드에 필요한 추가 매개변수.	V5.1 스니펫: <pre>String newVal = "new value"; setActivityCustomProperty("propName", newVal);</pre> V6.0 스니펫: <pre>String newVal = "new value"; setActivityCustomProperty("name-of-current-activity", "propName", newVal);</pre>
raiseFault(QName faultQName, Serializable message) 메소드는 더 이상 존재하지 않습니다.	가능하면 raiseFault(QName faultQName, String variableName)로 이주하십시오. 그렇지 않으면 raiseFault(QName faultQName) 메소드로 이주하거나 Serializable 오브젝트에 대한 새 BPEL 변수를 작성하십시오.

WebSphere Business Integration 어댑터와의 상호작용 이주:

JMS 클라이언트가 WebSphere Business Integration 어댑터인 경우, 엔터프라이즈 서비스 발견 도구를 사용하여 JMS 바인딩을 사용한 가져오기를 작성해야 할 수도 있습니다. 해당 가져오기는 WebSphere Business Integration 어댑터가 예상하는 정확한 형식으로 SDO를 직렬화하기 위해 특수 데이터 바인딩을 사용합니다.

엔터프라이즈 서비스 발견 도구에 액세스하려면 다음을 수행하십시오.

1. 파일 → 새로 작성 → 기타 → 비즈니스 통합으로 이동하여 엔터프라이즈 서비스 발견을 선택하십시오. 다음을 클릭하십시오.
2. **WebSphere Business Integration** 어댑터 아티팩트 임пор터를 선택하십시오. 다음을 클릭하십시오.
3. WebSphere Business Integration 어댑터의 구성(.cfg) 파일 및 어댑터가 사용하는 비즈니스 오브젝트의 XML 스키마가 들어 있는 디렉토리의 경로를 입력하십시오. 다음을 클릭하십시오.
4. 생성되는 조회를 조사한 후, 올바른 경우 **조회 실행**을 클릭하십시오. **조회로 발견된 오브젝트** 목록에서 추가하려는 오브젝트를(하나씩) 선택하고 >> 추가 단추를 클릭하십시오.
5. 비즈니스 오브젝트에 대한 구성 매개변수를 허용하고 확인을 클릭하십시오.
6. 각 비즈니스 오브젝트에 대해 반복하십시오.
7. 다음을 클릭하십시오.
8. 런타임 비즈니스 오브젝트 형식에 대해 **SDO**를 선택하십시오. 대상 프로젝트에 대해 방금 이주한 모듈을 선택하십시오. 폴더 필드는 공백으로 두십시오.
9. 완료를 클릭하십시오.

이 도구는 특수 데이터 바인딩이 예상하는 형식으로 이전 XSD를 이주하므로 모듈에서 이전 WebSphere Business Integration 어댑터 XSD를 제거하고 새 XSD를 사용하십시오. 모듈이 어댑터에서 메시지를 수신하지 않는 경우 이 도구에서 생성된 내보내기를 삭제하십시오. 모듈이 어댑터로 메시지를 송신하지 않는 경우 가져오기를 삭제하십시오. 이 기능에 대한 자세한 정보는 Information Center를 참조하십시오.

SOAP 인코드 배열 유형을 갖는 WSDL 인터페이스 이주:

이 섹션은 SOAP 인코드 배열 유형을 갖는 XML 스키마를 이주 또는 처리하는 방법을 설명합니다.

RPC 스타일을 갖는 Soap 인코드 배열 유형은 6.0에서 구체적 유형의 바운드되지 않은 시퀀스로서 취급됩니다. 프로그래밍 모델이 RPC 스타일 대신 문서/리터럴 랩 스타일쪽으로 이동하고 있기 때문에 어떤 방법으로도 soapend:Array 유형을 참조하는 XSD 유형을 작성하는 것은 바람직하지 않습니다.

SCA 응용프로그램이 soapend:Array 유형을 사용하는 외부 서비스를 호출해야 하는 경우가 있습니다. 일부 경우에 이를 피할 수 있는 방법이 없으며 다음은 이 상황을 다루는 방법을 보여 줍니다.

샘플 WSDL 코드:

```
<xsd:complexType name="Vendor">
<xsd:all>
<xsd:element name="name" type="xsd:string" />
<xsd:element name="phoneNumber" type="xsd:string" />
</xsd:all>
</xsd:complexType>
</xsd:schema>
<xsd:complexType name="Vendors">
<xsd:complexContent mixed="false">
<xsd:restriction base="soapenc:Array">
<xsd:attribute wsdl:arrayType="tns:Vendor[]" ref="soapenc:arrayType"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" />
</xsd:restriction>
</xsd:complexContent>
<xsd:complexType name="VendorsForProduct">
<xsd:all>
<xsd:element name="productId" type="xsd:string" />
<xsd:element name="vendorList" type="tns:Vendors" />
</xsd:all>
</xsd:complexType>
<xsd:complexType name="Product">
<xsd:all>
<xsd:element name="productId" type="xsd:string" />
<xsd:element name="productName" type="xsd:string" />
</xsd:all>
</xsd:complexType>
<message name="doFindVendorResponse">
<part name="returnVal" type="tns:VendorsForProduct" />
</message>
<operation name="doFindVendor">
<input message="tns:doFindVendor" />
<output message="tns:doFindVendorResponse" />
</operation>
```

이 웹 서비스의 클라이언트에 대한 샘플 코드:

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

```

DataObject resultElements=findVendorResult.getDataObject("vendorList");
Sequence results=resultElements.getSequence(0);
for (int i=0, n=results.size(); i
for (int i=0, n=results.size(); i

```

다음은 데이터 오브젝트의 루트 유형이 soapenc:Array인 다른 예제입니다. sampleElements DataObject가 위에 나열된 두 번째 스키마를 사용하여 작성되는 방식을 주목하십시오. 먼저 DataObject의 유형을 얻은 다음 sampleStructElement의 특성을 얻습니다. 이것이 실제로는 플레이스홀더 특성이며 DataObjects를 시퀀스에 추가할 때 사용할 유효한 특성을 얻기 위해서만 사용됩니다. 이와 비슷한 패턴을 사용자 시나리오에서 사용할 수 있습니다.

샘플 WSDL 코드:

```

<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>

```

이 웹 서비스의 클라이언트에 대한 샘플 코드:

```

// Create the input DataObject and get the SDO sequence for the any
// element
DataFactory dataFactory=DataFactory.INSTANCE;
DataObject arrayOfStruct = dataFactory.create("http://soapinterop.org/xsd","ArrayOfSOAPStruct");
Sequence sequence=arrayOfStruct.getSequence("any");
// Get the SDO property for the sample element that we want to use
// here to populate the sequence

```

```
// We have defined this element in an XSD file, see SampleElements.xsd
DataObject sampleElements=dataFactory.create("http://sample/elements",
"DocumentRoot");
Property property = sampleElements.getType().getProperty("sampleStructElement");
// Add the elements to the sequence
DataObject item=dataFactory.create("http://soapinterop.org/xsd", "SOAPStruct");
item.setInt("varInt", 1);
item.setString("varString", "Hello");
item.setFloat("varFloat", 1.0f);
sequence.add(property, item);
item=dataFactory.create("http://soapinterop.org/xsd", "SOAPStruct");
item.setInt("varInt", 2);
item.setString("varString", "World");
item.setFloat("varFloat", 2.0f);
sequence.add(property, item);
// Invoke the echoStructArray operation
System.out.println("[client] invoking echoStructArray operation");
DataObject echoArrayOfStruct = (DataObject)interopTest.invoke("echoStructArray", arrayOfStruct);
// Display the results
if (echoArrayOfStruct!=null) {
sequence=echoArrayOfStruct.getSequence("any");
for (int i=0, n=sequence.size(); i<n; i++) {
item=(DataObject)sequence.getValue(i);
System.out.println("[client] item varInt = "+
item.getInt("varInt")+
varString="+item.getString("varString")+
varFloat="+item.getFloat("varFloat"));
}
```

WebSphere Business Integration EJB 프로젝트 이주:

WebSphere Studio Application Developer Integration Edition에서 EJB 프로젝트는 CMM(Extended Messaging) 및 CMP/A(Component-Managed Persistence Anywhere)와 같은 특수 WebSphere Business Integration 기능을 가질 수 있습니다. 그런 프로젝트에 대한 배치 설명자는 이주되어야 하며 이 섹션은 해당 이주를 수행하는 방법을 설명합니다.

이 이주를 수행하려면 다음 단계를 완료하십시오.

1. WebSphere Business Integration EJB 프로젝트를 새 6.0 작업공간으로 복사하고 파일 → 가져오기 → 작업공간으로 기존 오브젝트 마법사를 사용하여 WebSphere Integration Developer에서 가져오십시오. 선택적으로 J2EE 이주 마법사를 실행할 수도 있습니다.
2. 6.0 작업공간에서 실행 중인 WebSphere Integration Developer의 모든 인스턴스를 닫으십시오.
3. EJB 프로젝트의 WebSphere Business Integration 배치 설명자를 이주할 다음 스크립트를 실행하십시오.

Windows:

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

Linux:

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

다음 매개변수가 지원되며, 작업공간 및 프로젝트 이름은 필수입니다.

사용법: WSADIEEJBProjectMigration.bat
 [-e eclipse-folder] -d workspace -p project
 eclipse-folder: Eclipse의 폴더의 위치. 대개 제품 설치 폴더
 아래에 있는 'eclipse'입니다.
 workspace: 이주될 WSADIE EJB 프로젝트를 포함하는 작업공간.
 project: 이주할 프로젝트의 이름.

예:

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

4. WebSphere Integration Developer를 열 때 갱신된 파일을 가져오기 위해 EJB 프로젝트를 새로 고쳐야 합니다.
5. EJB 프로젝트에서 **ibm-web-ext.xmi** 파일을 검색하십시오. 하나가 있는 경우 파일에서 다음 행이 요소 아래에 존재하는지 확인하십시오.

```
<webappext:WebAppExtension> element:  

<webApp href="WEB-INF/web.xml#WebApp"/>
```
6. 5.1에서 생성된 이전 배치 코드를 제거하십시오. WebSphere Application Server 가이드라인에 따라서 배치 코드를 다시 생성하십시오.

이름 공간 충돌에 대한 수동 수정 실행:

WebSphere Studio Application Developer Integration Edition 5.1에서 동일한 이름 및 대상 이름 공간을 갖는 두 개의 서로 다른 XSD 또는 WSDL 유형을 정의할 수 있습니다. 이 기능은 WebSphere Integration Developer 6.0에서는 지원되지 않습니다. 이주한 프로젝트를 빌드한 후에 중복 정의 오류가 발생하면 수동 이주를 수행해야 합니다.

이 문제를 해결하려면 다음 단계를 완료하십시오.

1. 두 정의가 동일하면 둘 중 하나를 삭제한 다음 프로젝트를 정리하고 다시 빌드하십시오. 기존 WSDL/XSD 파일을 삭제하지 않은 정의가 포함된 파일로 지시하여, 발생 가능한 오류를 수정하십시오.
2. 두 정의가 동일하지 않고 이주된 서비스에서 두 정의를 모두 사용해야 할 경우 정의 이름 또는 대상 이름 공간의 이름을 바꾸십시오. 전체 파일에서 중복인 정의가 많지 않을 경우 해당 이름을 바꾸는 것이 좋습니다. 파일에서 모든 정의가 중복인 경우 모든 정의의 대상 이름 공간을 바꾸는 것이 좋습니다. 프로젝트를 정리하고 다시 빌드하여 수정한 정의에서 사용할 아티팩트가 새로운 정의 이름 또는 이름 공간을 참조하는지 확인하십시오.
3. WSDL 파일에서 동일한 이름 공간에 대해 두 개의 import 문이 있는 경우 이 문제는 WSDL 중 하나가 다른 WSDL을 가져오고 이 WSDL은 다음 WSDL을 가져오는 식으로 import를 변경하여 이 WSDL 파일 당 이름 공간에 하나의 import만 있도록 하면 해결할 수 있습니다. 그런 다음 프로젝트를 정리하고 다시 빌드하십시오.

5.1 WSIF(Web Services Invocation Framework) 정의 수동 삭제:

소스 아티팩트 이주를 완료한 후에는 더 이상 사용하지 않는 6.0 프로젝트에서 모든 5.1 WSIF 바인딩 및 서비스 WSDL 정의를 삭제해야 합니다. WSIF 바인딩 또는 서비스가 계속 사용되는 유일한 경우는 서비스 이주에 대한 이용 상황입니다.

다음 WSDL 이름 공간은 바인딩 또는 서비스 정의가 5.1 WSIF 서비스이며 더 이상 사용되지 않으면 버려질 수도 있다는 것을 나타냅니다.

EJB WSIF 이름 공간:

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

Java WSIF 이름 공간:

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

JMS WSIF 이름 공간:

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

Business Process WSIF 이름 공간:

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

Transformer WSIF 이름 공간:

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

IMS WSIF 이름 공간:

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

CICS-ECI WSIF 이름 공간:

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

CICS-EPI WSIF 이름 공간:

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

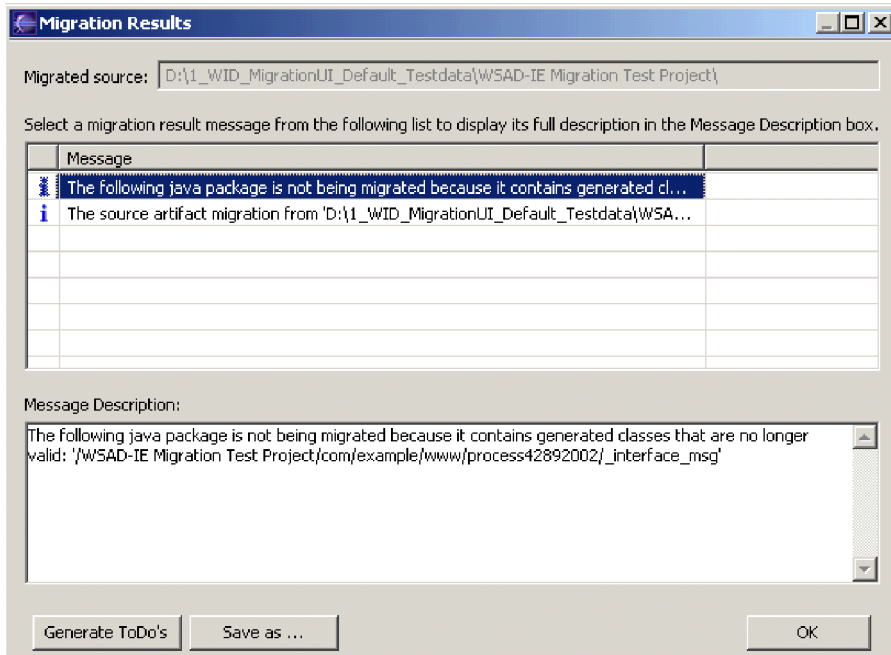
HOD WSIF 이름 공간:

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

소스 아티팩트 이주 확인

이주 마법사가 완료되면 오류, 경고 및/또는 정보 메시지로 구성된 목록이 표시됩니다. 이 메시지를 사용하여 소스 아티팩트 이주를 확인할 수 있습니다.

이주 마법사가 완료되었을 때 다음 페이지가 표시됩니다.



각 메시지를 조사하여 완전히 이주할 수 없었던 아티팩트를 즉시 수정하기 위해 어떤 조치를 취해야 하는지를 확인하십시오.

이주 부분을 완료했는지 확인하려면 비즈니스 통합 Perspective로 전환하여 이전 서비스 프로젝트의 모든 프로세스 및 WSDL 인터페이스가 새 모듈로 표시되는지 확인하십시오. 프로젝트를 빌드하고 프로젝트 빌드를 방해하는 오류를 수정하십시오.

비즈니스 통합 응용프로그램의 이주를 완료하기 위해 필요한 수동 이주 단계를 완료한 후, 응용프로그램을 EAR 파일로서 내보내고 적당한 자원을 구성하여 WebSphere Process Server에 설치하십시오.

WebSphere Integration Developer를 사용하여 새 클라이언트 코드를 생성하거나 클라이언트 코드를 이주할 때 필요한 수동 이주 단계를 수행하십시오. 클라이언트가 응용프로그램에 액세스할 수 있는지와 응용프로그램이 이전 런타임 환경에서 수행한 동작을 금지하지 않는지 확인하십시오.

소스 아티팩트 이주 실패에 대한 작업

WebSphere Studio Application Developer Integration Edition에서의 소스 아티팩트 이주 실패 시 이를 처리하는 방법입니다.

다음은 몇 가지 가능한 소스 아티팩트 이주 실패입니다.

- 다음 메시지를 수신할 수 있습니다.

"이주 오류 메시지"

이유: 심각한 이주 실패

메시지: IBM 담당자에게 문의

이 경우 새 작업공간의 .metadata 폴더에 있는 WebSphere Integration Developer 로그 파일을 확인하여 오류 세부사항을 확인해야 합니다. 가능한 경우 오류의 원인을 해결하고 새 작업공간에서 작성된 모듈을 삭제 후 다시 이주하십시오.

이주 마법사가 이 메시지를 표시하지 않고 완료되었다면 정보, 경고 및 오류 메시지 목록이 표시됩니다. 이들은 서비스 프로젝트의 일부 부분이 자동으로 이주될 수 없으며 이주를 완료하기 위해서는 수동 변경을 수행해야 함을 나타냅니다.

소스 아티팩트 이주 프로세스의 우수 사례

WebSphere Studio Application Developer Integration Edition 소스 아티팩트 이주 프로세스에는 많은 우수 사례가 있습니다.

다음 예제에서는 새 프로그래밍 모델로 이주하는지 확인하도록 WebSphere Studio Application Developer Integration Edition 서비스를 디자인하는 방법을 표시합니다.

- 가능할 때마다 지정 활동을 사용하십시오(고급 변환이 필요할 때에만 사용되는 변환기 서비스와는 반대임). 중간 컴포넌트는 SCA 모듈이 변환기 서비스를 호출하기 위해 생성되어야 하기 때문에 이 방법을 사용해야 합니다. 또한 WebSphere Integration Developer에 5.1에서 작성된 변환기 서비스에 대한 특수 도구 지원이 없습니다. (변환기 서비스의 작동으로 변경해야 하는 경우 WSDL 또는 XML 편집기를 사용하여 WSDL 파일에 임베드된 XSLT를 수정해야 합니다.)
- WS-I(Web Services Interoperability) 스펙 및 6.0에서 선호하는 스타일별로 WSDL 메시지당 하나의 파트를 지정하십시오.
- WSDL doc-literal 스타일이 6.0에서 선호하는 스타일이므로 이를 사용하십시오.
- 모든 복합 유형에 이름이 지정되었는지 확인하여 각 복합 유형이 대상 이름 공간 및 이름으로 고유하게 식별되도록 하십시오. 다음은 복합 유형 및 해당 유형의 요소를 정의할 때 권장하는 방법을 보여줍니다. (요소를 사용하는 요소 정의 다음에 복합 유형 정의 표시)

```
<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>
```

다음 예제는 익명의 복합 유형입니다(익명 복합 유형 정의가 포함된 요소). 익명의 복합 유형은 SDO가 XML로 직렬화되는 경우에는 문제를 일으킬 수도 있기 때문에 사용을 피해야 합니다.

```
<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>

```

- 외부 이용자를 위한 서비스를 출력하는 경우 IBM Web Service를 사용하여(Apache SOAP/HTTP와 반대 로서) 서비스 배치 코드를 생성하십시오. IBM Web Service는 6.0에서 직접 지원되고 Apache 웹 서비스 는 지원되지 않기 때문입니다.
- 이주 중에 사용자가 수행해야 하는 재구성의 양을 최소화하기 위해 5.1에서 WSDL 및 XSD 파일을 구성 하는 두 가지 방법이 있습니다. 6.0에서, WSDL 및 XSD 파일과 같은 공유 아티팩트는 BI 서비스가 참조 하기 위해 BI 프로젝트(비즈니스 통합 모듈 및 라이브러리)에 위치해야 합니다.
 - 둘 이상의 서비스 프로젝트가 공유하는 모든 WSDL 파일들은 서비스 프로젝트가 참조할 수 있는 Java 프로젝트에 보존하십시오. 6.0으로의 이주 중에 5.1 공유 Java 프로젝트와 동일한 이름을 갖는 새로운 비즈니스 통합 라이브러리를 작성합니다. 5.1 공유 Java 프로젝트의 모든 아티팩트를 라이브러리로 복사 하여 해당 아티팩트를 사용하는 서비스 프로젝트를 이주할 때 이주 마법사에서 아티팩트를 해석할 수 있 도록 하십시오.
 - 서비스 프로젝트가 참조하는 모든 WSDL/XSD 파일의 로컬 사본은 서비스 프로젝트 자체에 보존하십시 오. WebSphere Studio Application Developer Integration Edition 서비스 프로젝트는 WebSphere Integration Developer에서 비즈니스 통합 모듈로 이주하며 모듈은 다른 모듈에 대한 종속성이 없습니 다. (WSDL 또는 XSD 파일을 공유하기 위해 다른 서비스 프로젝트에 대한 종속성이 있는 서비스 프로 젝트는 분명하게 이주하지 않을 수 있습니다.)
- Business Process Choreographer Generic Messaging API(Generic MDB)는 6.0에서 지원되지 않으므로 사용하지 마십시오. MDB 인터페이스 오퍼링 동적 바인딩은 6.0에서 사용할 수 없습니다.
- 특별한 버전의 프로세스에만 한정된 생성된 세션 Bean을 호출하는 대신 BPC(Business Process Choreographer) 일반 EJB API를 사용하십시오. 이러한 세션 Bean은 V6.0.0.0에서 생성되지 않습니다.
- 동일한 오퍼레이션에 대해 복수 응답을 갖는 비즈니스 프로세스가 있는 경우, 프로세스 중에 해당 오퍼레이 셴에 대한 모든 응답에서 응답마다 한 세트의 클라이언트 설정만이 지원되는 6.0의 클라이언트 설정과 동일 한 설정의 클라이언트가 있는 지 확인하십시오.
- 다음 가이드라인에 따라 BPEL Java 스니펫을 디자인하십시오.
 - 사용자 정의 Java 클래스에 WSIFMessage 매개변수를 전송하지 마십시오 - 가능한 WSIFMessage 데 이터 형식에 따라 달라지지 않도록 하십시오.
 - 가능하면 WSIF 메타데이터 API 사용을 피하십시오.

- BPEL Java 스니펫에서 BPEL 변수와 동일한 이름을 갖는 로컬 변수의 선언을 피하십시오. 6.0에서 BPEL 변수는 스니펫에서 직접 액세스할 수 있으므로 동일한 이름을 갖는 로컬 변수는 충돌을 유발할 수 있습니다.
- WSDL PortTypes/Messages에서 생성된 Java/EJB 스켈레톤은 WSIF 클래스에 종속되기 때문에(예: WSIFFormatPartImpl) 가능하면 하향식 EJB 또는 Java 서비스 작성을 피하십시오. 대신 먼저 Java/EJB 인터페이스를 작성하고 Java 클래스/EJB 주위에 서비스를 생성하십시오(상향식 접근).
- soapenc:Array 유형을 참조하는 WSDL 인터페이스의 작성 또는 사용을 피하십시오. 이 유형의 인터페이스는 기본적으로 SCA 프로그래밍 모델에서 지원되지 않습니다.
- 상위 레벨 요소가 배열 유형(maxOccurs 속성이 1보다 큼)인 메시지 유형의 작성을 피하십시오. 이 유형의 인터페이스는 기본적으로 SCA 프로그래밍 모델에서 지원되지 않습니다.
- WSDL 인터페이스를 정확하게 정의하십시오. 가능하면 xsd:anyType 유형을 참조하는 XSD complexTypes를 피하십시오.
- EJB 또는 Java Bean에서 생성하는 임의의 WSDL 및 XSD의 경우 대상 이름 공간을 고유하게 지정(Java 클래스 이름 및 패키지 이름은 대상 이름 공간으로 나타남)하여 WebSphere Process Server V6으로 이주할 때 충돌을 피하십시오. WebSphere Process Server V6에서 동일한 이름 및 대상 이름 공간을 갖는 두 개의 서로 다른 WSDL/XSD 정의는 허용되지 않습니다. 이러한 상황은 웹 서비스 마법사 또는 Java2WSDL 명령을 대상 이름 공간에 대한 명시적인 지정 없이 사용할 경우에 흔히 발생합니다. (대상 이름 공간은 EJB 또는 Java Bean의 패키지 이름에 대해 고유하지만 클래스 자체에 대해서는 고유하지 않기 때문에 웹 서비스가 동일한 패키지 안에서 두 개 이상의 EJB 또는 Java Bean에 대해 생성될 때 문제가 발생합니다.) 이 문제는 웹 서비스 마법사에서 사용자 정의 패키지를 이름 공간 맵핑에 지정하거나 **-namespace** Java2WSDL 명령행 옵션을 사용하여, 생성된 파일의 이름 공간이 주어진 클래스에 대해 고유하도록 만들면 해결됩니다.
- 가능하면 모든 WSDL 파일에 고유한 이름 공간을 사용하십시오. WSDL 1.1 스펙에 따르면 동일한 이름 공간을 갖는 두 개의 서로 다른 WSDL 파일을 가져오는 것에는 제한사항이 있으며 WebSphere Integration Developer 6.0에서 이러한 제한사항은 엄격하게 적용됩니다.

이주 프로세스 제한사항(소스 아티팩트 이주 시)

WebSphere Studio Application Developer Integration Edition 소스 아티팩트 이주 프로세스에는 일정한 제한사항이 있습니다.

다음 목록에서는 소스 아티팩트 이주 시 이주 프로세스의 몇 가지 제한사항을 자세히 설명합니다.

일반적인 제한사항

- 이주 마법사는 전체 WebSphere Studio Application Developer Integration Edition 작업공간을 처리할 수 없습니다. 한 번에 하나의 WebSphere Studio Application Developer Integration Edition 서비스 프로젝트를 이주함을 의미합니다.
- 이주 마법사는 응용프로그램 2진을 이주하지 않습니다. 오직 WebSphere Studio Application Developer Integration Edition 서비스 프로젝트에 있는 소스 아티팩트만 이주합니다.
- 비즈니스 규칙 Bean은 WebSphere Process Server 6.0에서 사용되지 않지만, WebSphere Process Server 설치 중에 WebSphere Process Server 6.0 서버에서 "있는 그대로" 실행할 수 있게 사용하지 않는 비즈니스

스 규칙 Bean에 대한 지원을 설치하는 옵션이 있습니다. 그러나 이전 비즈니스 규칙 Bean을 위한 도구 지원은 없으며, 이전 비즈니스 규칙 Bean 아티팩트를 도구에서 컴파일하려는 경우 WebSphere Integration Developer 문서에 따라 임베디드 WebSphere Process Server 6.0 테스트 서버 위에 지원되지 않는 해당 기능을 설치한 후 수동으로 프로젝트 클래스 경로에 지원되지 않는 jar 파일을 외부 jar로서 추가해야 합니다. 6.0 스펙에 따라서 비즈니스 규칙을 다시 작성하려면 WebSphere Integration Developer에서 사용 가능한 새 비즈니스 규칙 도구를 사용해야 합니다.

- 확장 메시징 지원은 WebSphere Process Server 6.0에서 지원되지 않지만 WebSphere Process Server 설치 중에 WebSphere Process Server 6.0 서버에서 기존 응용프로그램이 "있는 그대로" 실행되도록 지원되지 않는 확장 메시징 기능에 대한 지원을 설치하는 옵션이 있습니다. 그러나 이전 확장 메시징 기능을 위한 도구 지원은 없으며, 이전 확장 메시징 아티팩트가 도구에서 컴파일되기 원하는 경우 WebSphere Integration Developer 문서에 따라 임베디드 WebSphere Process Server 6.0 테스트 서버 위에 지원되지 않는 해당 기능을 설치한 후 수동으로 프로젝트 클래스 경로에 지원되지 않는 jar 파일을 "외부 jars"로서 추가해야 합니다.
- 표준 제공 JMS 데이터 바인딩은 사용자 정의 JMS 헤더 특성에 대한 액세스를 제공하지 않습니다. 모든 사용자 정의 JMS 헤더 특성에 액세스하기 위해 사용자 정의 데이터 바인딩을 SCA 서비스에 작성해야 합니다.

SCA 프로그래밍 모델 제한사항

- SDO 버전 1 스펙은 COBOL 또는 C 바이트 배열에 대한 액세스를 제공하지 않습니다. 이것은 IMS 다중 세그먼트에 대해 작업하는 사용자에게 영향을 줍니다.
- 직렬화에 대한 SDO 버전 1 스펙은 COBOL 재정의 또는 C 유니온을 지원하지 않습니다.
- SCA 프로그래밍 모델에 따라서 소스 아티팩트를 다시 디자인할 때 문서/리터럴 랩 WSDL 스타일(WebSphere Integration Developer 도구를 사용하여 작성되는 새 아티팩트에 대한 기본 스타일임)이 메소드 과부하를 지원하지 않음을 주의하십시오. 다른 WSDL 스타일은 여전히 지원되므로 이런 경우에는 문서/리터럴 랩 이외의 다른 WSDL 스타일/인코딩을 사용할 것을 권장합니다.
- 배열에 대한 기본 지원이 제한됩니다. soapenc:Array 유형을 갖는 WSDL 인터페이스를 공개하는 외부 서비스를 호출하기 위해서는 "maxOccurs" 속성이 1보다 큰 요소를 정의하는 WSDL 인터페이스를 작성해야 합니다. (배열 유형 정의를 위한 권장 접근 방식입니다.)

BPEL 이주 프로세스 기술 제한사항

- **BPEL 오퍼레이션별 다중 응답** - WebSphere Business Integration Server Foundation에서 비즈니스 프로세스는 동일한 오퍼레이션에 대해 하나의 수신 활동 및 다중 응답 활동을 가질 수 있습니다.
- **BPEL Java 스니펫 이주 시 제한사항** - WebSphere Studio Application Developer Integration Edition에서 WebSphere Integration Developer로 프로그래밍 모델이 상당 부분 변경되었으며 지원되는 모든 WebSphere Studio Application Developer Integration Edition API를 해당 WebSphere Integration Developer API로 직접 이주할 수 없습니다. 모든 Java 로직은 BPEL Java 스니펫에서 찾을 수 있으므로 자동 이주 도구가 모든 Java 스니펫을 새 프로그래밍 모델로 변환시키지 못할 수도 있습니다. 대부분의 표준 스니펫 API 호출은 자동으로 5.1 Java 스니펫 프로그래밍 모델에서 6.0 Java 스니펫 프로그래밍 모델로 이주됩니다. WSIF API 호출은 가능한 경우 DataObject API 호출로 이주됩니다. WSIFMessage 오브젝트

를 허용하는 모든 사용자 정의 Java 클래스는 수동으로 이주해야 하므로 해당 클래스에서 대신 `commonj.sdo.DataObject` 오브젝트를 허용하여 리턴합니다.

- **WSIFMessage 메타데이터 API** - 가능한 경우 모든 WSIF API는 사용하지 않아야 합니다. 이주 마법사는 WSIFMessage 메타데이터 및 기타 WSIF API를 동등한 SDO로 자동 이주할 수 없으므로 수동으로 이주해야 합니다.
- **EndpointReference/EndpointReferenceType API** - 이 클래스는 자동으로 이주되지 않습니다. 파트너 링크 getter/setter 메소드가 5.1의 `com.ibm.websphere.srm.bpel.wsaddressing.EndpointReferenceType` 오브젝트 대신 `commonj.sdo.DataObject` 오브젝트를 다루기 때문에 수동 이주가 필요합니다.
- **중복된 이름의 복합 유형** - WSDL 또는 XSD에서 응용프로그램이 이름 공간 및 로컬 이름이 동일하거나 이름 공간은 다르지만 로컬 이름은 동일한 복합 유형을 선언하는 경우 이 유형을 사용하는 Java 스니펫을 올바르게 이주하지 못할 수도 있습니다. 이주 마법사를 완료한 후 스니펫이 올바른지 확인하십시오.
- **java.lang 패키지의 Java 클래스와 로컬 이름이 동일한 복합 유형** - WSDL 또는 XSD에서 응용프로그램이 J2SE 1.4.2의 `java.lang` 패키지에 있는 클래스와 동일한 로컬 이름의 복합 유형을 선언하는 경우 해당 `java.lang` 클래스를 사용하는 Java 스니펫을 올바르게 이주하지 못할 수도 있습니다. 이주 마법사를 완료한 후 스니펫이 올바른지 확인하십시오.
- **BPEL Java 스니펫의 로컬 변수와 동일한 이름을 갖는 BPEL 변수** - BPEL Java 스니펫에 BPEL 변수와 동일한 이름을 갖는 로컬 변수를 정의한 경우, Java 스니펫에서 이제 BPEL 변수를 이름으로 액세스할 수 있으며 충돌이 존재할 수 있기 때문에 수동으로 이를 수정해야 합니다.
- **읽기 전용 및 읽기/쓰기 BPEL 변수** - 5.1 Java 스니펫 코드에서 BPEL 변수를 "읽기 전용"으로 설정할 수 있으며 이것은 이 오브젝트를 수정해도 BPEL 변수 값에 전혀 영향을 주지 않는다는 것을 나타냅니다. 또한 BPEL 변수를 "읽기/쓰기"로 설정할 수도 있으며 이것은 오브젝트를 수정하면 BPEL 변수 자체에 대해 반영된다는 것을 나타냅니다. 다음은 Java 스니펫을 5.1 BPEL Java 스니펫에서 "읽기 전용"으로 액세스할 수 있는 네 가지 방법을 보여줍니다.

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

다음은 BPEL 변수를 5.1 BPEL Java 스니펫에서 "읽기/쓰기"로 액세스할 수 있는 두 가지 방법입니다.

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

6.0에서 BPEL 변수에 대한 읽기 전용 및 읽기/쓰기 액세스는 "스니펫별 기반"으로 처리되며 BPEL Java 스니펫에 BPEL 변수에 대한 업데이트를 스니펫이 실행을 완료한 후에 버리나 유지하는지 여부를 지정하는, 특별 주석을 추가할 수 있다는 것을 의미합니다. 다음은 6.0 BPEL Java 스니펫 유형의 기본 액세스 설정입니다.

```
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"

이주 시 이러한 주석은 변수가 6.0의 기본값이 아닌 방식으로 액세스되는 경우에 자동으로 작성됩니다. 충돌이 발생하면(즉 BPEL 변수가 동일한 스니펫에서 "읽기 전용" 및 "읽기/쓰기"로 액세스되는 경우) 경고가 발행되며 액세스는 "읽기/쓰기"로 설정됩니다. 해당 경고가 수신되면 현재 상황에서 BPEL 변수 액세스를 "읽기/쓰기"로 설정하는 것이 올바른지 확인하십시오. 이 설정이 올바르지 않으면 WebSphere Integration Developer BPEL 편집기에서 직접 정정해야 합니다.

- 복합 유형에서 **Many-valued** 기본요소 특성 - 5.1에서, 다중 값 특성은 특성 유형의 배열로 표시됩니다. 마찬가지로 특성을 가져와 설정하는 호출은 배열을 사용합니다. 6.0에서는, java.util.List가 이 표시에 사용됩니다. 자동 이주 시 값이 여러 개인 특성이 Java 오브젝트의 일부 유형인 모든 경우를 처리하지만 특성 유형이 Java 기본요소(int, long, short, byte, char, float, double 및 boolean)인 경우 전체 배열을 가져와 설정하는 호출은 변환되지 않습니다. 이 경우 수동 이주 시 나머지 스니펫에서 사용하기 위해 해당 Java 랩퍼 클래스(Integer, Long, Short, Byte, Character, Float, Double 및 Boolean)에서/로부터 기본요소를 랩 또는 랩 해제하는 루프를 추가해야 할 수도 있습니다.
- 복합 유형을 표시하는 생성된 클래스의 인스턴스화 - 5.1에서, 응용프로그램에 정의되는 복합 유형의 생성된 클래스는 기본 비 인수 생성자를 사용하여 Java 스니펫에서 쉽게 인스턴스화될 수 있습니다. 이에 대한 예제는 다음과 같습니다.

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

6.0에서는, 특수 팩토리 클래스를 사용하여 이들 유형을 인스턴스화해야 하거나, 포함 유형의 인스턴스를 사용하여 하위 유형을 작성할 수 있습니다. BPEL 프로세스 변수 InputVariable이 유형 InputMessage를 갖는 것으로 정의된 경우, 이전 스니펫의 6.0 버전은 다음과 같습니다.

```
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");
```

스니펫 변환기에서 위와 같이 변경하려고 하지만 원래 인스턴스화가 발생한 순서가 parent-then-child 패턴을 따르지 않는 경우 수동으로 이주해야 합니다. 즉, 변환기에서는 스니펫의 인스턴스화 명령문을 지능적으로 재정렬하지 않습니다.

- WebSphere Business Integration Server Foundation 5.1에서 동적 참조는 다음과 같이 EndpointReferenceType 유형의 WSDL 메시지 파트로 또는 이름 공간의 EndpointReference 요소로 표시됩니다.

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

이런 참조는 표준 비즈니스 프로세스 이름 공간에서 표준 service-ref 요소 유형으로 이주됩니다.

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

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

모든 참조를 제대로 해석하도록 이러한 스키마 정의를 사용자 프로젝트로 수동으로 가져오기에 대한 지시사항은 BPEL 편집기 문서를 참조하십시오.

- **BPEL 변수 메시지 유형** - Java 스니펫에서 사용되는 모든 BPEL 변수에 대해 WSDL 메시지 유형을 지정해야 합니다. "messageType" 속성을 지정하지 않고 BPEL 변수를 액세스하는 Java 스니펫은 수정할 수 없습니다.

주의사항

본 IBM 제품에 포함된 The XDoclet Documentation은 사용 허가를 취득하였으며 다음 저작권 표시가 적용됩니다. Copyright (c) 2000-2004, XDoclet Team. All rights reserved.

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

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

이 정보는 미국에서 제공되는 제품 및 서비스용으로 작성된 것입니다. IBM은 다른 국가에서 이 책에 기술된 제품, 서비스 또는 기능을 제공하지 않을 수도 있습니다. 현재 사용할 수 있는 제품 및 서비스에 대한 정보는 한국 IBM 담당자에게 문의하십시오. 이 책에서 IBM 제품, 프로그램 또는 서비스를 언급했다고 해서 해당 IBM 제품, 프로그램 또는 서비스만을 사용할 수 있다는 것을 의미하지는 않습니다. IBM의 지적 재산을 침해하지 않는 한, 기능상으로 동등한 제품, 프로그램 또는 서비스를 대신 사용할 수도 있습니다. 비IBM 제품, 프로그램 또는 서비스의 운영에 대한 평가 및 검증은 사용자의 책임입니다.

IBM은 이 책에서 다루고 있는 특정 내용에 대해 특허를 보유하고 있거나 현재 특허 출원 중일 수 있습니다. 이 책을 제공한다고 해서 특허에 대한 라이선스까지 부여하는 것은 아닙니다. 라이선스에 대한 의문사항은 다음으로 문의하십시오.

135-270
서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터
전화번호: 080-023-8080

2바이트(DBCS) 정보에 관한 라이선스 문의는 한국 IBM 고객만족센터에 문의하거나 다음 주소로 서면 문의하시기 바랍니다.

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

다음 단락은 현지법과 상충하는 영국이나 기타 국가에서는 적용되지 않습니다. IBM은 타인의 권리 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여(단, 이에 한하지 않음) 묵시적이든 명시적이든 어떠한 종류의 보증없이 이 책을 "현상태대로" 제공합니다. 일부 국가에서는 특정 거래에서 명시적 또는 묵시적 보증의 면책사항을 허용하지 않으므로, 이 사항이 적용되지 않을 수도 있습니다.

이 정보에는 기술적으로 부정확한 내용이나 인쇄상의 오류가 있을 수 있습니다. 이 정보는 주기적으로 변경되며, 이 변경사항은 최신판에 통합됩니다. IBM은 이 책에서 설명한 제품 및/또는 프로그램을 사전 통고없이 언제든지 개선 및/또는 변경할 수 있습니다.

이 정보에서 비IBM의 웹 사이트는 단지 편의상 제공된 것으로, 어떤 방식으로든 이들 웹 사이트를 옹호하고자 하는 것은 아닙니다. 해당 웹 사이트의 자료는 본 IBM 제품 자료의 일부가 아니므로 해당 웹 사이트 사용으로 인한 위험은 사용자 본인이 감수해야 합니다.

IBM은 귀하의 권리를 침해하지 않는 범위 내에서 적절하다고 생각하는 방식으로 귀하가 제공한 정보를 사용하거나 배포할 수 있습니다.

(1) 독립적으로 작성된 프로그램과 기타 프로그램(본 프로그램 포함) 간의 정보 교환 및 (2) 교환된 정보의 상호 이용을 목적으로 정보를 원하는 프로그램 라이선스 사용자는 다음 주소로 문의하십시오.

135-270

서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터

이러한 정보는 해당 조항 및 조건에 따라(예를 들면, 사용료 지불 포함) 사용할 수 있습니다.

이 정보에 기술된 라이선스가 있는 프로그램 및 이 프로그램에 대해 사용 가능한 모든 라이선스가 있는 자료는 IBM이 IBM 기본 계약, IBM 프로그램 라이선스 계약(IPLA) 또는 이와 동등한 계약에 따라 제공한 것입니다.

본 문서에 포함된 모든 성능 데이터는 제한된 환경에서 산출된 것입니다. 따라서 다른 운영 환경에서 얻어진 결과는 상당히 다를 수 있습니다. 일부 성능은 개발 레벨 상태의 시스템에서 측정되었을 수 있으므로 이러한 측정치가 일반적으로 사용되고 있는 시스템에서도 동일하게 나타날 것이라고는 보증할 수 없습니다. 또한, 일부 성능은 추정치일 수도 있으므로 실제 결과는 다를 수 있습니다. 이 문서의 사용자는 해당 데이터를 사용자의 특정 환경에서 검증해야 합니다.

비IBM 제품에 관한 정보는 해당 제품의 공급업체, 공개 자료 또는 기타 범용 소스로부터 얻은 것입니다. IBM에서는 이러한 제품들을 테스트하지 않았으므로, 비IBM 제품과 관련된 성능의 정확성, 호환성 또는 기타 청구에 대해서는 확신할 수 없습니다. 비IBM 제품의 성능에 대한 의문사항은 해당 제품의 공급업체에 문의하십시오.

IBM의 향후 방향 또는 의도에 관한 모든 언급은 별도의 통지없이 변경될 수 있습니다.

이 정보에는 일상의 비즈니스 운영에서 사용되는 자료 및 보고서에 대한 예제가 들어 있습니다. 이들 예제에는 개념을 가능한 완벽하게 설명하기 위해 개인, 회사, 상표 및 제품의 이름이 사용될 수 있습니다. 이들 이름은 모두 가공의 것이며 실제 기업의 이름 및 주소와 유사하더라도 이는 전적으로 우연입니다.

저작권 라이선스:

이 정보에는 여러 운영 플랫폼에서의 프로그래밍 기법을 보여주는 원어로 된 샘플 응용프로그램이 들어 있습니다. 귀하는 이러한 샘플 프로그램의 작성 기준이 된 운영 플랫폼의 응용프로그램 프로그래밍 인터페이스(API)에 부합하는 응용프로그램을 개발, 사용, 판매 또는 배포할 목적으로 추가 비용없이 이들 샘플 프로그램을 어떠한 형태로든 복사, 수정 및 배포할 수 있습니다. 이러한 샘플 프로그램은 모든 조건하에서 완전히 테스트된 것은 아닙니다. 따라서 IBM은 이들 샘플 프로그램의 신뢰성, 서비스 가능성 또는 기능을 보증하거나 암시하지

않습니다. 귀하는 IBM의 응용프로그램 프로그래밍 인터페이스(API)에 부합하는 응용프로그램을 개발, 사용, 판매 또는 배포할 목적으로 추가 비용없이 이러한 샘플 응용프로그램을 어떠한 형태로든 복사, 수정 및 배포할 수 있습니다.

이러한 샘플 프로그램 또는 파생 제품의 각 사본이나 그 일부에는 반드시 다음과 같은 저작권 표시가 포함되어야 합니다.

(C) (귀하의 회사명) (연도). 이 코드의 일부는 IBM Corp의 샘플 프로그램에서 파생됩니다. (C) Copyright IBM Corp. 2000, 2005. All rights reserved.

이 정보를 소프트카피로 보는 경우에는 사진과 컬러 삽화가 제대로 나타나지 않을 수도 있습니다.

프로그래밍 인터페이스 정보

프로그래밍 인터페이스 정보는 본 프로그램을 사용하는 응용프로그램 소프트웨어 작성을 돕기 위해 제공됩니다.

귀하는 범용 프로그래밍 인터페이스를 통해 본 프로그램 톨의 서비스를 제공하는 응용프로그램 소프트웨어를 작성할 수 있습니다.

그러나 본 정보에는 진단, 수정 및 성능 조정 정보도 포함되어 있습니다. 진단, 수정 및 성능 조정 정보는 응용 프로그램 소프트웨어의 디버그를 돕기 위해 제공된 것입니다.

경고: 본 진단, 수정 및 조정 정보는 변경될 수 있으므로 프로그램 인터페이스로서 사용될 수 없습니다.

상표 및 서비스표

다음 웹 사이트 <http://www.ibm.com/legal/copytrade.shtml>을 참조하십시오.

귀하의 의견은 저희에게 매우 소중한 것이며, 고객 여러분들께 보다 좋은 제품을 제공해드리기 위해 최선을 다하겠습니다.



SA30-2703-01

