IBM Integration Bus

Using IIB Embedded Global Cache

Featuring:

Add operations to an Integration Service
Use JAXB classes to:

- Load database records to the embedded Global Cache
- Access data from the Global Cache

June 2015
Hands-on lab built at product
Version 10.0.0.0

Contents

1. Introduction

In an earlier exercise you built the Employee Service to retrieve EMPLOYEE data from the SAMPLE database.

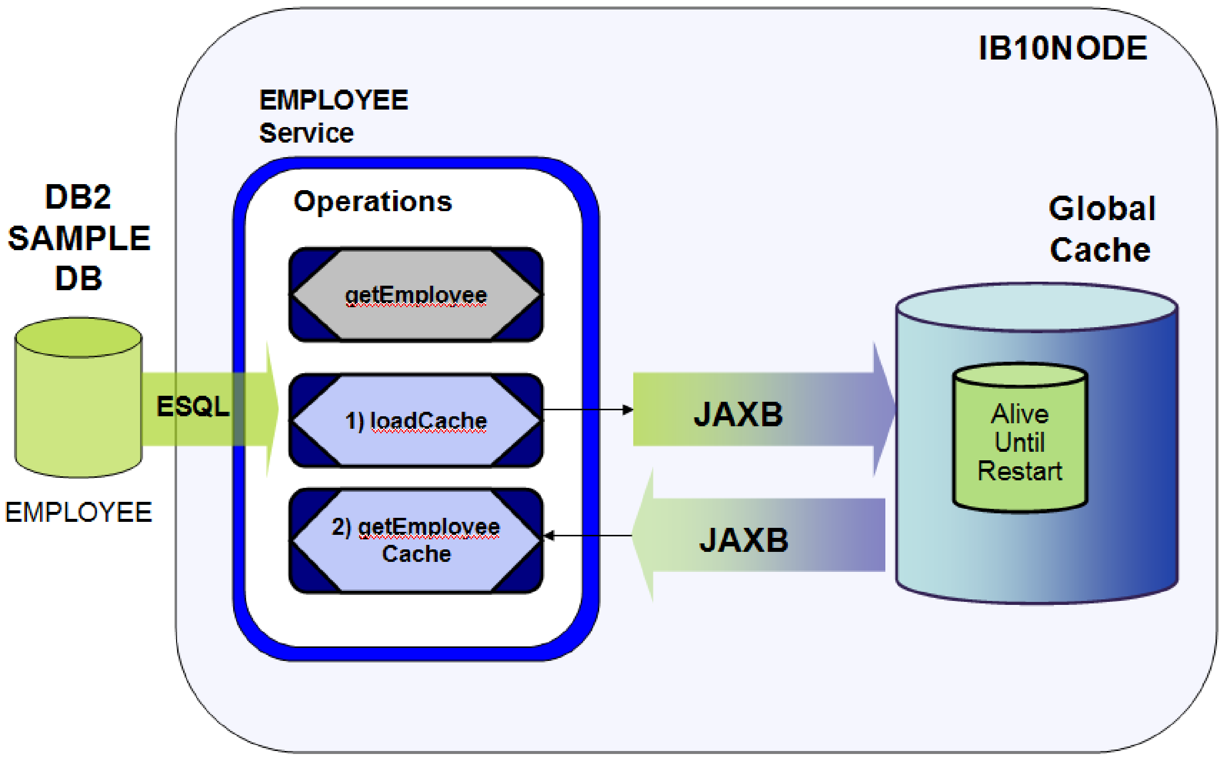
This lab guide will show you how to extend the Employee Service by adding operations that will load the data from the SAMPLE database to computer memory and allow it to be retrieved from there for improved performance.

IIB embedded global cache

- an IBM WebSphere eXtreme Scale grid designed and optimized for use within and between integration nodes and is supplied as part of IBM Integration Bus.
- “a repository for data that you want to reuse”.

IBM WebSphere eXtreme Scale

- “elastic, scalable, in-memory data grid (IMDG) technology”.



Java Architecture for XML Binding (JAXB) classes are generated by the Integration Toolkit to provide an easy to use programming interface to the stored data.

The EmployeeService will be modified to add a further two operations:

1) **To load data from the EMPLOYEE table (DB2 SAMPLE database) into a WebSphere eXtreme Scale (WXS) cache using JAXB classes**

   The “loadData” operation will use ESQL to read data (rows) from the EMPLOYEE table and pass this data to a Java program. The Java program will then formulate the data into name/ value pairs, the value will be written to the WXS cache as a JAXB class (complex type) enabling easy access to the data.

2) **To access data from this cached table using JAXB classes in a Java compute node.**

The “getEmployeeCache” operation will show how an Integration Service can easily access the data stored as a complex type in the WXS cache using an IIB Java compute node.

The lab will guide you through how the Integration Toolkit provides a facility to easily create Java XML Bindings related to an existing message model along with Java code to access this data.

2. Pre-requisite tasks

2.1 Import Employee Service

This lab guide builds on the operations created in EmployeeService. You will now import a completed solution (with one operation) into your current environment. From a run time perspective the service will run on **IB10NODE** on Integration Server "**server1**".

1. Using Integration Toolkit create a new workspace called "**`EmployeeServiceGlobalCache`**".

2. Right click on the Application Development window and click import.

   Import the PI files EmployeeServiceInterfaceV10.zip and EmployeeServiceV10.zip from **`C:\student10\Integration_service\solution`**

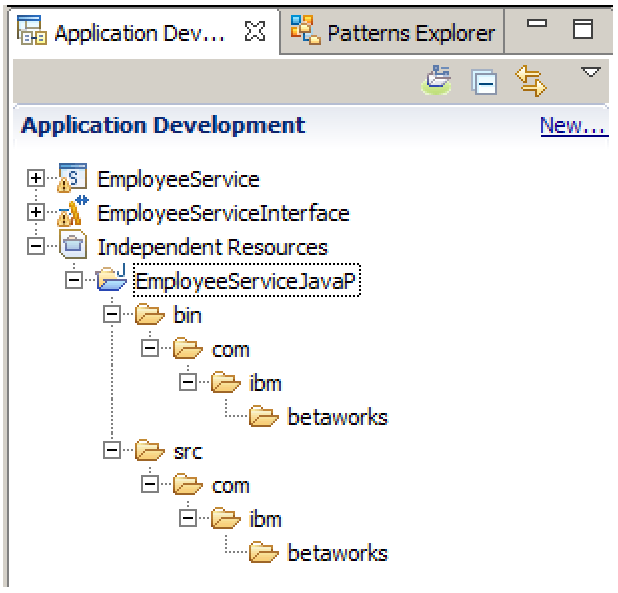
2.2 Generate the JAXB Java Object Classes for the data

The following section will prepare the JAXB classes for use within the operations that you will add later in the guide. The Integration Toolkit provides a "Generate JAXB Object classes" feature that automatically generates the JAXB object classes you will need based on the schemas you will be working with. You will now use this feature to generate the JAXB object classes for the EMPLOYEE data.

1. Import the PI file called **`EmployeeServiceJavaP.zip`** from **`c:\student10\integration_service_globalcache\resources\`**

   The PI file contains a Java Project that we will use throughout this lab guide. (*The Appendix for this lab guide gives instructions on how this Java project was created*).

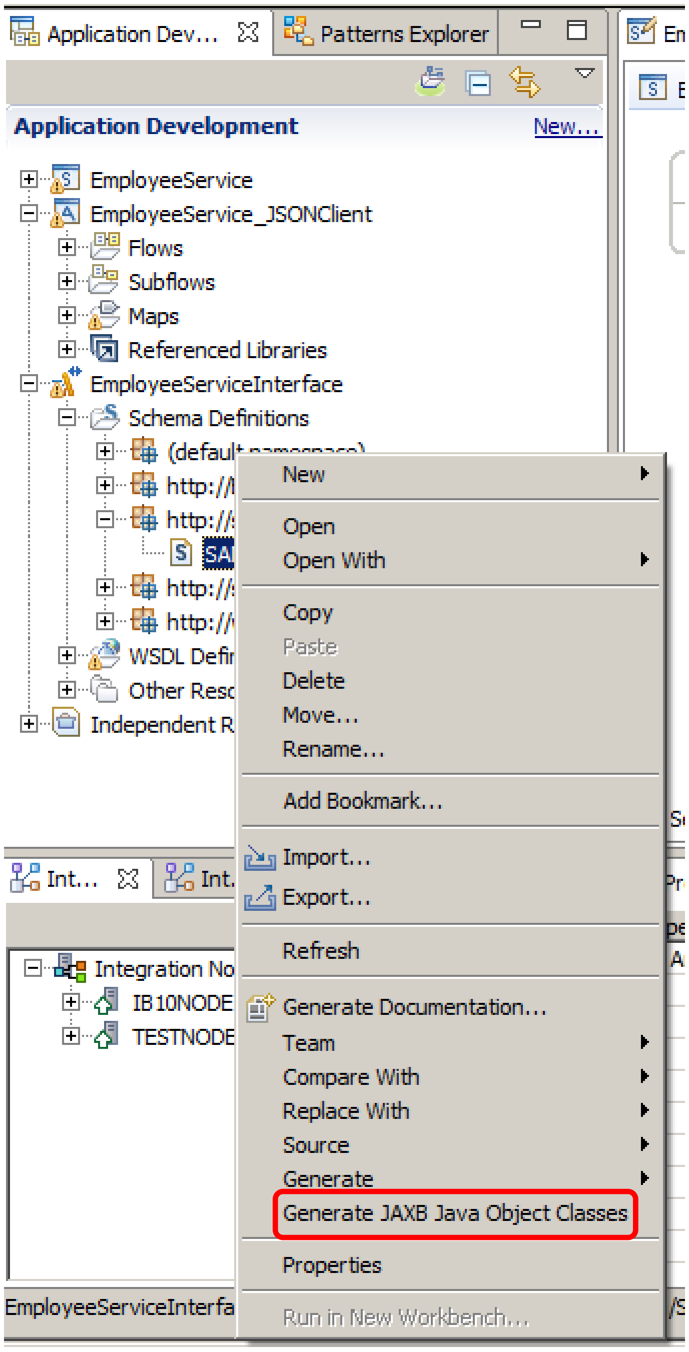
   The Java Project will appear in "Independent Resources". Note the project is created with com.ibm.betaworks defined as a Java package:



```
EmployeeService
EmployeeServiceInterface
Independent Resources
  EmployeeServiceJavaP
    bin
      com
        ibm
          betaworks
    src
      com
        ibm
          betaworks
```

2. Expand the Schema definitions in the library called “EmployeeServiceInterface”.

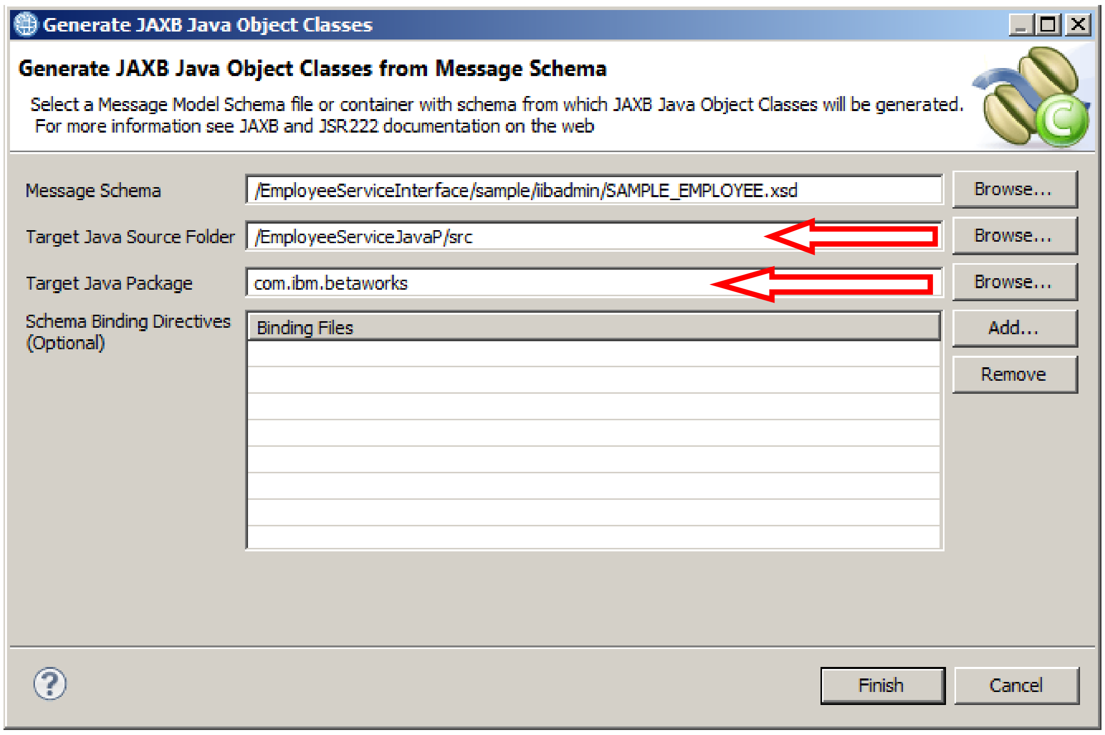
   Right click on the schema “SAMPLE_EMPLOYEE” under namespace “http://sample/iibadmin” and click “Generate JAXB Java Object Classes” from the context menu:



This will open the “Generate JAXB Java Object Classes from Message Schema“ wizard.

3. In the wizard, use the Browse buttons to specify the **Target Java Source Folder** of "/EmployeeServiceJavaP/src" and **Target Java Package** of "com.ibm.betaworks" and click Finish.

   Note: there will be no response from the wizard.



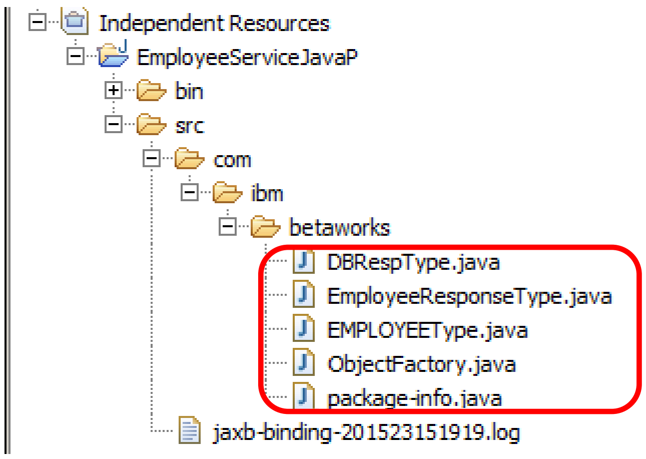
Generate JAXB Java Object Classes

Generate JAXB Java Object Classes from Message Schema

Select a Message Model Schema file or container with schema from which JAXB Java Object Classes will be generated.
For more information see JAXB and JSR222 documentation on the web

Message Schema	/EmployeeServiceInterface/sample/iibadmin/SAMPLE_EMPLOYEE.xsd
Target Java Source Folder	/EmployeeServiceJavaP/src
Target Java Package	com.ibm.betaworks
Schema Binding Directives (Optional)	Binding Files

4. The wizard generates 5 Java source files in the EmployeeServiceJavaP project:



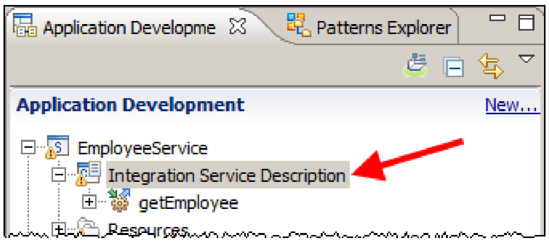
```
Independent Resources
  EmployeeServiceJavaP
    bin
    src
      com
        ibm
          betaworks
            DBRespType.java
            EmployeeResponseType.java
            EMPLOYEEType.java
            ObjectFactory.java
            package-info.java
      jaxb-binding-201523151919.log
```

3. Add Cache operations to Employee Service

3.1 Add loadCache operation

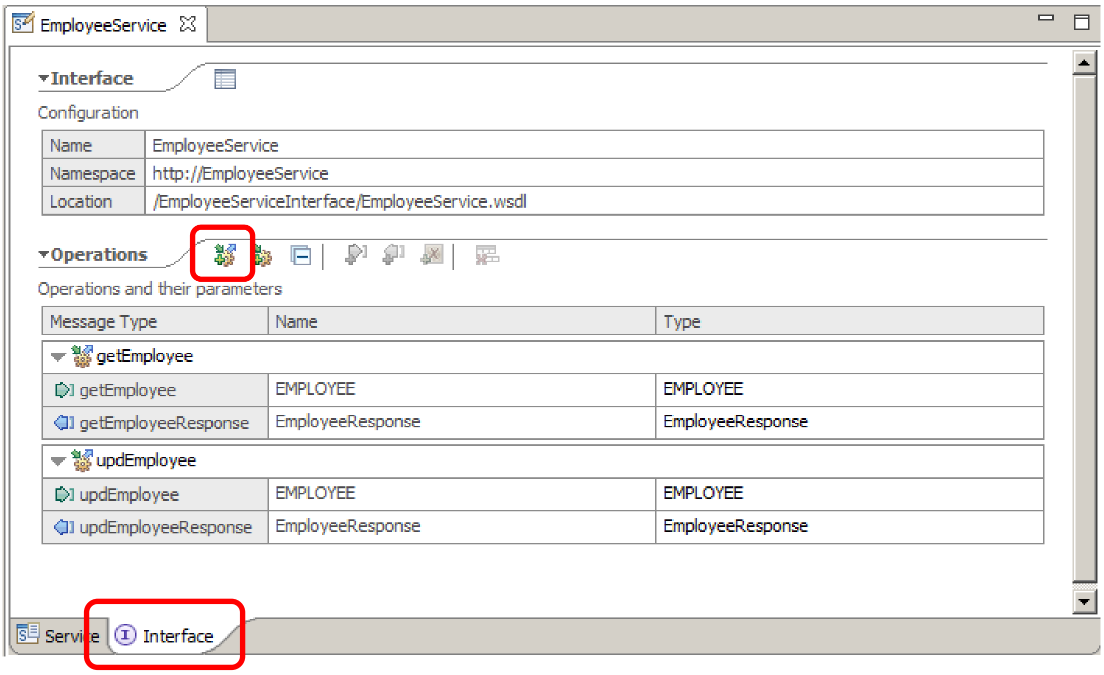
An additional operation will now be added to the EmployeeService. The two way operation will be used to load the data from the EMPLOYEE table into the IIB (embedded) Global Cache.

1. In the Application Development tab, expand the EmployeeService and open the Integration Service Description by double clicking on it:



2. Click on the Interface tab to show the operations currently available. You will see the operations that were added when the EmployeeService was first created

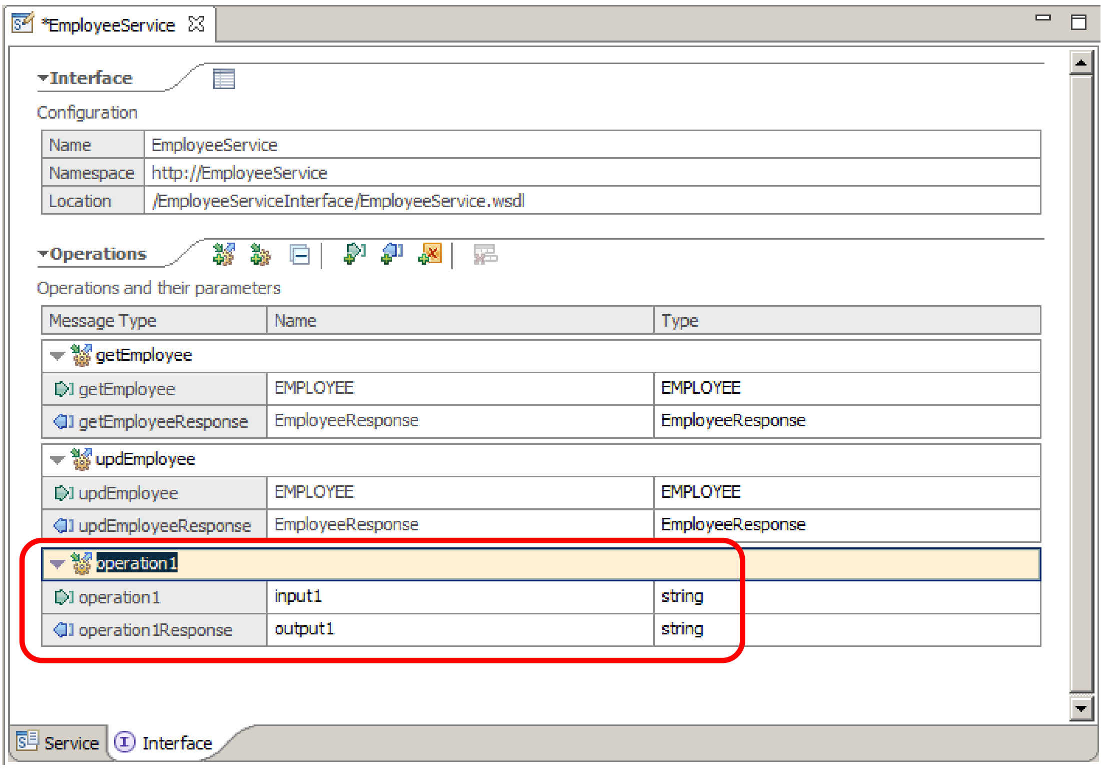
   Click the "Add Request Response Operation" icon to add an operation:



Name	EmployeeService
Namespace	http://EmployeeService
Location	/EmployeeServiceInterface/EmployeeService.wsdl

Message Type	Name	Type
getEmployee		
getEmployee	EMPLOYEE	EMPLOYEE
getEmployeeResponse	EmployeeResponse	EmployeeResponse
updEmployee		
updEmployee	EMPLOYEE	EMPLOYEE
updEmployeeResponse	EmployeeResponse	EmployeeResponse

3. An additional operation will be displayed in the list of operations with default values:



Configuration

Name	EmployeeService
Namespace	http://EmployeeService
Location	/EmployeeServiceInterface/EmployeeService.wsdl

Operations and their parameters

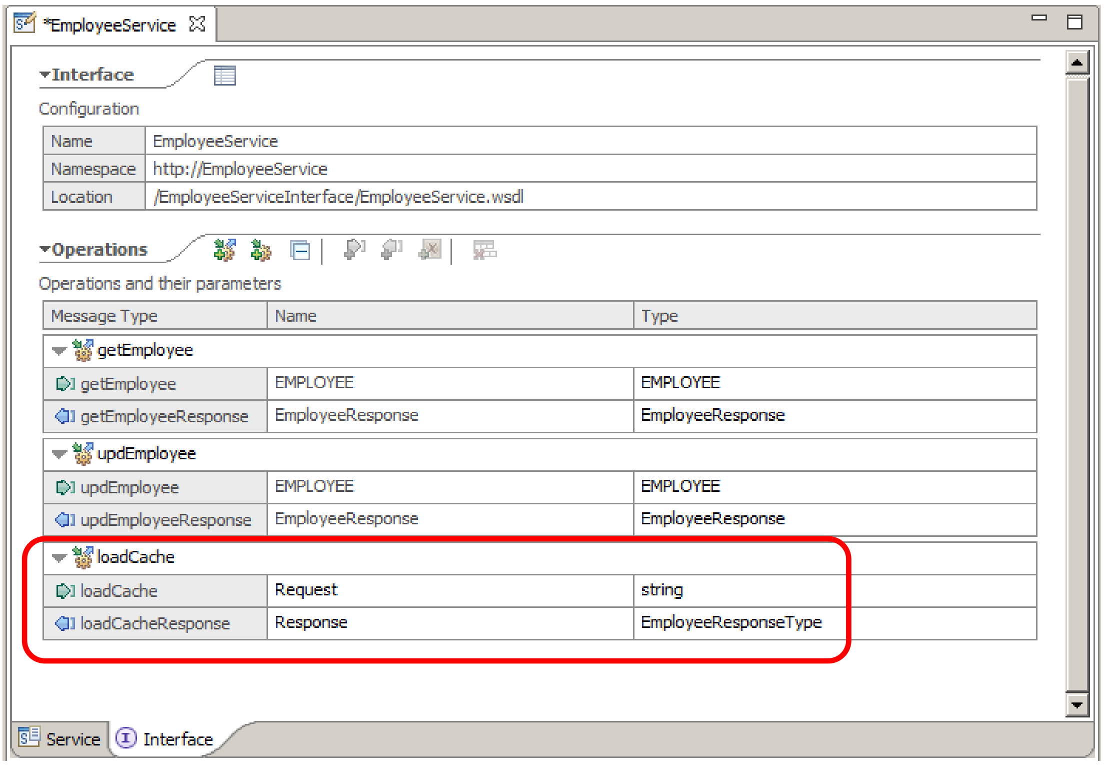
Message Type	Name	Type
getEmployee		
getEmployee	EMPLOYEE	EMPLOYEE
getEmployeeResponse	EmployeeResponse	EmployeeResponse
updEmployee		
updEmployee	EMPLOYEE	EMPLOYEE
updEmployeeResponse	EmployeeResponse	EmployeeResponse
operation1		
operation1	input1	string
operation1Response	output1	string

4. Overtype the default name “operation1” with “loadCache”

Notice that the operation request and response names automatically change with the name change after pressing Enter.

5. Set the **Name** and **Type** values as shown below, ensuring you preserve the case of the text as this is critical to the Java code that you will add later in the lab guide.

- Change loadCache input name to “Request”
- Leave loadCache input type as “string”
- Change loadCacheResponse output name to “Response”
- Change loadCacheResponse output type to “EmployeeResponseType” (*click on the word string” and choose “EmployeeResponseType” from the list:*)



Configuration

Name	EmployeeService
Namespace	http://EmployeeService
Location	/EmployeeServiceInterface/EmployeeService.wsdl

Operations and their parameters

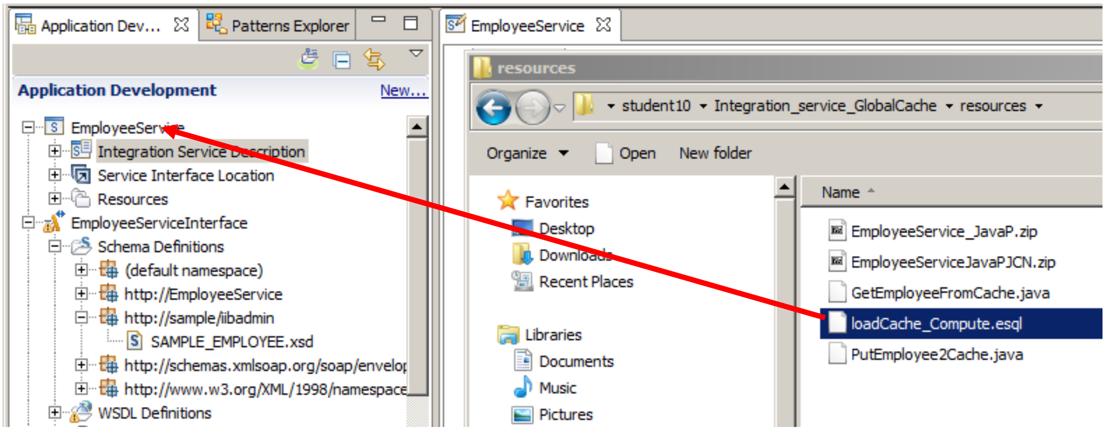
Message Type	Name	Type
getEmployee		
getEmployee	EMPLOYEE	EMPLOYEE
getEmployeeResponse	EmployeeResponse	EmployeeResponse
updEmployee		
updEmployee	EMPLOYEE	EMPLOYEE
updEmployeeResponse	EmployeeResponse	EmployeeResponse
loadCache		
loadCache	Request	string
loadCacheResponse	Response	EmployeeResponseType

6. Save the Integration Service (<ctrl> S).

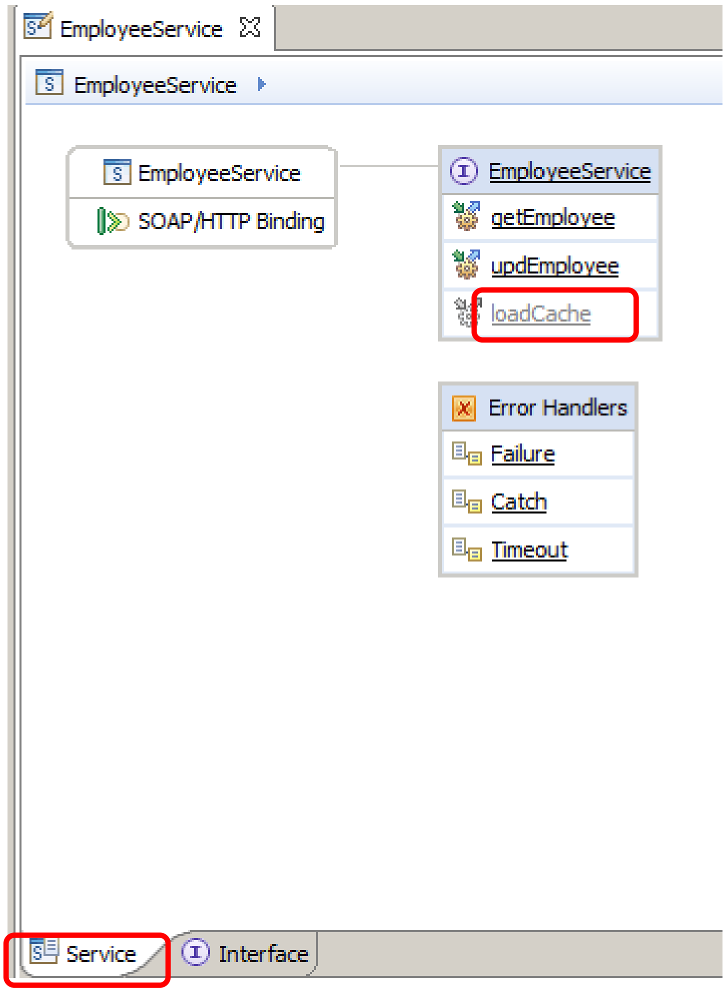
3.2 Complete loadCache operation subflow

You will now add a compute node and the required ESQL code to the loadCache operation.

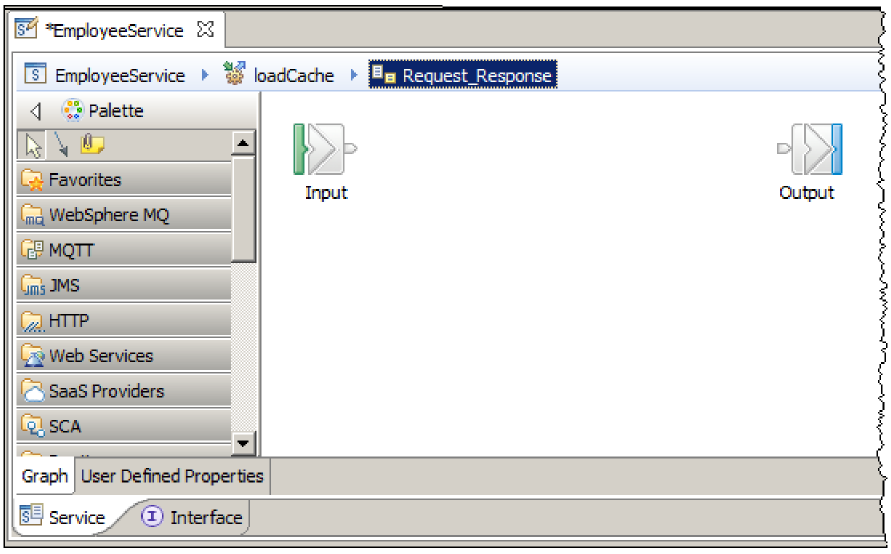
1. In Windows Explorer, navigate to `C:\student10\Integration_service_GlobalCache\resources` and drag and drop `loadCache_Compute.esql` onto EmployeeService:



2. In the Integration Service Description for EmployeeService, select the Service tab, then click loadCache (*loadCache will be greyed out as the subflow associated with that operation has not yet been defined*) :



3. A generated loadCache Request_Response subflow will open:



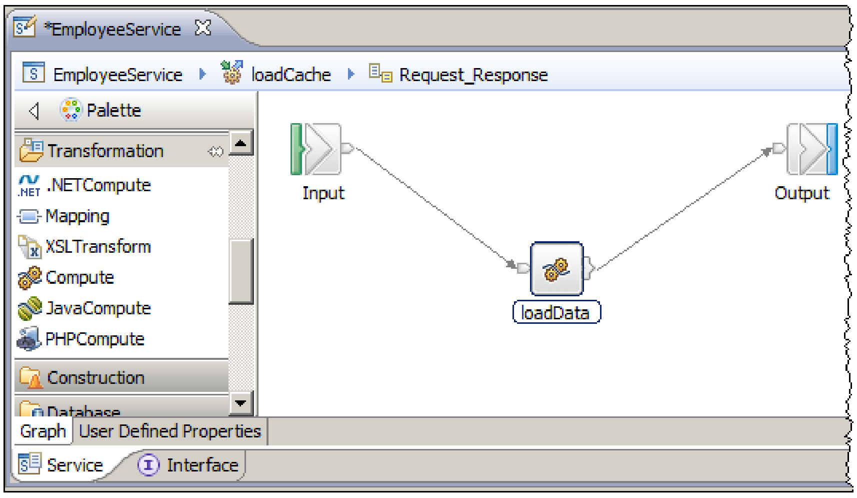
4. Add a compute node to the canvas.

a) Open the Transformation folder in the Palette and add a "Compute node" to the canvas.

Call the compute node "loadData" (*please make sure you preserve the case*).

b) Join the input and output terminals via the loadData compute node (wire the "Out" terminal in the compute node to the Output label).

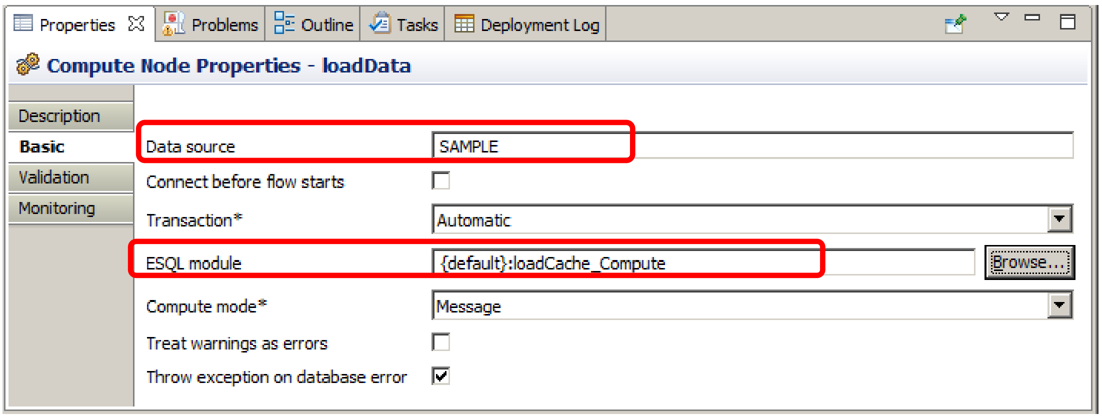
The resulting canvas should look like this:



5. Highlight the loadData compute node.

   a) In the properties tab specify the "`SAMPLE`" database as the Data source.

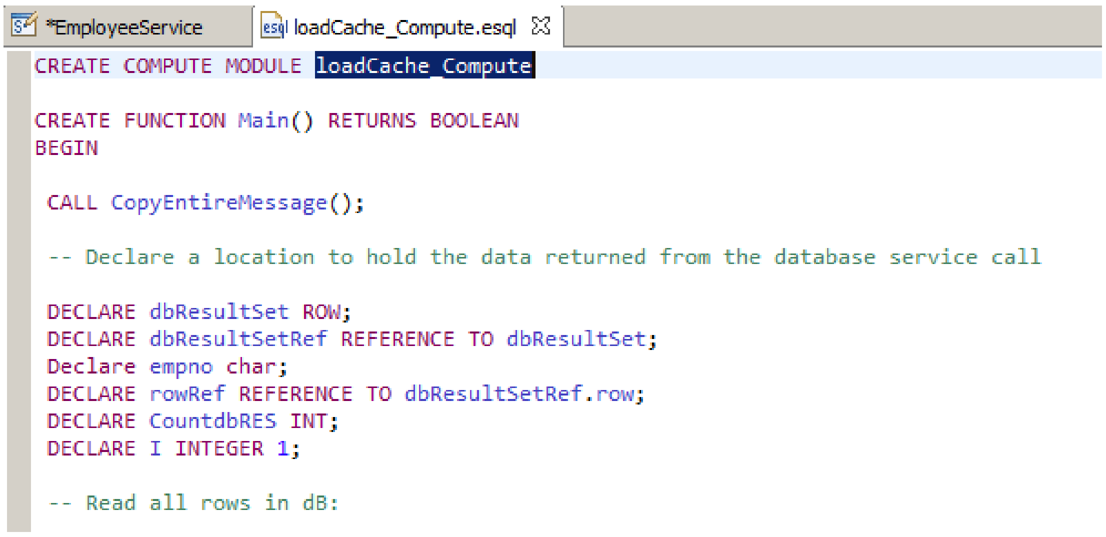
   b) Click Browse next to ESQL module and select `{default}:loadCache_Compute` (this is the module that has just been added).



Compute Node Properties - loadData

Data source	SAMPLE
Connect before flow starts	☐
Transaction*	Automatic
ESQL module	{default}:loadCache_Compute
Compute mode*	Message
Treat warnings as errors	☐
Throw exception on database error	✓

6. Double click the compute node. The ESQL code should be displayed like this:



```
CREATE COMPUTE MODULE loadCache_Compute

CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN

 CALL CopyEntireMessage();

 -- Declare a location to hold the data returned from the database service call

 DECLARE dbResultSet ROW;
 DECLARE dbResultSetRef REFERENCE TO dbResultSet;
 Declare empno char;
 DECLARE rowRef REFERENCE TO dbResultSetRef.row;
 DECLARE CountdbRES INT;
 DECLARE I INTEGER 1;

 -- Read all rows in dB:
```

The ESQL code reads through every record in the EMPLOYEE table and passes the data to the Java program

The ESQL code has a procedure call ("PutJAXB2CACHE") which is called for each row in the EMPLOYEE table. It is in this Java code where the EMPLOYEE data is built ready to add to the Global Cache).

Close the ESQL code panel.

7. Save and close the EmployeeService <ctrl S>

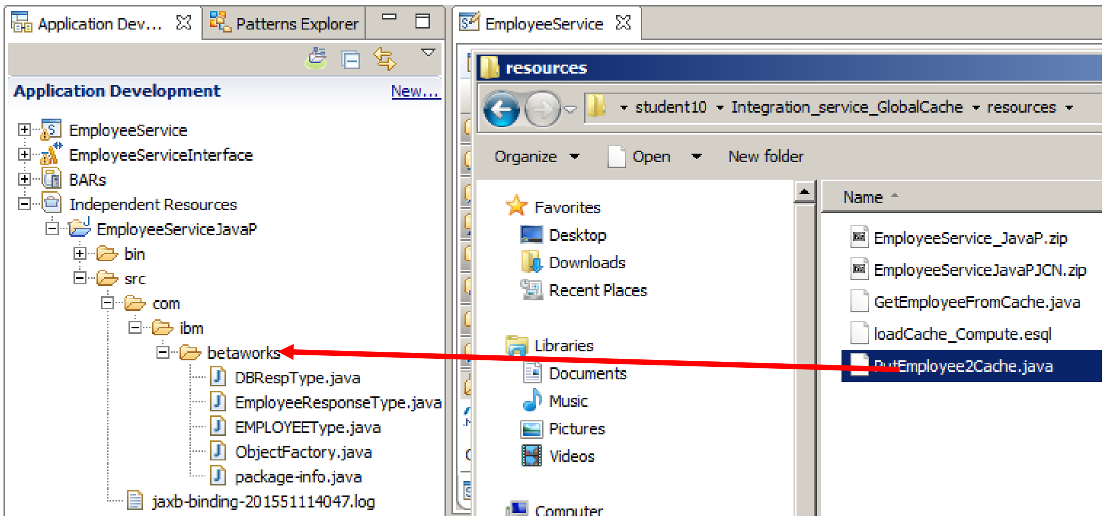
3.3 Add “PutJAXB2CACHE” Java code

PutJAXB2CACHE Java code is required because Java is the current programming language able to access the embedded IIB cache (calls to the embedded cache are not possible directly from ESQL code).

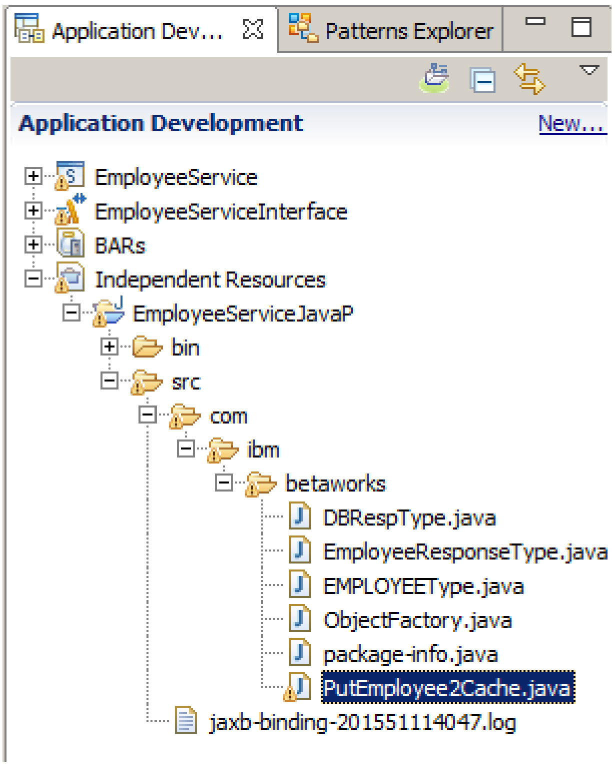
The ESQL code you added gets every database record from the EMPLOYEE table and calls this Java program to add each row to the Global Cache as a complex type.

You will now add the Java code that will be called by the ESQL code you defined in the previous section.

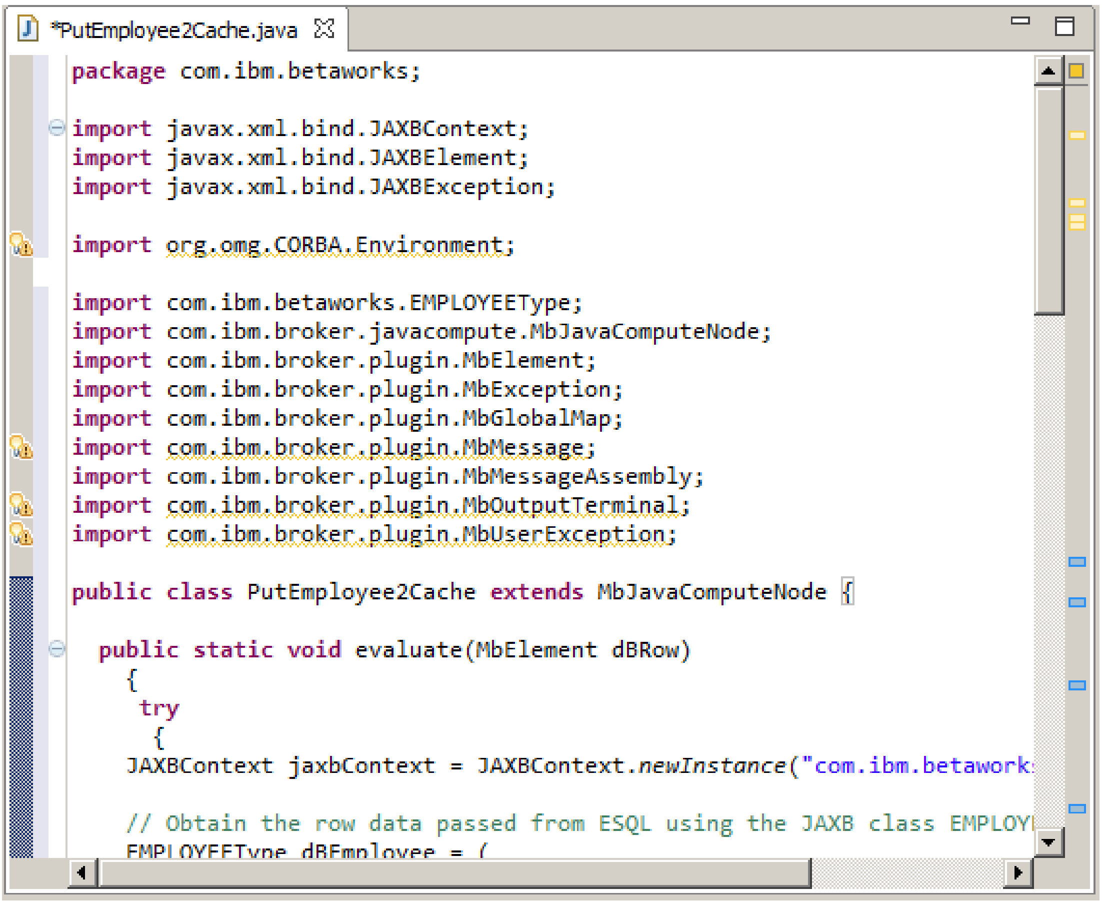
1. In Windows Explorer, navigate to `C:\student10\Integration_service_GlobalCache\resources` and drag and drop `PutEmployee2Cache.java` onto `src.com.ibm.betaworks` in EmployeeServiceJavaP:



2. The java code will be added to the project. Double click it:



3. The Java editor will open. The code will look like this. Close the editor when you are ready.



```
package com.ibm.betaworks;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBElement;
import javax.xml.bind.JAXBException;

import org.omg.CORBA.Environment;

import com.ibm.betaworks.EMPLOYEEType;
import com.ibm.broker.javacompute.MbJavaComputeNode;
import com.ibm.broker.plugin.MbElement;
import com.ibm.broker.plugin.MbException;
import com.ibm.broker.plugin.MbGlobalMap;
import com.ibm.broker.plugin.MbMessage;
import com.ibm.broker.plugin.MbMessageAssembly;
import com.ibm.broker.plugin.MbOutputTerminal;
import com.ibm.broker.plugin.MbUserException;

public class PutEmployee2Cache extends MbJavaComputeNode {

  public static void evaluate(MbElement dBRow)
    {
     try
      {
    JAXBContext jaxbContext = JAXBContext.newInstance("com.ibm.betawork

    // Obtain the row data passed from ESQL using the JAXB class EMPLOYI
    EMPLOYEEType dBEmployee = (
```

4. Close the message flow editor.

You have now defined the required items to load data from the EMPLOYEE table to the embedded Cache.

3.4 Add getEmployeeCache operation

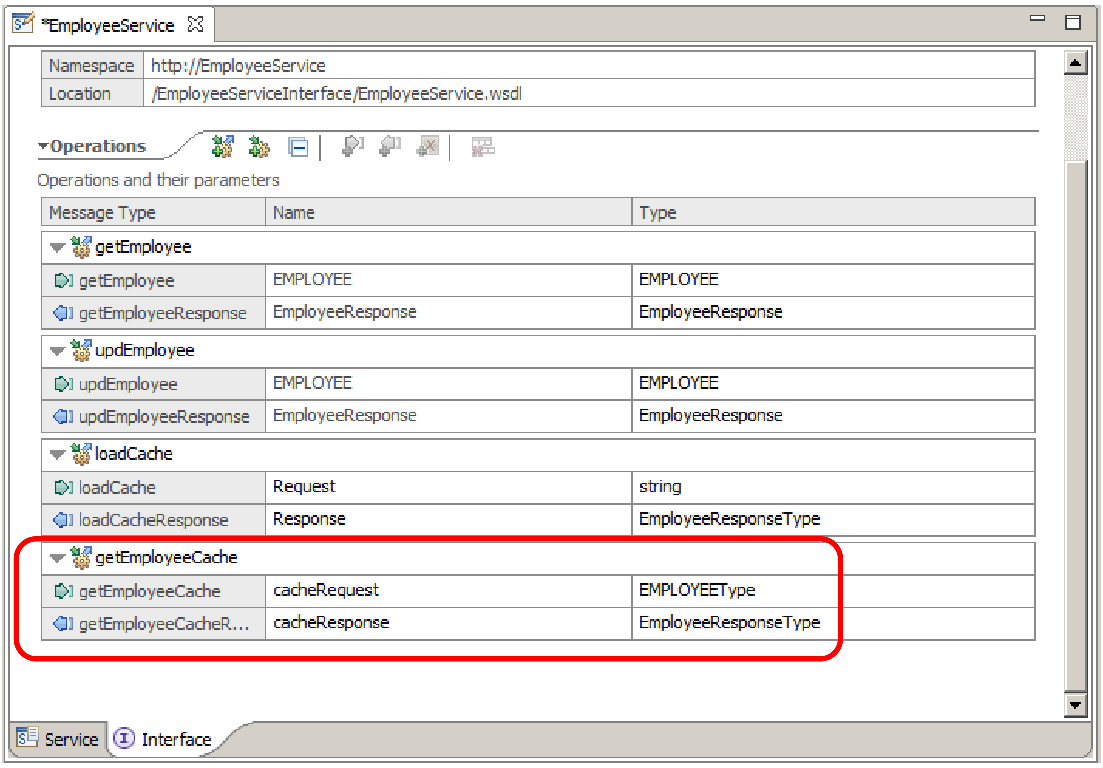
The “getEmployeeCache” operation will obtain the details of a specific employee from the data written to the IIB embedded cache by the “loadCache” operation.

You will now add this operation to the EmployeeService. This will be very similar to adding the loadCache operation above.

1. Open the “Integration Service Description” for the EmployeeService.

2. In the Interface tab, click the “Add Request Response operation” icon and add the “getEmployeeCache” operation by setting the **Name** and **Type** values as shown below (*Note: the case of the names is critical as they are used in the Java code, please make sure you preserve the case*):

   - Change getEmployeeCache input name to “cacheRequest”
   - Change getEmployeeCache input type to “EMPLOYEEType”
   - Change getEmployeeCacheResponse output name to “cacheResponse”
   - Change getEmployeeCacheResponse output type to “EmployeeResponseType”



Namespace	http://EmployeeService
Location	/EmployeeServiceInterface/EmployeeService.wsdl

Operations and their parameters

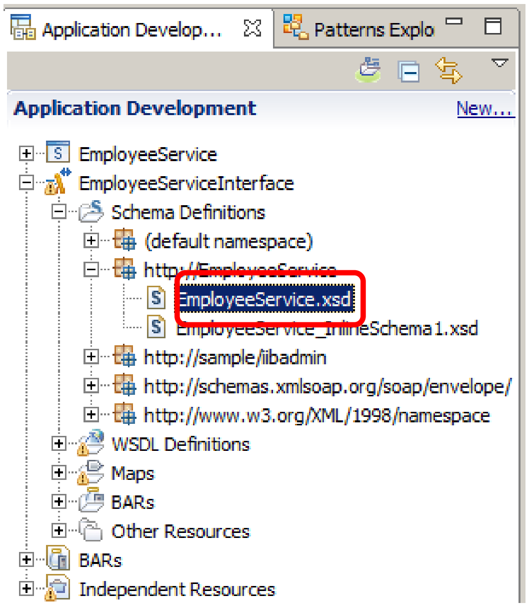
Message Type	Name	Type
getEmployee		
getEmployee	EMPLOYEE	EMPLOYEE
getEmployeeResponse	EmployeeResponse	EmployeeResponse
updEmployee		
updEmployee	EMPLOYEE	EMPLOYEE
updEmployeeResponse	EmployeeResponse	EmployeeResponse
loadCache		
loadCache	Request	string
loadCacheResponse	Response	EmployeeResponseType
getEmployeeCache		
getEmployeeCache	cacheRequest	EMPLOYEEType
getEmployeeCacheR...	cacheResponse	EmployeeResponseType

3. Save the EmployeeService (<ctrl> S).

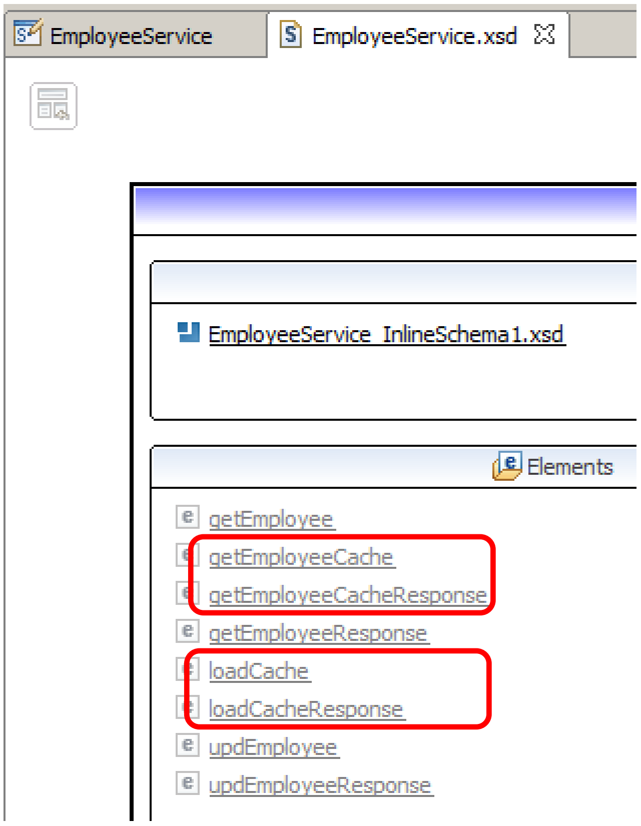
   Ensure you do this now. If you don’t save the service now the new operation will not be included when generating the JAXB classes in the next section.

3.5 Review automatic changes to EmployeeService schema

1. In the Application Development window, double click on EmployeeService.xsd:



2. The new Request/Response operation names you have defined have been added automatically as elements to the schema. Close the window when you are ready.



4. Configure JAXB object classes for the new service operations

4.1 Generate JAXB object classes

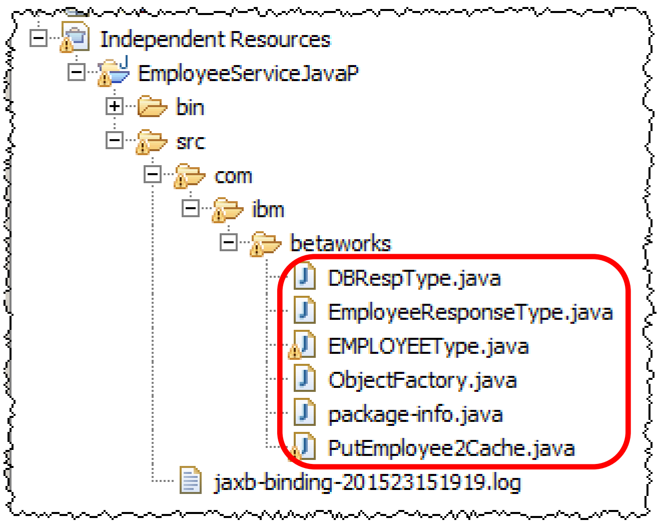
Adding the new operations has automatically updated the Schema for the EmployeeService with schema definitions for the input and output of each operation. The sample code provided with this lab guide uses JAXB object classes to "set" and "get" both input and output data passed to the operations. The JAXB "*setters*" and "*getters*" can be generated automatically by the Integration Toolkit based on the schema for the EmployeeService.

When we generated the JAXB object classes earlier we generated them for the SAMPLE schema (i.e. the data in the EMPLOYEE table).

You will now generate JAXB object classes based on the schemas that have been defined for the service operations above. You will do this now to avoid errors being produced when adding the Java code to retrieve the data from the cache.

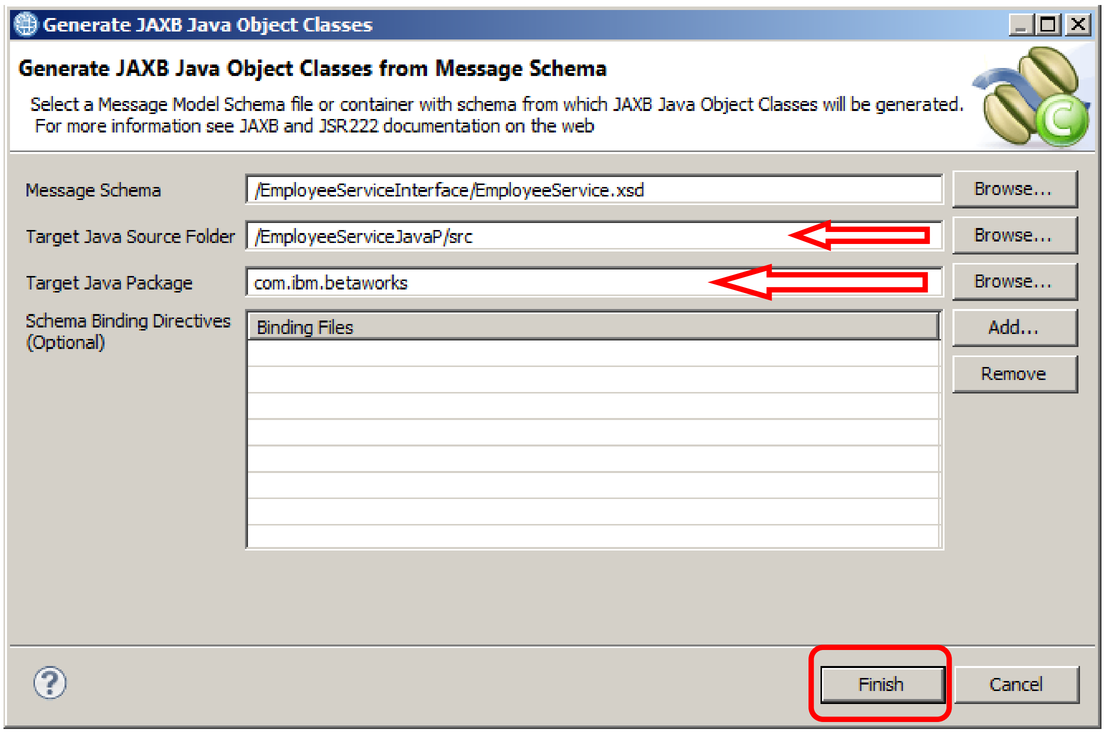
1. In the Integration Toolkit Application Development window, expand the EmployeeServiceJavaP project until you see all the Java code in package com.ibm.betaworks.

   Note there are 6 Java programs in the package.



2. In the Application Development, right click on "EmployeeService.xsd" and choose "Generate JAXB Java Object classes".

3. In the "Generate JAXB Java Object Classes from Message Schema" use the Browse button to specify the Target Java Source Folder and Package name as follows and click Finish.



Generate JAXB Java Object Classes

Generate JAXB Java Object Classes from Message Schema

Select a Message Model Schema file or container with schema from which JAXB Java Object Classes will be generated.
For more information see JAXB and JSR222 documentation on the web

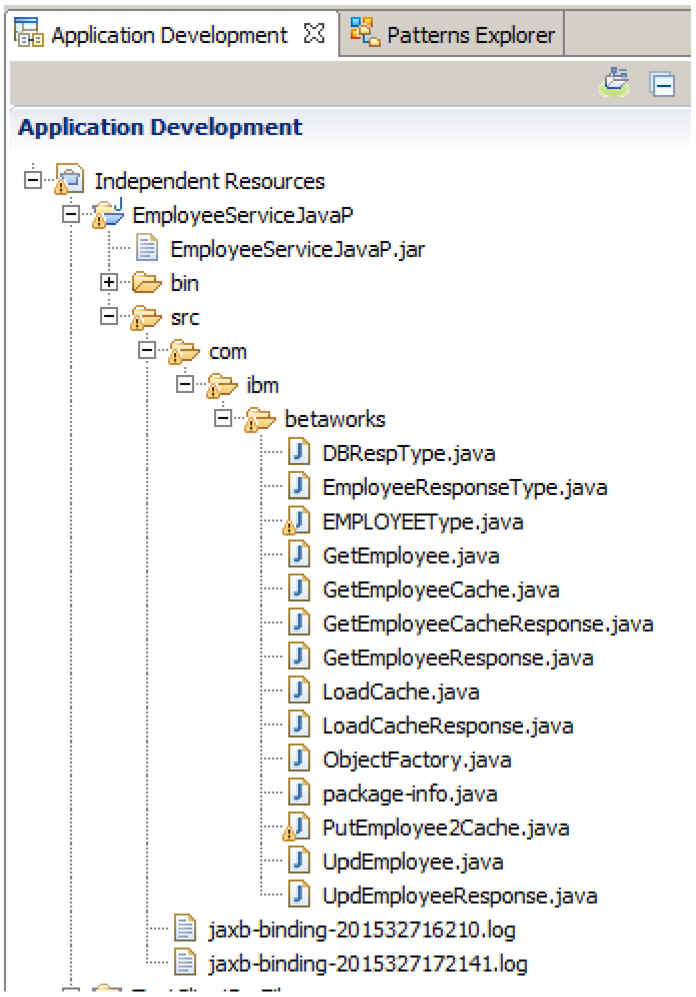
Message Schema: /EmployeeServiceInterface/EmployeeService.xsd

Target Java Source Folder: /EmployeeServiceJavaP/src

Target Java Package: com.ibm.betaworks

Schema Binding Directives (Optional): Binding Files

4. In the Application Development window, note an additional 8 Java programs (two for each operation – one for each request and one for each response in the operation) have been added to the com.ibm.betaworks package in the EmployeeServiceJavaP project. Some of these will be used by the “getEmployeeFromCache.java” program in the next section:



```
Independent Resources
  EmployeeServiceJavaP
    EmployeeServiceJavaP.jar
    bin
    src
      com
        ibm
          betaworks
            DBRespType.java
            EmployeeResponseType.java
            EMPLOYEEType.java
            GetEmployee.java
            GetEmployeeCache.java
            GetEmployeeCacheResponse.java
            GetEmployeeResponse.java
            LoadCache.java
            LoadCacheResponse.java
            ObjectFactory.java
            package-info.java
            PutEmployee2Cache.java
            UpdEmployee.java
            UpdEmployeeResponse.java
      jaxb-binding-201532716210.log
      jaxb-binding-2015327172141.log
```

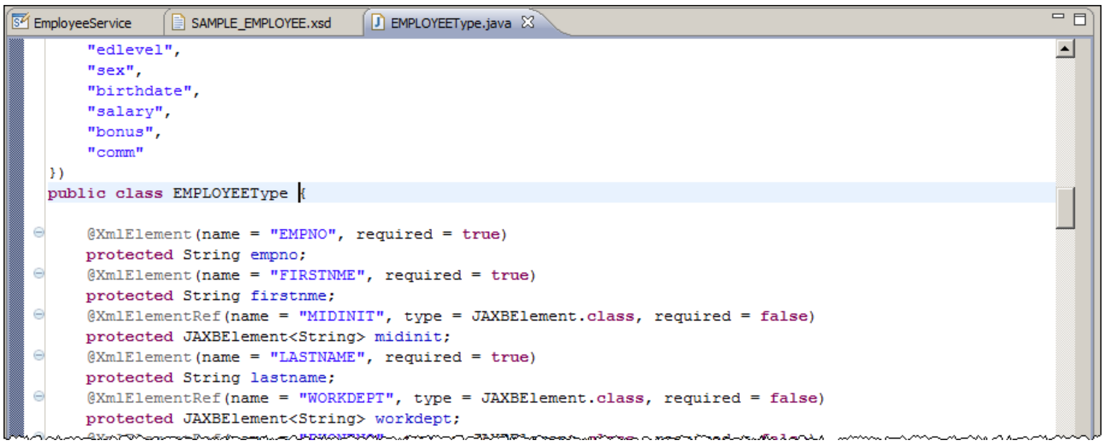
4.2 Define “implements Serializable” on the EMPLOYEEType class definition

The loadCache operation will write the data extracted from the DB2 EMPLOYEE table as a complex type defined by the generated JAXB class EMPLOYEEType to the Global Cache. This enables the getEmployeeFromCache operation to easily retrieve the data in a readily consumable format. This avoids needing to parse the data that is stored in the cache as we know the layout of the data is defined by the complex type EMPLOYEEType.

In order to do this, the EMPLOYEEType class needs to be defined as “Serializable”. You will now edit EMPLOYEEType.java to define it as Serializable.

Note: *If you Generate the JAXB classes again using Integration Toolkit (for example if you added a further operation), the change defined in this section will need to be performed again on the regenerated EMPLOYEEType.java code*.

1. Open the EMPLOYEEType.java code (double click on the name) and search for “class EMPLOYEEType”:

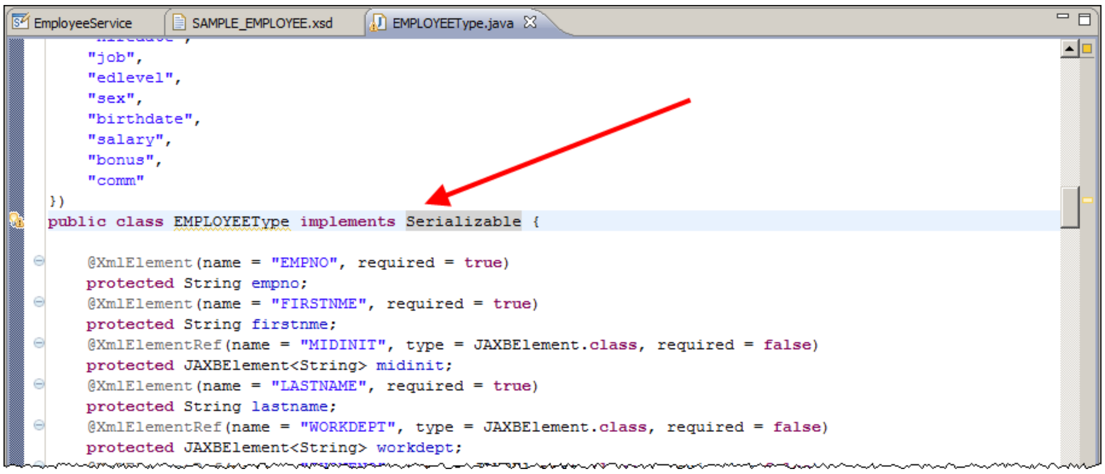


```
    "edlevel",
    "sex",
    "birthdate",
    "salary",
    "bonus",
    "comm"
})
public class EMPLOYEEType {

    @XmlElement(name = "EMPNO", required = true)
    protected String empno;
    @XmlElement(name = "FIRSTNME", required = true)
    protected String firstnme;
    @XmlElementRef(name = "MIDINIT", type = JAXBElement.class, required = false)
    protected JAXBElement<String> midinit;
    @XmlElement(name = "LASTNAME", required = true)
    protected String lastname;
    @XmlElementRef(name = "WORKDEPT", type = JAXBElement.class, required = false)
    protected JAXBElement<String> workdept;
```

Note the class is not defined as Serializable.

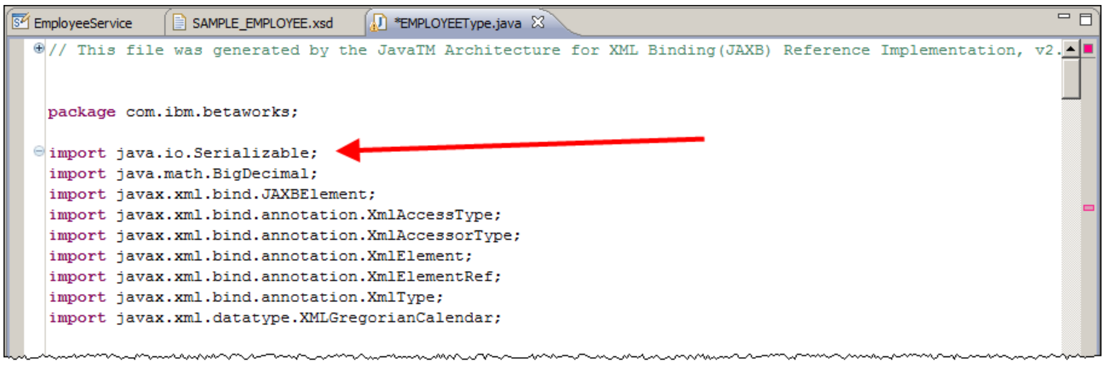
2. Use Control assist (<ctrl> and <space>) to add **`implements Serializable`** after the class name “EMPLOYEEType”

(*type “impl” then <ctrl><space>, this will add “implements” then add a space. Next type “Ser” and then <ctrl><space>, a list of options will appear, choose “Serializable” from the list of options*):



```
    "job",
    "edlevel",
    "sex",
    "birthdate",
    "salary",
    "bonus",
    "comm"
})
public class EMPLOYEEType implements Serializable {

    @XmlElement(name = "EMPNO", required = true)
    protected String empno;
    @XmlElement(name = "FIRSTNME", required = true)
    protected String firstnme;
    @XmlElementRef(name = "MIDINIT", type = JAXBElement.class, required = false)
    protected JAXBElement<String> midinit;
    @XmlElement(name = "LASTNAME", required = true)
    protected String lastname;
    @XmlElementRef(name = "WORKDEPT", type = JAXBElement.class, required = false)
    protected JAXBElement<String> workdept;
```

3. Go to the top of the code. You will see that "import java.io.Serializable" has been automatically added to the list of import statements at the top of the code:

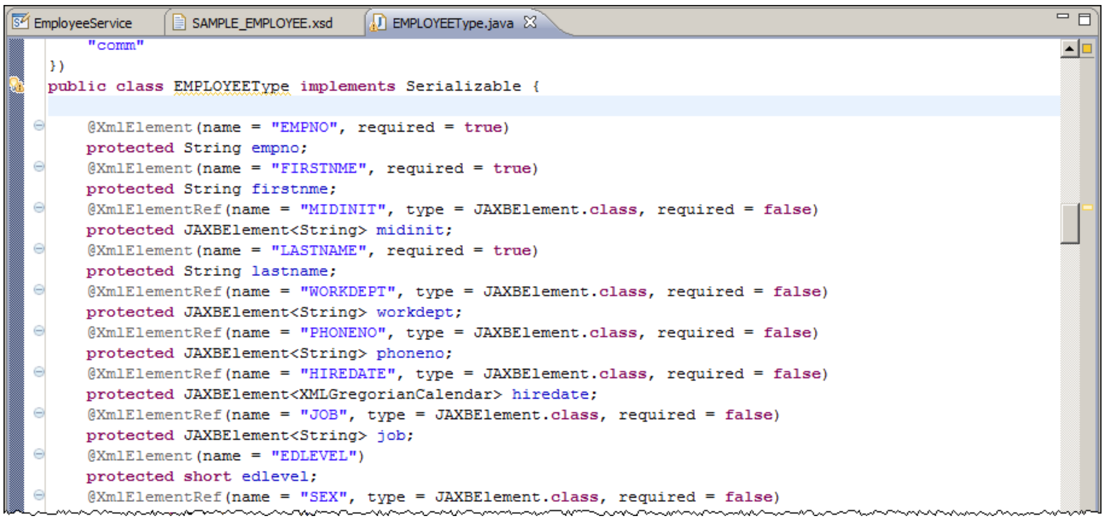


```
// This file was generated by the JavaTM Architecture for XML Binding(JAXB) Reference Implementation, v2.


package com.ibm.betaworks;

import java.io.Serializable;
import java.math.BigDecimal;
import javax.xml.bind.JAXBElement;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlElementRef;
import javax.xml.bind.annotation.XmlType;
import javax.xml.datatype.XMLGregorianCalendar;
```

4. Take a few minutes to review the content of the EMPLOYEEType Java class.

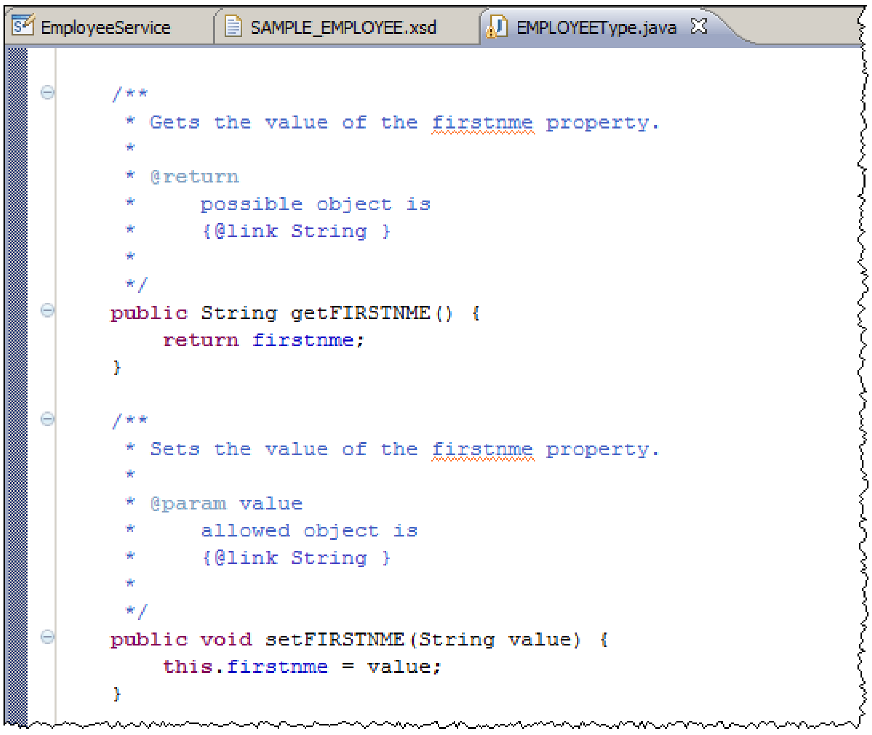
   Note that generating the JAXB classes using the Integration Toolkit has:

   a) defined every field in the database schema as a corresponding JAXB Java definition which matches the name of the database column name and definition:



```
    "comm"
})
public class EMPLOYEEType implements Serializable {

    @XmlElement(name = "EMPNO", required = true)
    protected String empno;
    @XmlElement(name = "FIRSTNME", required = true)
    protected String firstnme;
    @XmlElementRef(name = "MIDINIT", type = JAXBElement.class, required = false)
    protected JAXBElement<String> midinit;
    @XmlElement(name = "LASTNAME", required = true)
    protected String lastname;
    @XmlElementRef(name = "WORKDEPT", type = JAXBElement.class, required = false)
    protected JAXBElement<String> workdept;
    @XmlElementRef(name = "PHONENO", type = JAXBElement.class, required = false)
    protected JAXBElement<String> phoneno;
    @XmlElementRef(name = "HIREDATE", type = JAXBElement.class, required = false)
    protected JAXBElement<XMLGregorianCalendar> hiredate;
    @XmlElementRef(name = "JOB", type = JAXBElement.class, required = false)
    protected JAXBElement<String> job;
    @XmlElement(name = "EDLEVEL")
    protected short edlevel;
    @XmlElementRef(name = "SEX", type = JAXBElement.class, required = false)
```

5. b) a pair of “getters” and “setters” have been defined, one for each database field in the schema, the following shows the getter and setter for the FIRSTNME field in the schema:



```
/**
 * Gets the value of the firstnme property.
 *
 * @return
 *     possible object is
 *     {@link String }
 *
 */
public String getFIRSTNME() {
    return firstnme;
}

/**
 * Sets the value of the firstnme property.
 *
 * @param value
 *     allowed object is
 *     {@link String }
 *
 */
public void setFIRSTNME(String value) {
    this.firstnme = value;
}
```

These can be used to set or get the values for the variables.

6. Ensure nothing else is changed in the Java code. Save the file using <ctrl> S and close the EMPLOYEEType.java window.

7. Go back into the EMPLOYEEType.java code and ensure that your changes were saved correctly (*if this isn’t added to the code, then adding the complex type to the global cache will fail*).

5. Complete the getEmployeeCache operation

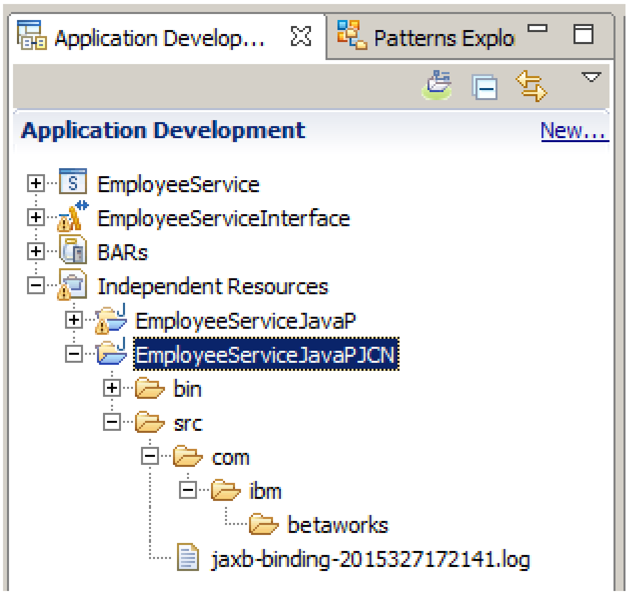
5.1 Add the Java code for getEmployeeCache

You will now add the Java code that will retrieve the employee data from the embedded cache.

1. Import the PI file called **`EmployeeServiceJavaPJCN.zip`** from **`c:\student10\integration_service_globalcache\resources\`**

   Note: This project will be used to contain a Java program called by a Java Compute Node (JCN). There is a separate library for this, because this particular Java program alone must be contained within the application, rather than just be referenced by it.

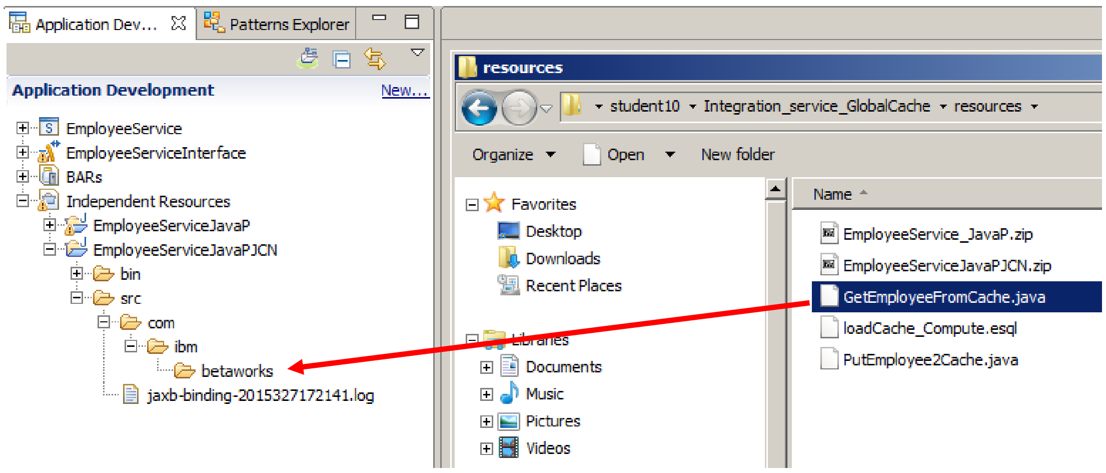
   The Java Project will appear in "Independent Resources". Note the project is created with com.ibm.betaworks defined as a Java package:



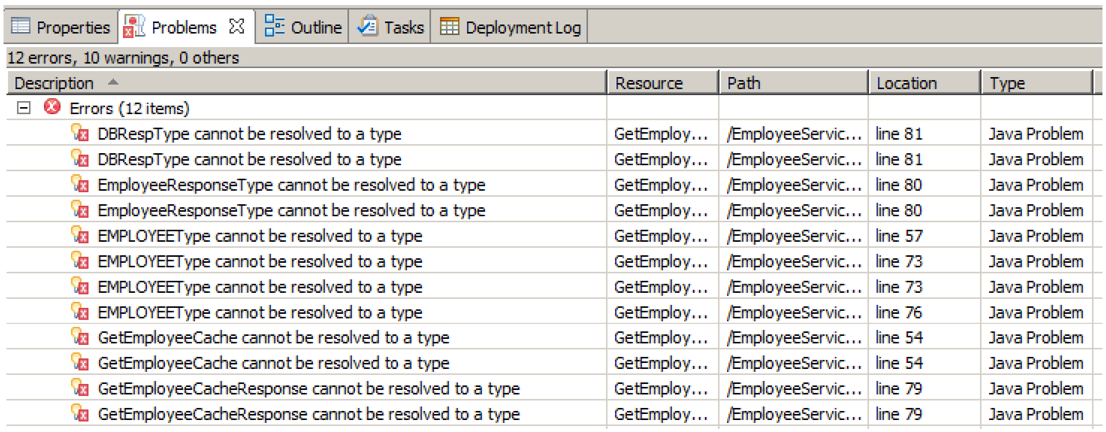
2. In Windows Explorer, navigate to

   `C:\student10\Integration_service_GlobalCache\resources`

   and drag and drop "GetEmployeeFromCache.java" onto "src.com.ibm.betaworks" in EmployeeServiceJavaPJCN:



3. Adding this code results in 12 errors being displayed in the Problems tab. This occurs because the new Java project (EmployeeServiceJavaPJCN) does not yet have access to the JAXB classes in Java project EmployeeServiceJavaP. This will be addressed a little later on.



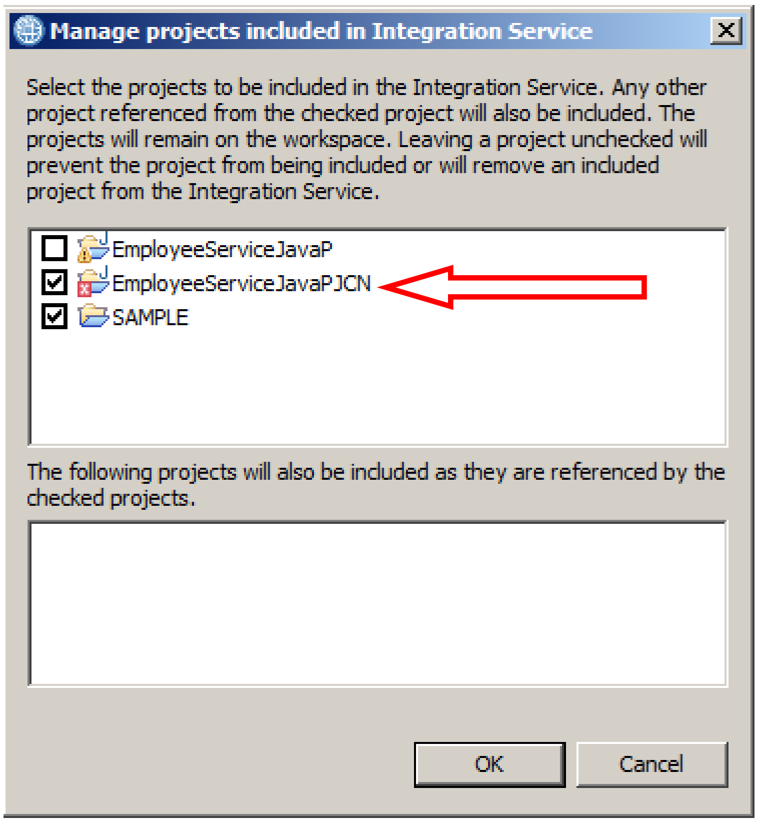
12 errors, 10 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (12 items)				
DBRespType cannot be resolved to a type	GetEmploy…	/EmployeeServic…	line 81	Java Problem
DBRespType cannot be resolved to a type	GetEmploy…	/EmployeeServic…	line 81	Java Problem
EmployeeResponseType cannot be resolved to a type	GetEmploy…	/EmployeeServic…	line 80	Java Problem
EmployeeResponseType cannot be resolved to a type	GetEmploy…	/EmployeeServic…	line 80	Java Problem
EMPLOYEEType cannot be resolved to a type	GetEmploy…	/EmployeeServic…	line 57	Java Problem
EMPLOYEEType cannot be resolved to a type	GetEmploy…	/EmployeeServic…	line 73	Java Problem
EMPLOYEEType cannot be resolved to a type	GetEmploy…	/EmployeeServic…	line 73	Java Problem
EMPLOYEEType cannot be resolved to a type	GetEmploy…	/EmployeeServic…	line 76	Java Problem
GetEmployeeCache cannot be resolved to a type	GetEmploy…	/EmployeeServic…	line 54	Java Problem
GetEmployeeCache cannot be resolved to a type	GetEmploy…	/EmployeeServic…	line 54	Java Problem
GetEmployeeCacheResponse cannot be resolved to a type	GetEmploy…	/EmployeeServic…	line 79	Java Problem
GetEmployeeCacheResponse cannot be resolved to a type	GetEmploy…	/EmployeeServic…	line 79	Java Problem

4. Right click on the EmployeeService in the Application Development window and click “Manage included projects”.

   Add the EmployeeServiceJavaPJCN project to the list of projects referenced by the service and click OK .

   Do **not** include EmployeeServiceJavaP. This contains the JAXB classes that must not be included in the bar file build.



Select the projects to be included in the Integration Service. Any other project referenced from the checked project will also be included. The projects will remain on the workspace. Leaving a project unchecked will prevent the project from being included or will remove an included project from the Integration Service.

- [ ] EmployeeServiceJavaP
- [x] EmployeeServiceJavaPJCN
- [x] SAMPLE

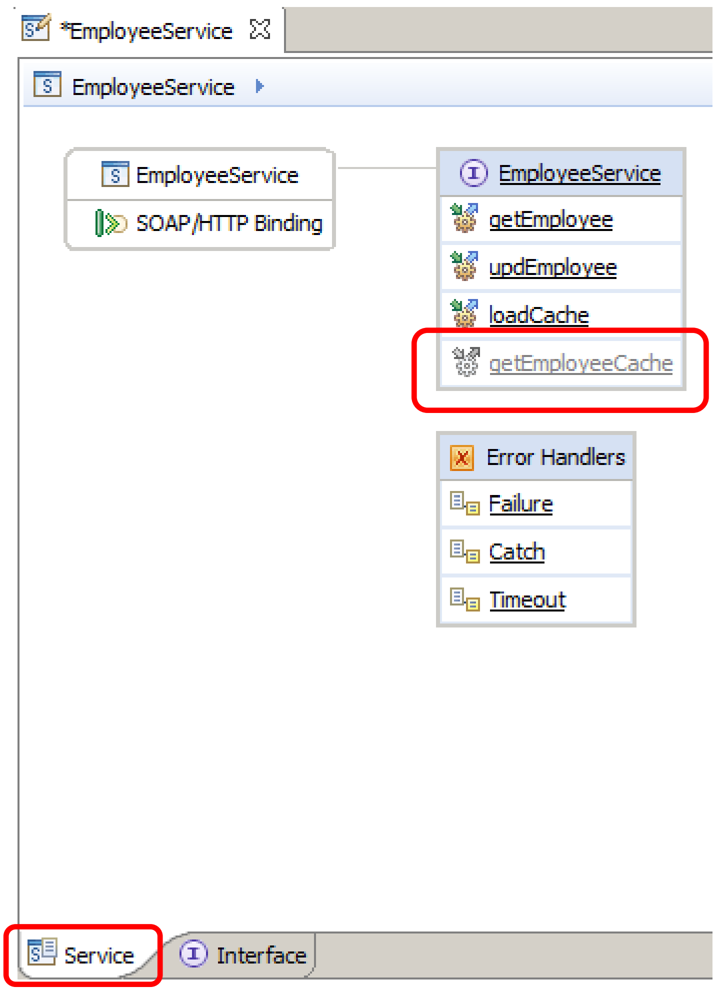
The following projects will also be included as they are referenced by the checked projects.

5.2 Add Java Compute Node to subflow

You will now complete the definition for GetEmployeeCache by adding a Java compute node to the sub flow associated with the operation.

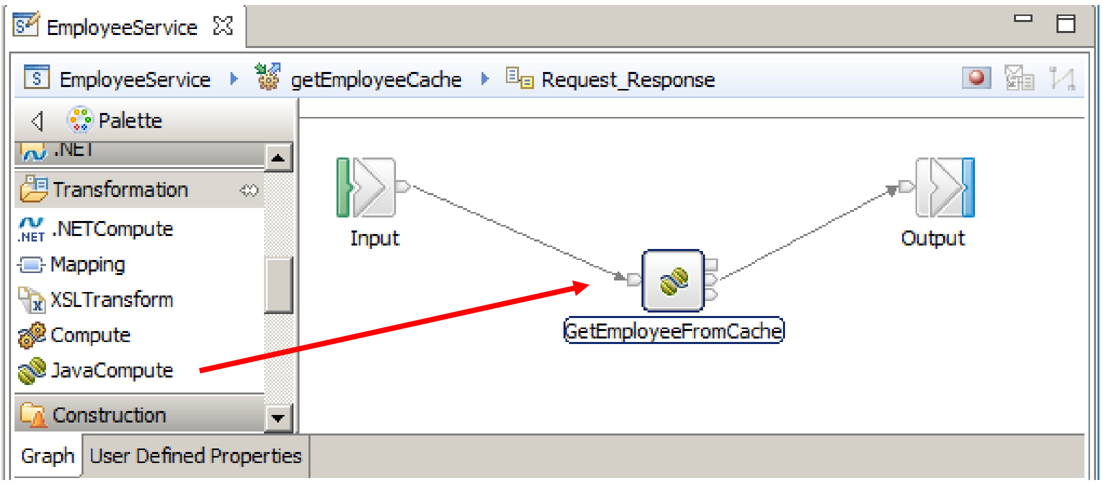
1. Back in the Integration Development perspective, in the EmployeeService definition, switch to the Service Tab (at side of the Interface tab).

   Click getEmployeeCache from the view of operations defined for the service (*getEmployeeCache will be greyed out as the subflow associated with that operation has not yet been defined*)



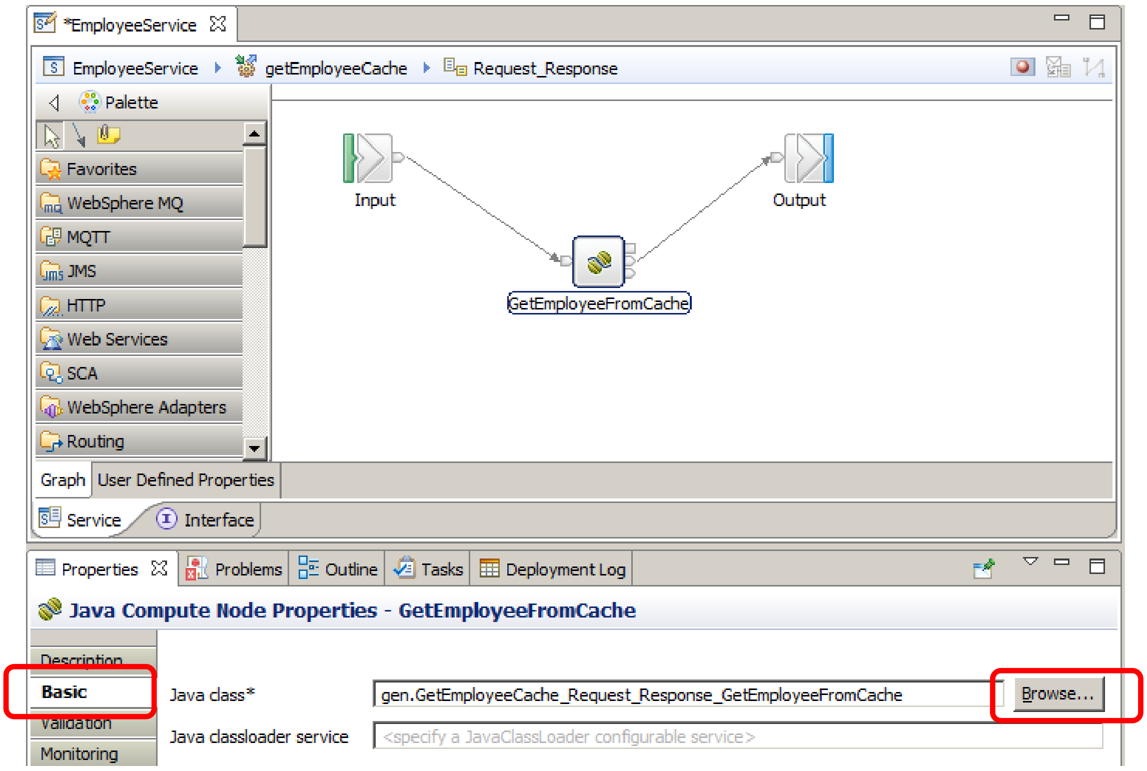
   The flow editor for the getEmployeeCache subflow will be shown with an Input and Output node predefined.

2. Add a Java Compute node called "**`GetEmployeeFromCache`**" to the canvas and join the Input and Output labels through this node.



3. Do not open the Java compute node yet. Highlight the GetEmployeeFromCache Java compute node by clicking it once so that the properties tab shows the details for the node.

   Ensure the Basic tab is open in the lower pane and click the "browse" button next to the "Java class*" field.

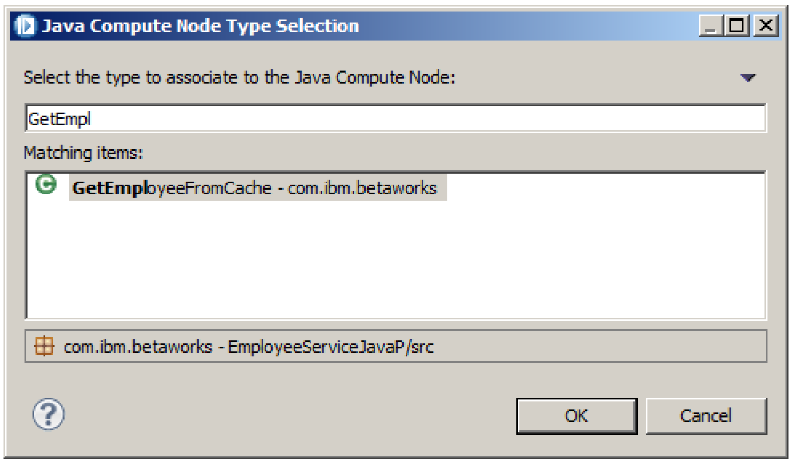


Java Compute Node Properties - GetEmployeeFromCache

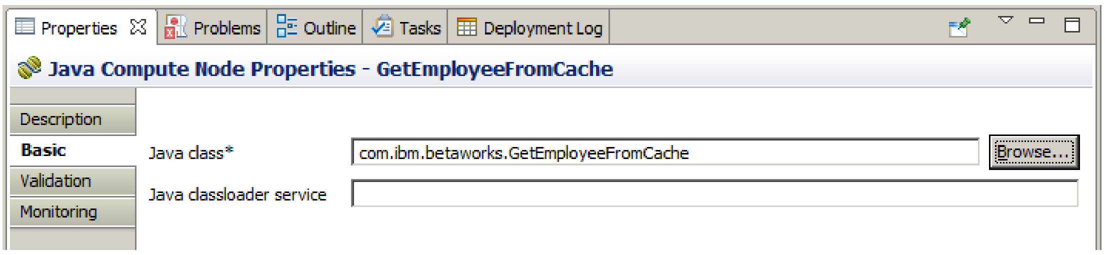
Java class*	gen.GetEmployeeCache_Request_Response_GetEmployeeFromCache
Java classloader service	<specify a JavaClassLoader configurable service>

4. In the “Java Compute Node Type selection” window, type the first few letters of the class name (eg “`GetEmpl`”).

   Click the matching class (this should be “**`com.ibm.betaworks.GetEmployeeFromCache`**” that you added earlier and click OK:


Java Compute Node Type Selection
Select the type to associate to the Java Compute Node:
GetEmpl
Matching items:
GetEmployeeFromCache - com.ibm.betaworks
com.ibm.betaworks - EmployeeServiceJavaP/src
OK
Cancel

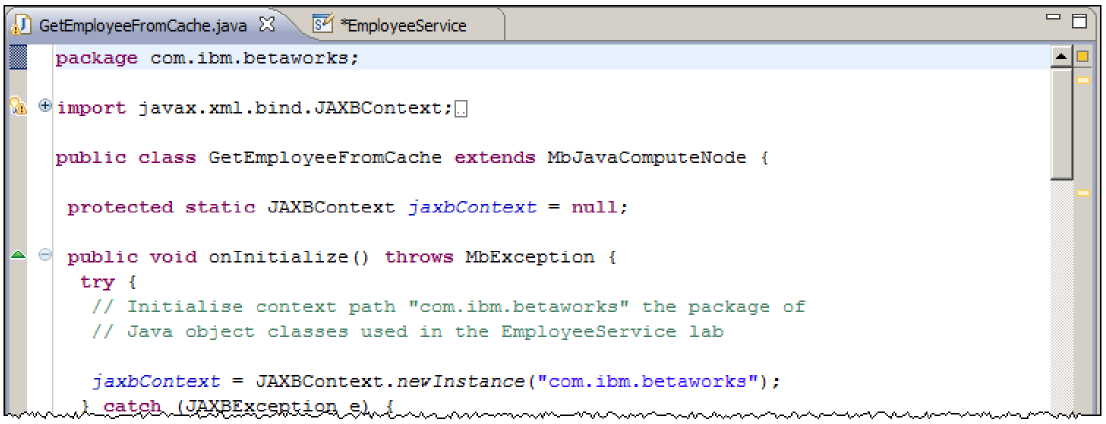
5. The resulting Basic property should look like this:



Java class* com.ibm.betaworks.GetEmployeeFromCache

Java classloader service

6. Double click on the GetEmployeeFromCache node and check the Java code should look like this. *(If not, you will need to retrace your steps).*



```
package com.ibm.betaworks;

import javax.xml.bind.JAXBContext;

public class GetEmployeeFromCache extends MbJavaComputeNode {

 protected static JAXBContext jaxbContext = null;

 public void onInitialize() throws MbException {
  try {
   // Initialise context path "com.ibm.betaworks" the package of
   // Java object classes used in the EmployeeService lab

   jaxbContext = JAXBContext.newInstance("com.ibm.betaworks");
  } catch (JAXBException e) {
```

   Close the GetEmployeeFromCache.java window.

7. Save the EmployeeService (ctrl s).

6. Add Java classes to Shared Classes library

IIB access to the Global Cache using JAXB classes mandates that the classes are defined in the Shared Classes library for the Integration Node where you intend to run the Service. **These classes must not exist within the application that is deployed to the Integration node**. To make things simple we will add all Java classes used throughout the global cache operations to the Shared Classes library. By contrast, **the Java program called by the JCN must be directly included within the deployed application,** which is why there is a separate Java project for this.

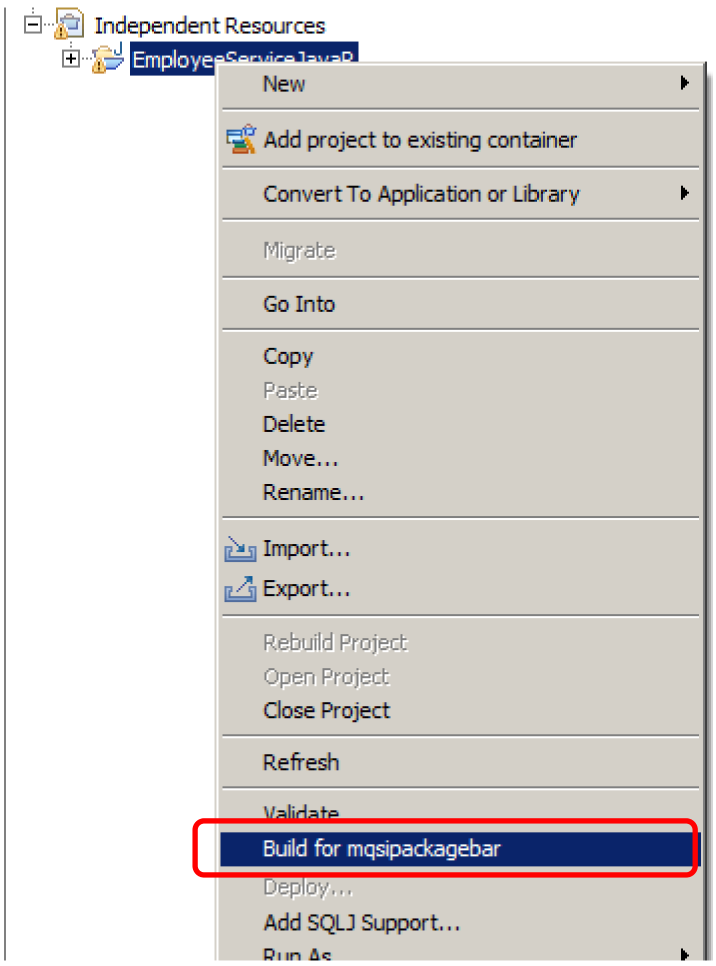
In the following section you will prepare the environment to enable this to happen correctly. You will:

1) build the jar file to be used in the Shared Classes folder;
2) update the Java build path for the Java project containing the code used by the JCN

6.1 Build a jar file and add it to Shared Classes folder

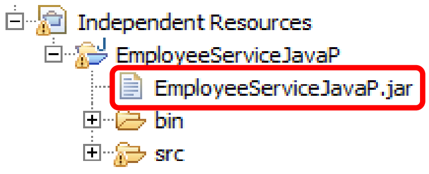
The first task is to create a jar file that you will add to the shared classes folder.

1. Right click on the EmployeeServiceJavaP project in the “Independent Resources” section of the Application Development window and choose “Build for mqsipackagebar”:

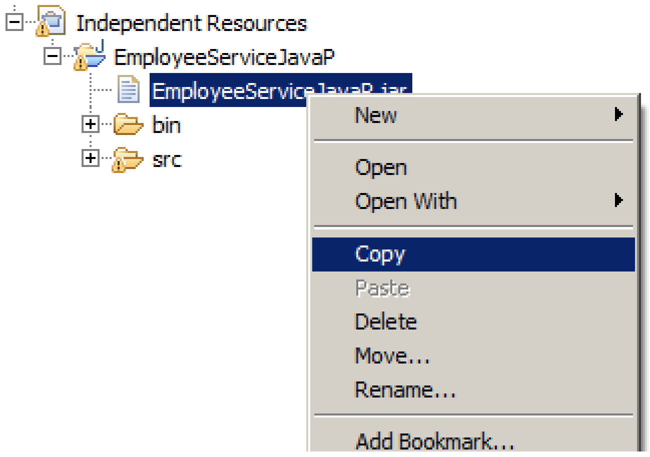


If the build is successful you will see a Build Information window, click ok to respond to it.

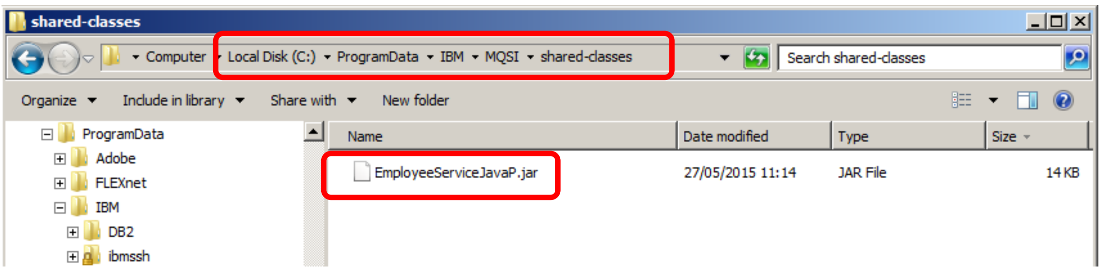
2. The jar file is now available in the workspace under the name of the project.



3. Right click on EmployeeServiceJavaP.jar and click **copy:**



4. Using Windows Explorer, navigate to the IIB **shared-classes** folder (*If you are using the supplied VMware environment for this lab guide, the Library is in "`C:\ProgramData\IBM\MQSI\shared-classes`"*).



Name	Date modified	Type	Size
EmployeeServiceJavaP.jar	27/05/2015 11:14	JAR File	14 KB

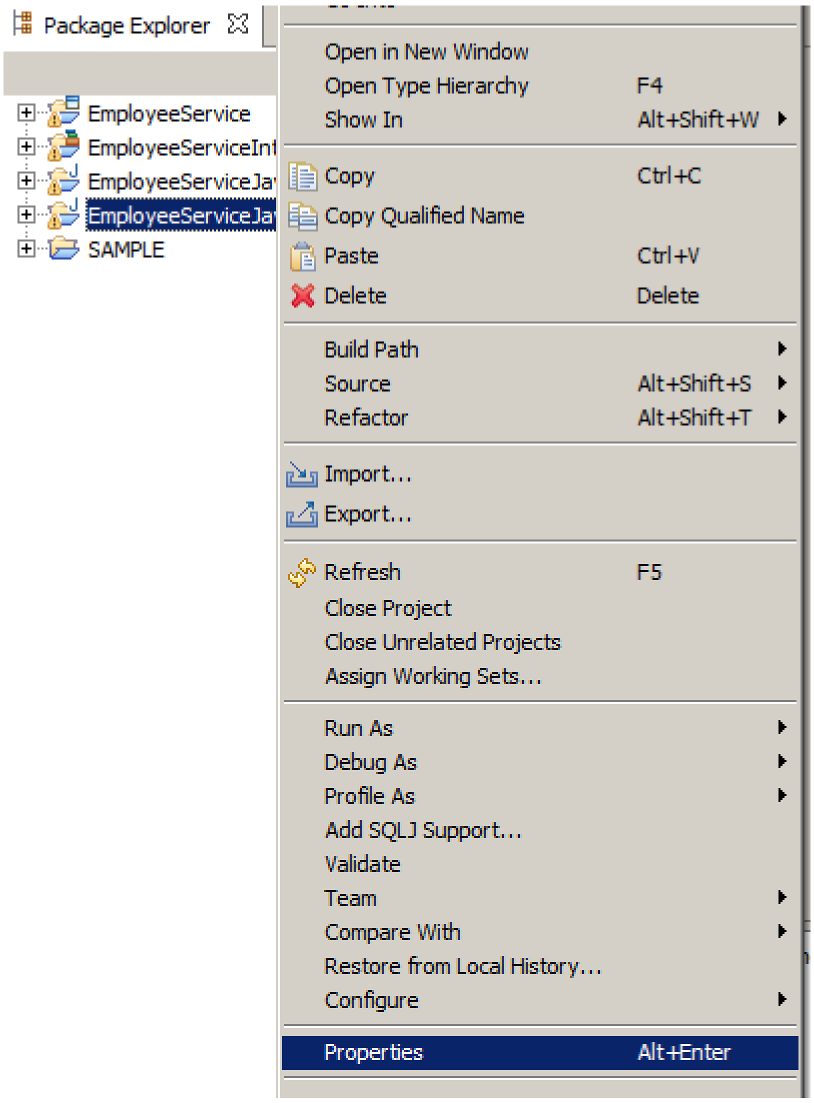
Paste a copy of the jar file **EmplpoyeeServiceJavaP.jar** into the **shared-classes** folder. (*If the file already exists, replace it.*)

5. Stop and restart IB10NODE for the jar file to be picked up by the Integration Node.

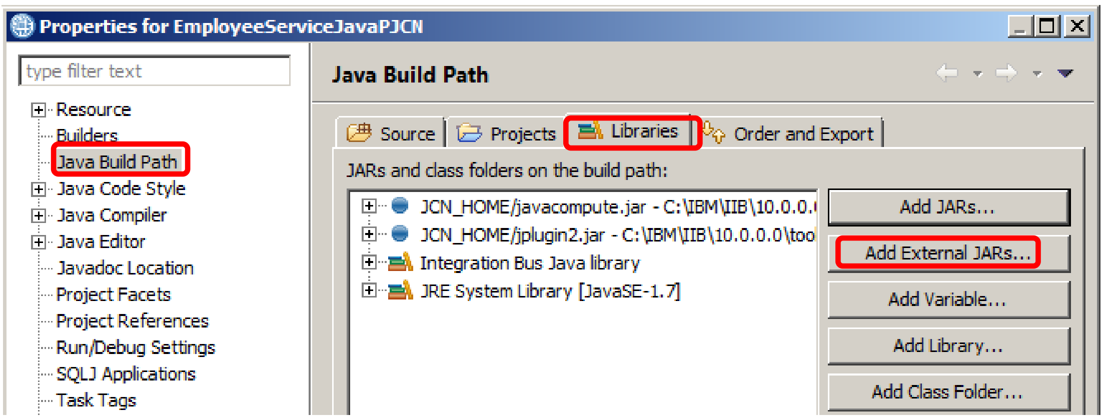
6.2 Update Java Build Path for Java Project containing JCN code

You will recall there are 12 errors in the Java code **GetEmployeeFromCache.java** called from the Java Compute Node, because the program contains references to JAXB Java classes that are not in the Java Project EmployeeServiceJavaPJCN. Thereafore, the new EmployeeServiceJavaP.jar file must be added to the libraries in the Java Build Path for EmployeeServiceJavaPJCN.

1. Open the Java perspective
2. In Package Explorer, right-click on EmployeeServiceJavaPJCN and select Properties.



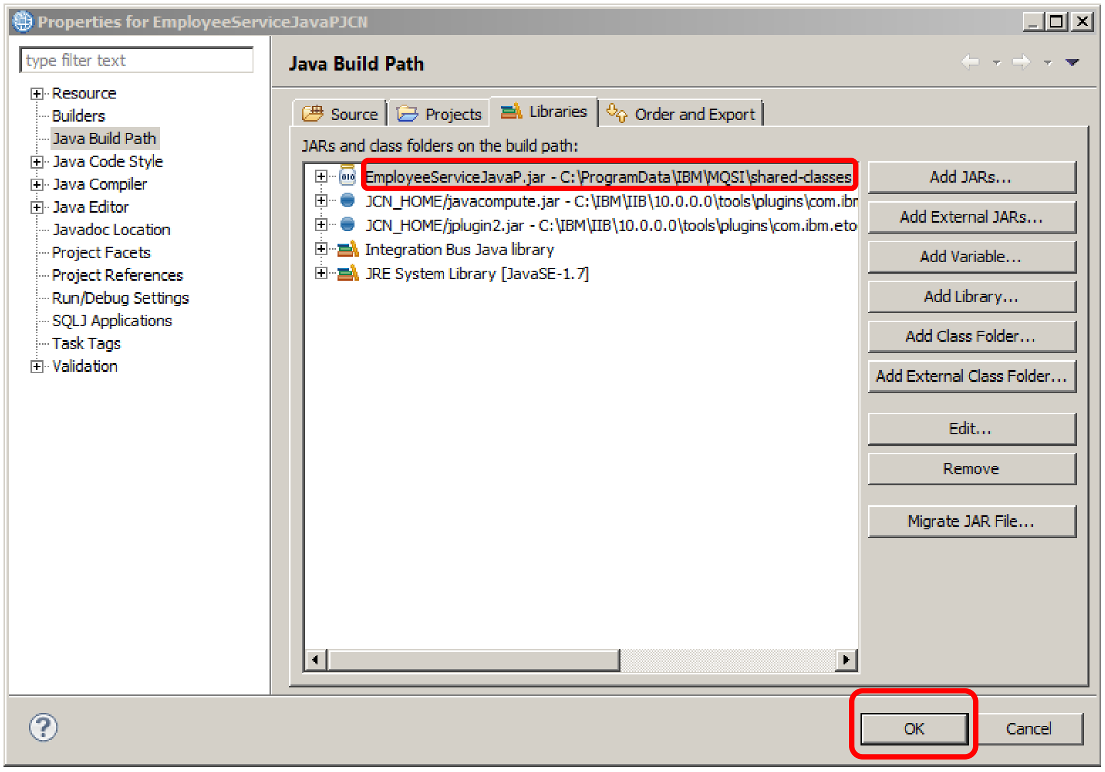
3. Select Java Build Path at left, then the Libraries tab and click on Add External JARs.



JARs and class folders on the build path:

JCN_HOME/javacompute.jar - C:\IBM\IIB\10.0.0.
JCN_HOME/jplugin2.jar - C:\IBM\IIB\10.0.0.0\too
Integration Bus Java library
JRE System Library [JavaSE-1.7]

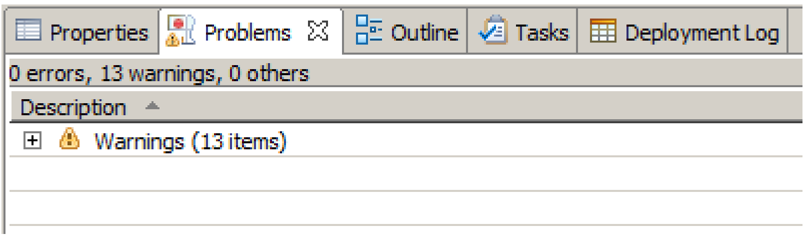
4. Navigate to `C:\ProgramData\IBM\MQSI\shared-classes` and select EmployeeServiceJavaP.jar. After the file has been selected, confirm it is in the list, as shown and then click OK.



JARs and class folders on the build path:

EmployeeServiceJavaP.jar - C:\ProgramData\IBM\MQSI\shared-classes
JCN_HOME/javacompute.jar - C:\IBM\IIB\10.0.0.0\tools\plugins\com.ibr
JCN_HOME/jplugin2.jar - C:\IBM\IIB\10.0.0.0\tools\plugins\com.ibm.eto
Integration Bus Java library
JRE System Library [JavaSE-1.7]

5. Close the Java perspective. Now that the Java Build Path has been updated, any errors arising from the GetEmployeeFromCache.java code that was added to this Java project, should now be gone from the Problems tab.



0 errors, 13 warnings, 0 others

Warnings (13 items)

7. Configure the embedded Global Cache

The global cache is disabled by default on an integration node. In the next section you will enable the embedded IIB Global Cache.

1. From an Integration Console enter the following IIB reportproperties command to establish the current status of the Global Cache in your environment:

   ```
   mqsireportproperties IB10NODE -b cachemanager -o CacheManager -r
   ```

   The command will respond similar to the following:

   ```
   CacheManager
     uuid='CacheManager'
     policy='disabled'
     portRange='2820-2839'
     listenerHost=''
     shutdownMode='fast'

   BIP8071I: Successful command completion.
   ```

   `Policy='disabled'` means the Global Cache is disabled and can not be used. The following steps will enable the Global Cache.

2. Stop the IB10NODE

   (use the Integration Toolkit or type **mqsistop IB10NODE** in an Integration Console)

3. In an Integration Console type the following command:

   ```
   mqsichangebroker IB10NODE –b default
   ```

   This command enables the default configuration for the IIB Global Cache on IB10NODE.

   The expected response is

   ```
   BIP8071I: Successful command completion.
   ```

4. Start the IB10NODE and ensure the node comes up correctly.

   (use the Integration Toolkit or type **mqsistart IB10NODE** in an Integration Console)

5. Run the following command again to confirm that the CacheManager policy is now set to default:

   ```
   mqsireportproperties IB10NODE -b cachemanager -o CacheManager -r
   ```

6. Now enter the following command to check the contents of the Global Cache:

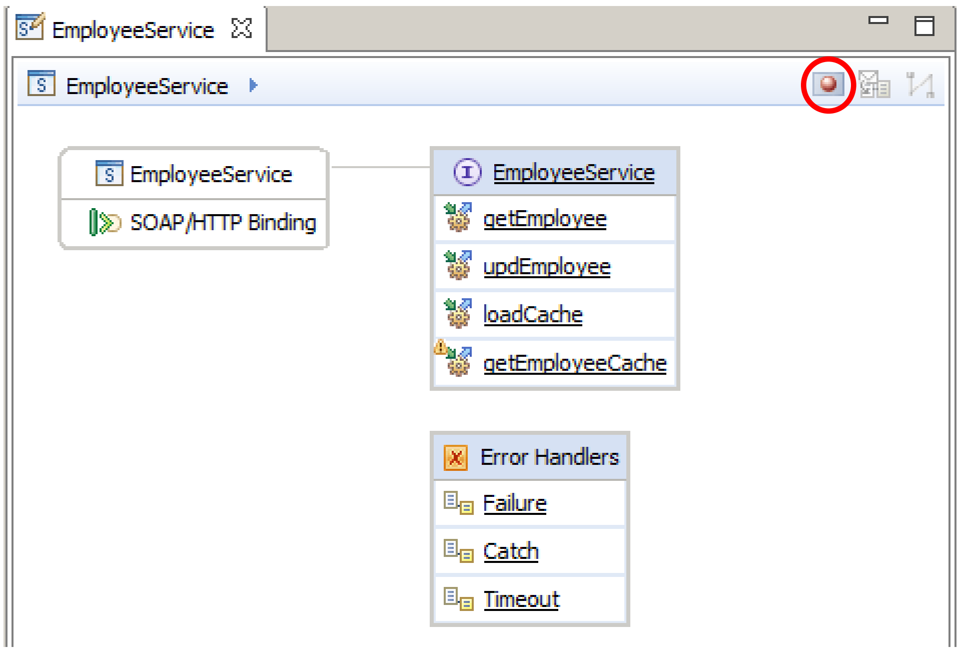
   ```
   mqsicacheadmin IB10NODE -c showMapSizes
   ```

7. Output similar to the attached will appear in the Integration Console, (*the output indicates that no Maps are currently available*):

```
C:\IBM\IIB\10.0.0.0>mqsicacheadmin IB10NODE -c showMapSizes
BIP7187I: Output from the mqsicacheadmin command. The output from
the WebSphere
eXtreme Scale xscmd utility is '
Starting at: 2015-05-19 11:49:46.377

CWXSI0068I: Executing command: showMapSizes

*** Displaying results for WMB data grid and mapSet map set.

*** Listing maps for IB10NODE_192.168.126.169_2800 ***
Map Name                    Partition Map Entries Used Bytes Shard
Type Containe
r
--------                    --------- ----------- ---------- -------
--- --------
-
SYSTEM.BROKER.CACHE.CLIENTS 0         1           640 B      Primary
IB10NODE
_192.168.126.169_2800_C-0
SYSTEM.BROKER.CACHE.SERVERS 0         1           656 B      Primary
IB10NODE
_192.168.126.169_2800_C-0
Server total: 2 (1 KB)

Total catalog service domain count: 2 (1 KB)
(The used bytes statistics are accurate only when you are using
simple objects o
r the COPY_TO_BYTES copy mode.)

CWXSI0040I: The showMapSizes command completed successfully.

Ending at: 2015-05-19 11:49:48.752
'
BIP8071I: Successful command completion.
```

8. Test the new operations using the Flow Exerciser

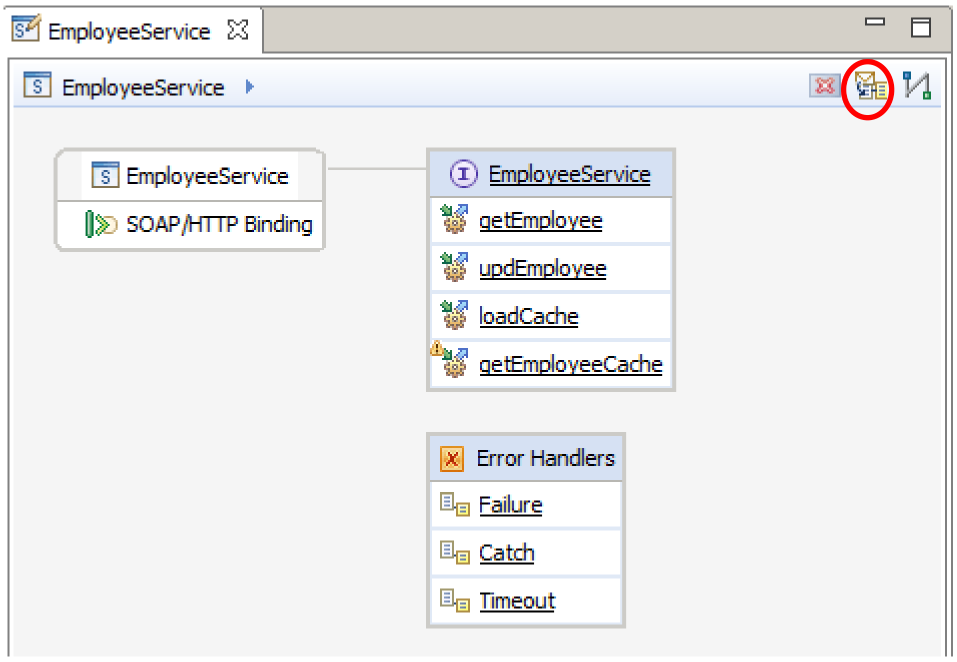
1. Click on the EmployeeService Integration Service Description, then click the red button to start the flow exerciser.



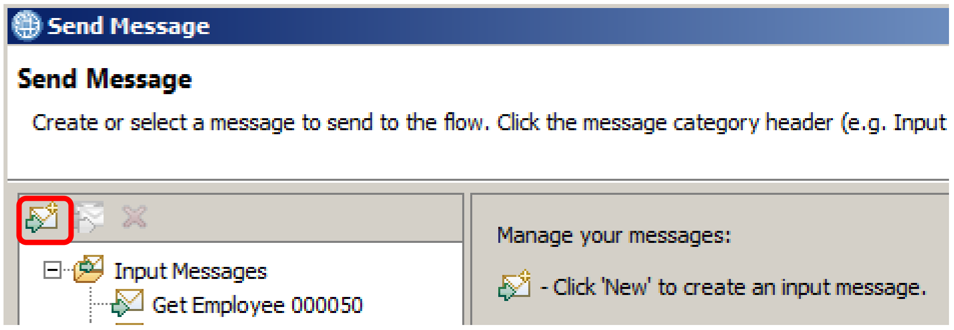
2. When the service has been deployed, a message box will appear to show that the integration server is ready to record messages. Click on the **Close** button to continue.

8.1 Test the loadCache operation

1. Click the **send message** icon



2. Click on **new message**


Send Message

Send Message

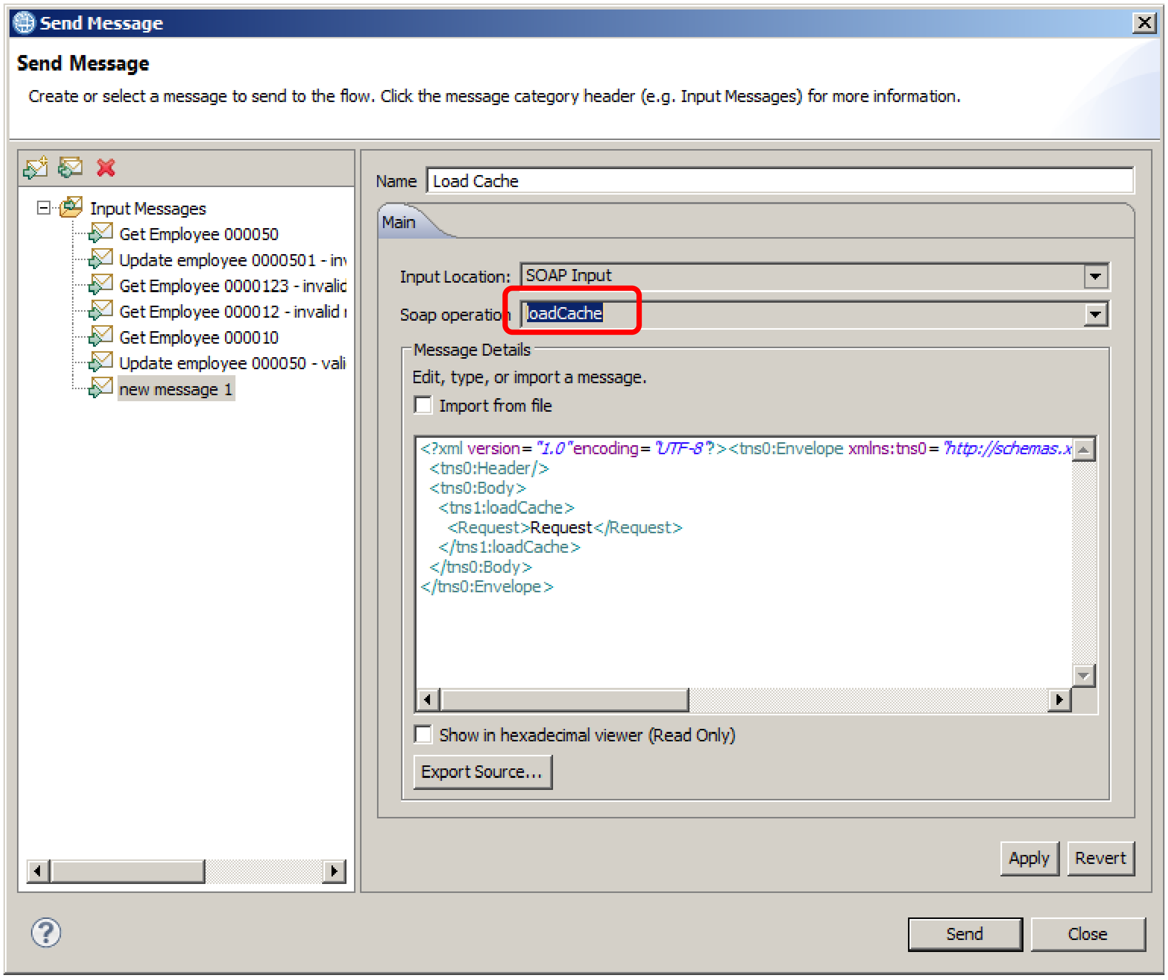
Create or select a message to send to the flow. Click the message category header (e.g. Input

Manage your messages:

- Click 'New' to create an input message.

3. Give the message a meaningful name, e.g. “Load Cache”.

   Select **loadCache** under **Soap operation** and click **Send**. This will send a SOAP message to the loadCache operation.



Send Message

Create or select a message to send to the flow. Click the message category header (e.g. Input Messages) for more information.

Input Messages
- Get Employee 000050
- Update employee 0000501 - inv
- Get Employee 0000123 - invalic
- Get Employee 000012 - invalid i
- Get Employee 000010
- Update employee 000050 - vali
- new message 1

Name: Load Cache

Input Location: SOAP Input

Soap operation: loadCache

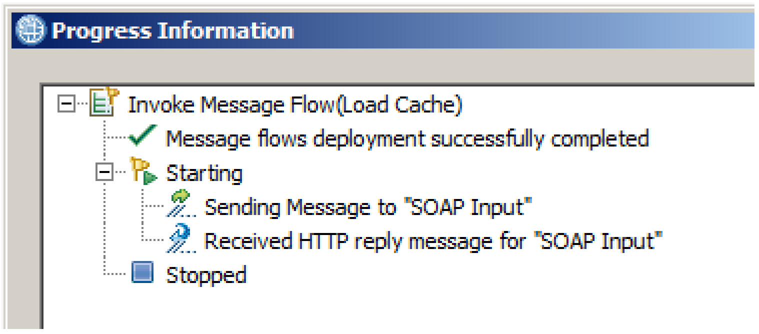
Message Details

Edit, type, or import a message.

```
<?xml version="1.0"encoding="UTF-8"?><tns0:Envelope xmlns:tns0="http://schemas.x
  <tns0:Header/>
  <tns0:Body>
    <tns1:loadCache>
      <Request>Request</Request>
    </tns1:loadCache>
  </tns0:Body>
</tns0:Envelope>
```

4. After a few seconds, an HTTP reply message should be received.

   Close the window when you are ready.



Progress Information

- Invoke Message Flow(Load Cache)
  - Message flows deployment successfully completed
  - Starting
    - Sending Message to "SOAP Input"
    - Received HTTP reply message for "SOAP Input"
  - Stopped

5. In the Integration Console, re-enter the "mqsicacheadmin" command:

   mqsicacheadmin *IB10NODE* -c showMapSizes

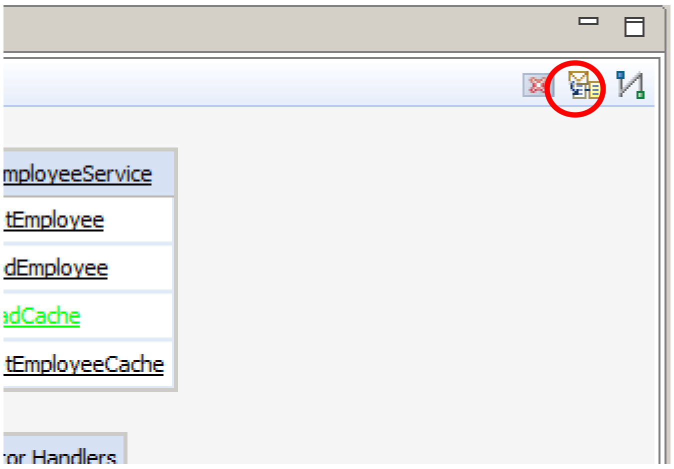
6. Output similar to the attached will be shown if the "loadCache" operation was successful.

   Map entries spread across multiple partitions exist within the map called "**aliveUntilRestart**":

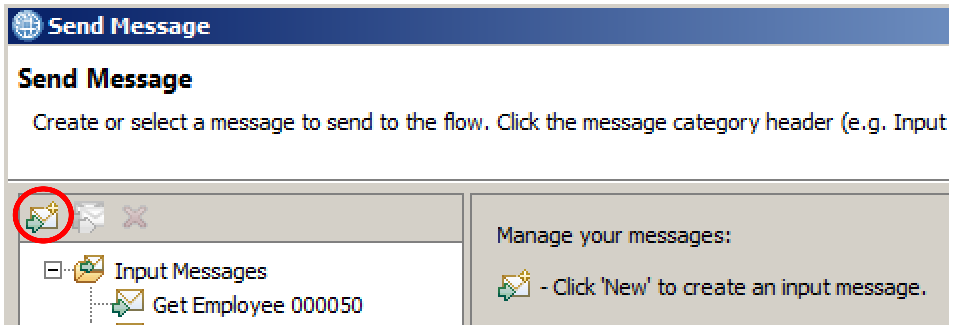
```
C:\IBM\IIB\10.0.0.0>mqsicacheadmin IB10NODE -c showMapSizes
BIP7187I: Output from the mqsicacheadmin command. The output from the WebSphere
eXtreme Scale xscmd utility is '
Starting at: 2015-05-19 11:53:40.458

CWXSI0068I: Executing command: showMapSizes

*** Displaying results for WMB data grid and mapSet map set.

*** Listing maps for IB10NODE_192.168.126.169_2800 ***
Map Name                    Partition Map Entries Used Bytes Shard Type Containe
r
--------                    --------- ----------- ---------- ---------- --------
-
SYSTEM.BROKER.CACHE.CLIENTS 0         1           640 B      Primary    IB10NODE
_192.168.126.169_2800_C-0
SYSTEM.BROKER.CACHE.SERVERS 0         1           656 B      Primary    IB10NODE
_192.168.126.169_2800_C-0
aliveUntilRestart           0         2           1 KB       Primary    IB10NODE
_192.168.126.169_2800_C-0
aliveUntilRestart           1         5           2 KB       Primary    IB10NODE
_192.168.126.169_2800_C-0
aliveUntilRestart           2         2           1 KB       Primary    IB10NODE
_192.168.126.169_2800_C-0
aliveUntilRestart           3         2           1 KB       Primary    IB10NODE
_192.168.126.169_2800_C-0
aliveUntilRestart           4         5           2 KB       Primary    IB10NODE
_192.168.126.169_2800_C-0
aliveUntilRestart           5         3           1 KB       Primary    IB10NODE
_192.168.126.169_2800_C-0
aliveUntilRestart           6         4           2 KB       Primary    IB10NODE
_192.168.126.169_2800_C-0
aliveUntilRestart           7         5           2 KB       Primary    IB10NODE
_192.168.126.169_2800_C-0
aliveUntilRestart           8         3           1 KB       Primary    IB10NODE
_192.168.126.169_2800_C-0
aliveUntilRestart           9         2           1 KB       Primary    IB10NODE
_192.168.126.169_2800_C-0
aliveUntilRestart           10        2           1 KB       Primary    IB10NODE
_192.168.126.169_2800_C-0
aliveUntilRestart           11        4           2 KB       Primary    IB10NODE
_192.168.126.169_2800_C-0
aliveUntilRestart           12        3           1 KB       Primary    IB10NODE
_192.168.126.169_2800_C-0
Server total: 44 (22 KB)

Total catalog service domain count: 44 (22 KB)
(The used bytes statistics are accurate only when you are using simple objects o
r the COPY_TO_BYTES copy mode.)

CWXSI0040I: The showMapSizes command completed successfully.

Ending at: 2015-05-19 11:53:42.474
'
BIP8071I: Successful command completion.
```

8.2 Test the getEmployeeCache operation – valid key

1. Click the **send message** icon again.



2. Click on **new message** again.


Send Message

Send Message

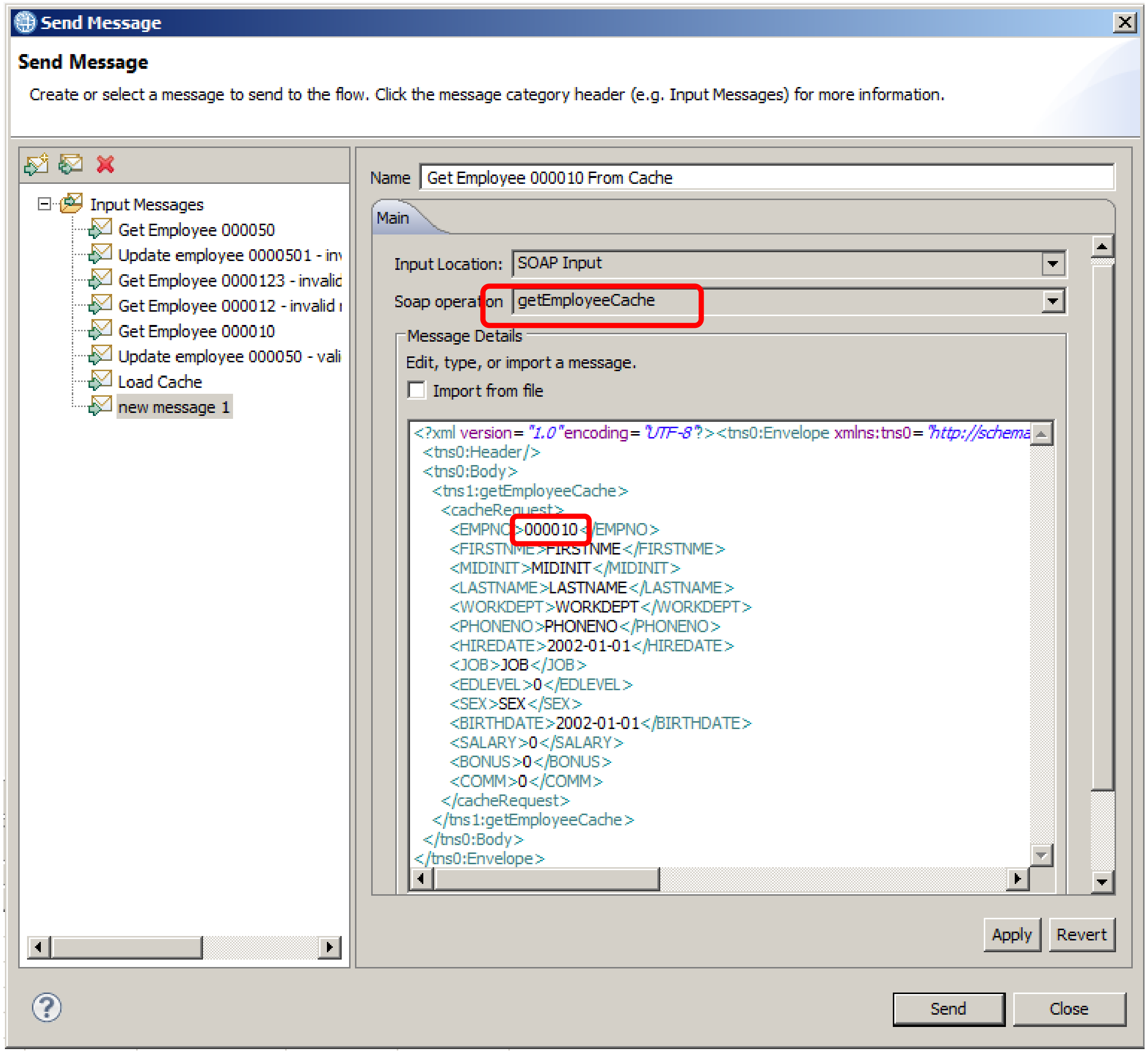
Create or select a message to send to the flow. Click the message category header (e.g. Input

Manage your messages:

- Click 'New' to create an input message.

3. Give the message a meaningful name, e.g. “Get Employee 000010 From Cache”.

   Select **getEmployeeCache** under **Soap operation**, enter an employee number of ‘000010’ and then click **Send**. This will send a SOAP message to the getEmployeeCache operation.


Send Message
Send Message
Create or select a message to send to the flow. Click the message category header (e.g. Input Messages) for more information.
Input Messages
Get Employee 000050
Update employee 0000501 - inv
Get Employee 0000123 - invalic
Get Employee 000012 - invalid
Get Employee 000010
Update employee 000050 - vali
Load Cache
new message 1
Main
Message Details
Edit, type, or import a message.
Import from file
Apply
Revert
Send
Close

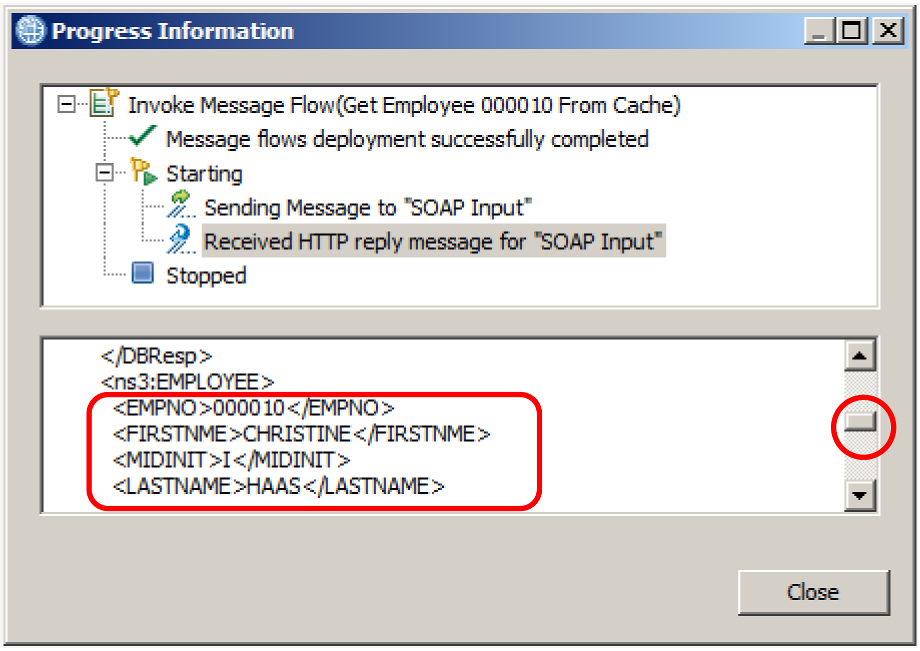
Name: Get Employee 000010 From Cache

Input Location: SOAP Input

Soap operation: getEmployeeCache

```
<?xml version="1.0" encoding="UTF-8"?><tns0:Envelope xmlns:tns0="http://schema
  <tns0:Header/>
  <tns0:Body>
    <tns1:getEmployeeCache>
      <cacheRequest>
        <EMPNO>000010</EMPNO>
        <FIRSTNME>FIRSTNME</FIRSTNME>
        <MIDINIT>MIDINIT</MIDINIT>
        <LASTNAME>LASTNAME</LASTNAME>
        <WORKDEPT>WORKDEPT</WORKDEPT>
        <PHONENO>PHONENO</PHONENO>
        <HIREDATE>2002-01-01</HIREDATE>
        <JOB>JOB</JOB>
        <EDLEVEL>0</EDLEVEL>
        <SEX>SEX</SEX>
        <BIRTHDATE>2002-01-01</BIRTHDATE>
        <SALARY>0</SALARY>
        <BONUS>0</BONUS>
        <COMM>0</COMM>
      </cacheRequest>
    </tns1:getEmployeeCache>
  </tns0:Body>
</tns0:Envelope>
```

4. Again, after a few seconds, an HTTP reply message should be received. Highlight and click on the notification. You should see that one row of the database has been returned. In the lower pane, scroll down to confirm the details are as expected. Close the window when you are ready.



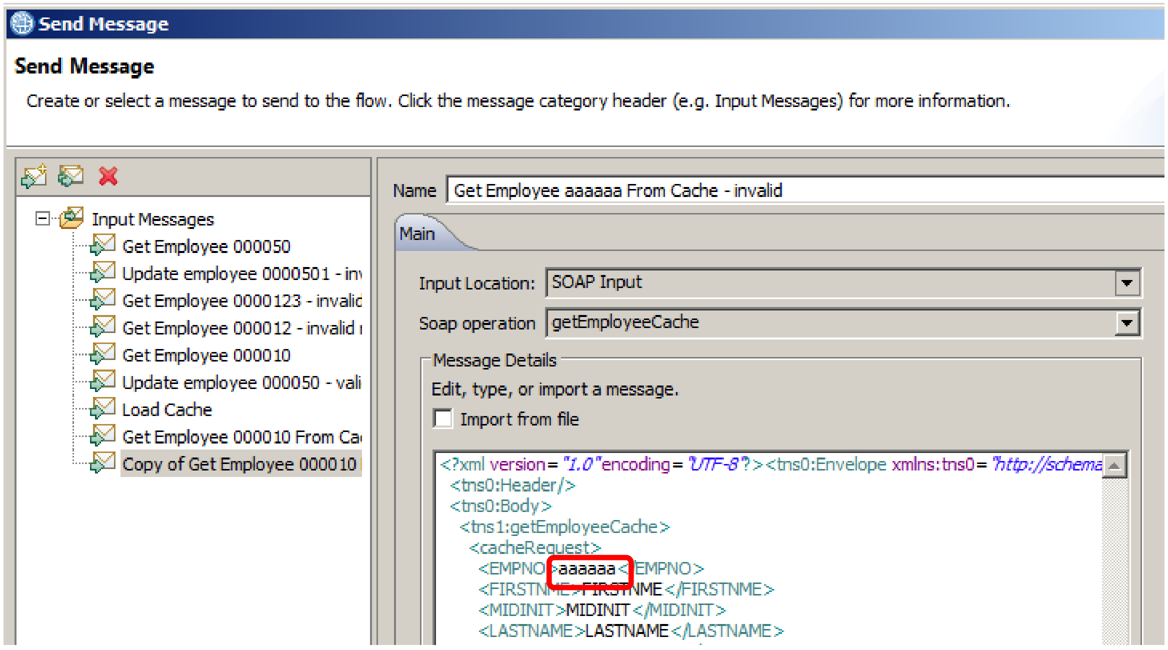
```
Invoke Message Flow(Get Employee 000010 From Cache)
    Message flows deployment successfully completed
    Starting
        Sending Message to "SOAP Input"
        Received HTTP reply message for "SOAP Input"
    Stopped
```

```
</DBResp>
<ns3:EMPLOYEE>
 <EMPNO>000010</EMPNO>
 <FIRSTNME>CHRISTINE</FIRSTNME>
 <MIDINIT>I</MIDINIT>
 <LASTNAME>HAAS</LASTNAME>
```

8.3 Test the getEmployeeCache operation – invalid key

1. Click the **send message** icon again.

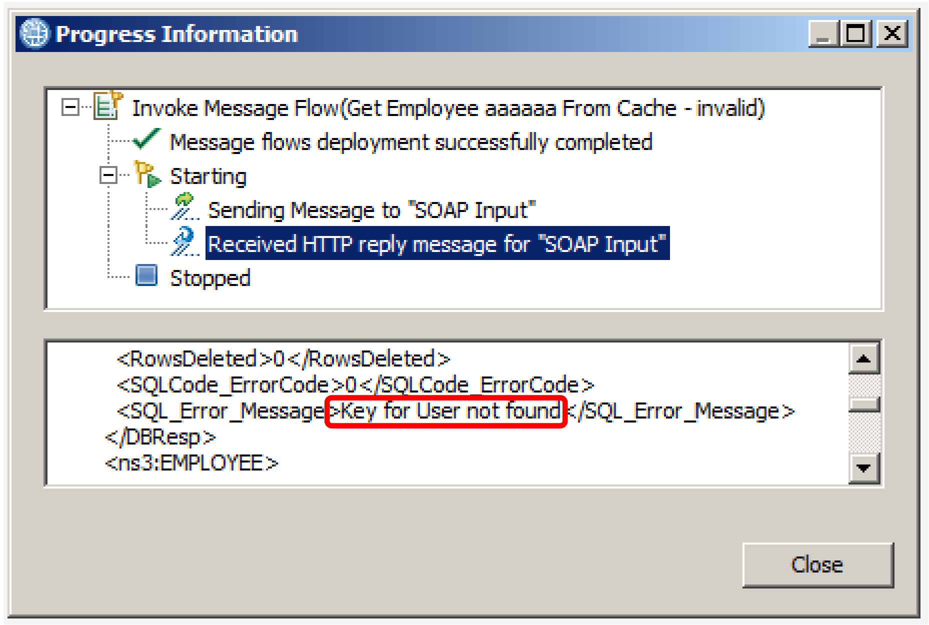
2. This time, right click and duplicate the previous message (for the getEmployeeCache operation), give it a suitable name and change the employee number to an invalid value, say ‘aaaaaa’ and click **Send**.


Send Message
Send Message
Create or select a message to send to the flow. Click the message category header (e.g. Input Messages) for more information.
Input Messages
Get Employee 000050
Update employee 0000501 - inv
Get Employee 0000123 - invalid
Get Employee 000012 - invalid
Get Employee 000010
Update employee 000050 - vali
Load Cache
Get Employee 000010 From Ca
Copy of Get Employee 000010
Main
Message Details
Edit, type, or import a message.
Import from file

Name: Get Employee aaaaaa From Cache - invalid
Input Location: SOAP Input
Soap operation: getEmployeeCache

```
<?xml version="1.0" encoding="UTF-8"?><tns0:Envelope xmlns:tns0="http://schema
 <tns0:Header/>
 <tns0:Body>
  <tns1:getEmployeeCache>
   <cacheRequest>
    <EMPNO>aaaaaa</EMPNO>
    <FIRSTNME>FIRSTNME</FIRSTNME>
    <MIDINIT>MIDINIT</MIDINIT>
    <LASTNAME>LASTNAME</LASTNAME>
```

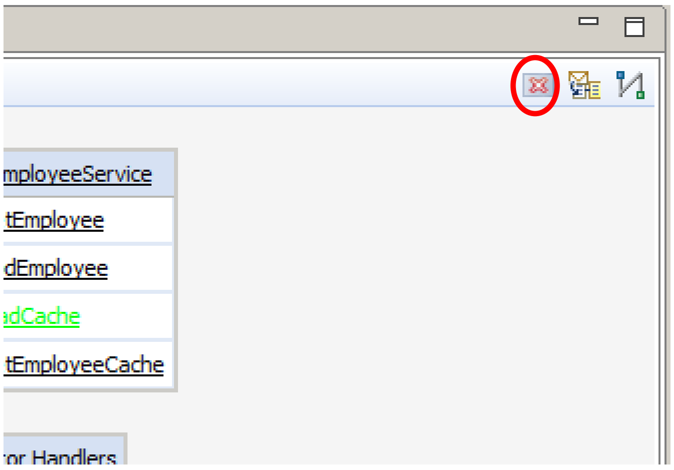
3. At the Progress Information window, highlight the reply and scroll down in the lower section to see an SQL error saying that the key was not found. Close the window when you are ready.



Invoke Message Flow(Get Employee aaaaaa From Cache - invalid)
Message flows deployment successfully completed
Starting
Sending Message to "SOAP Input"
Received HTTP reply message for "SOAP Input"
Stopped

```
  <RowsDeleted>0</RowsDeleted>
  <SQLCode_ErrorCode>0</SQLCode_ErrorCode>
  <SQL_Error_Message>Key for User not found</SQL_Error_Message>
 </DBResp>
 <ns3:EMPLOYEE>
```

4. You will be returned to the EmployeeService view. Return the flow to edit mode by clicking on the red icon.



5. A warning message will pop up. Click on **Yes** to continue.

6. Close the EmployeeService

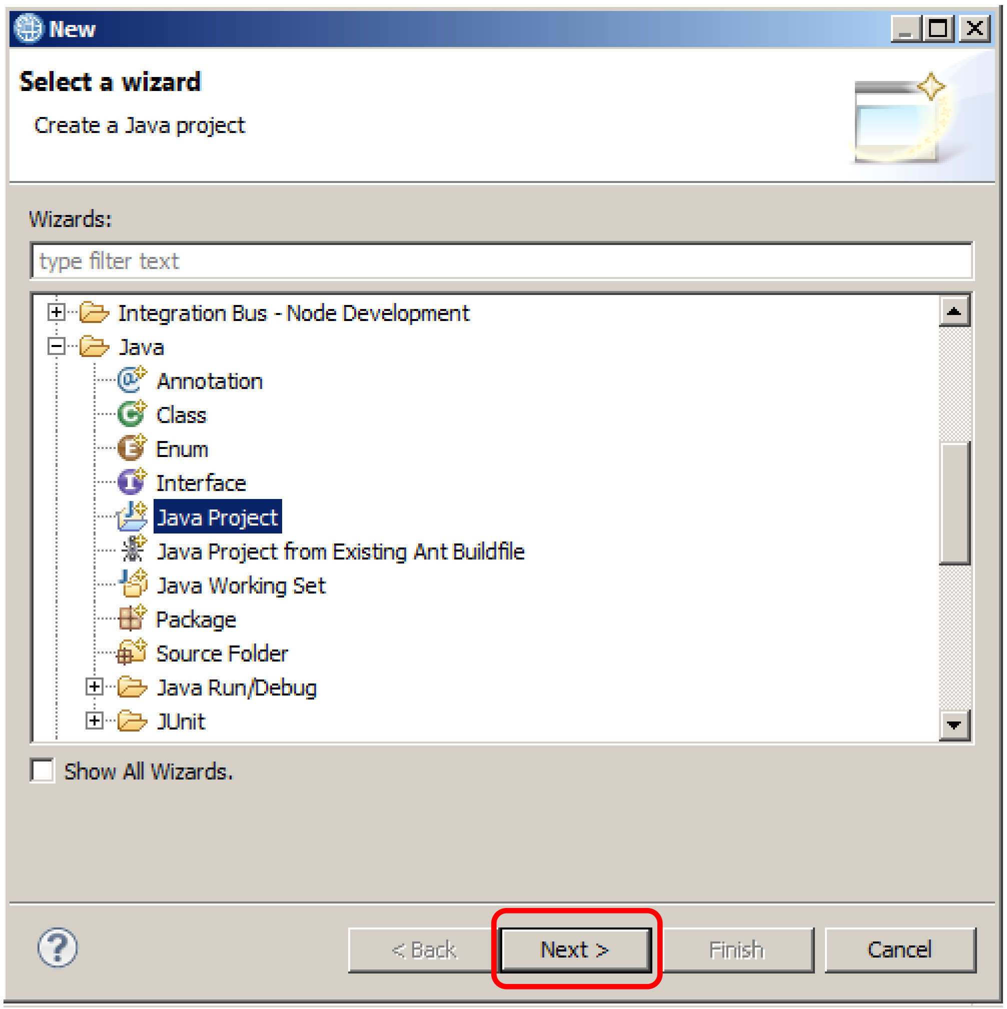
END OF LAB GUIDE

9. Appendix

9.1 Creating the Java Project for JAXB Java classes

If you want to create this Java Project (EmployeeServiceJavaP) from scratch, follow these instructions.

1. Right click on the “white space” of Application development background and click “New > Other”. In “Select a Wizard” dialog box expand Java and choose Java Project and click Next:



New

Select a wizard

Create a Java project

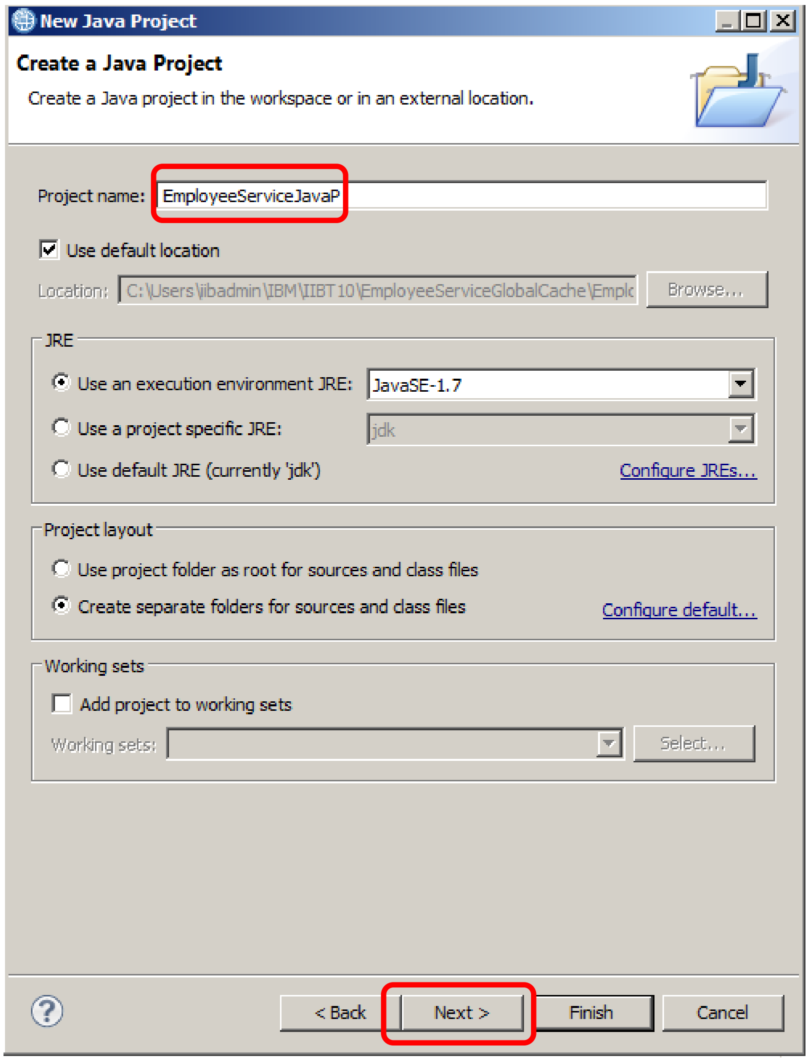
Wizards:

type filter text

- Integration Bus - Node Development
- Java
  - Annotation
  - Class
  - Enum
  - Interface
  - Java Project
  - Java Project from Existing Ant Buildfile
  - Java Working Set
  - Package
  - Source Folder
  - Java Run/Debug
  - JUnit

Show All Wizards.

2. Specify the Project name as “EmployeeServiceJavaP” and click next:


New Java Project
Create a Java Project
Create a Java project in the workspace or in an external location.

Project name: EmployeeServiceJavaP

[✓] Use default location

Location: C:\Users\iibadmin\IBM\IIBT10\EmployeeServiceGlobalCache\Emplo

JRE

- (●) Use an execution environment JRE: JavaSE-1.7
- ( ) Use a project specific JRE: jdk
- ( ) Use default JRE (currently 'jdk')

Project layout

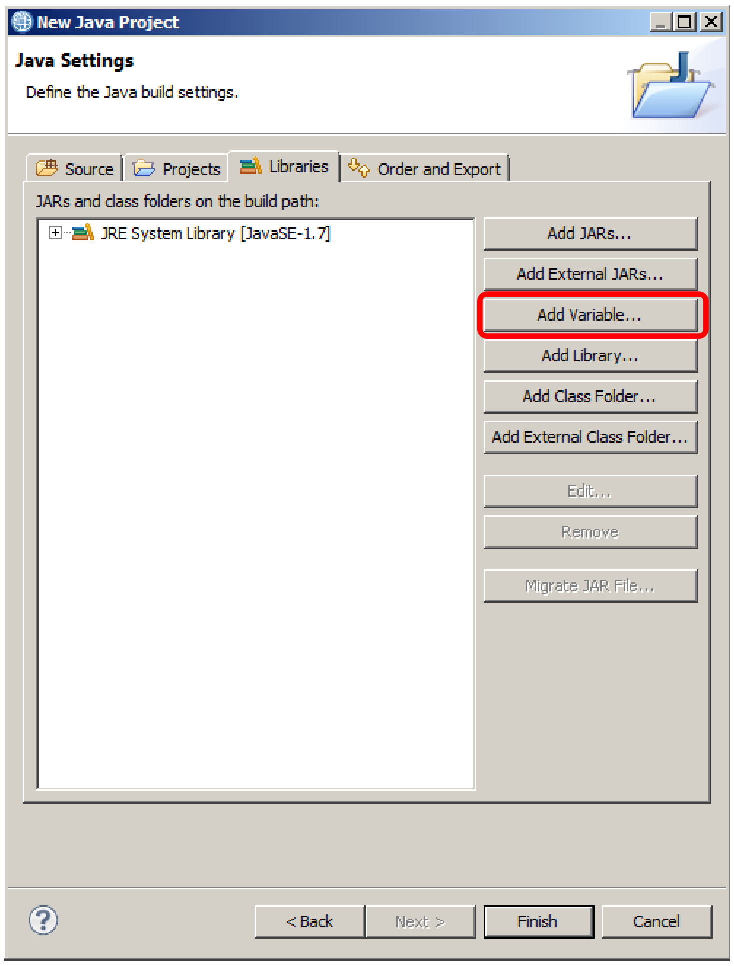
- ( ) Use project folder as root for sources and class files
- (●) Create separate folders for sources and class files

Working sets

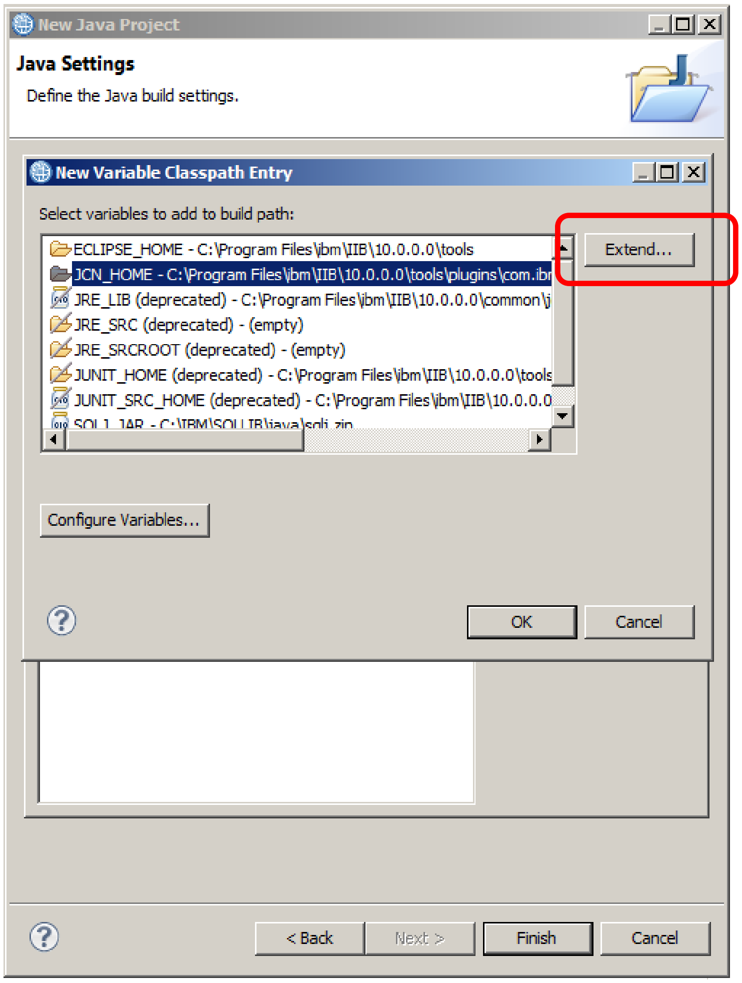
[ ] Add project to working sets

Working sets:

3. Choose the “Libraries” tab and click the “Add Variable” button:

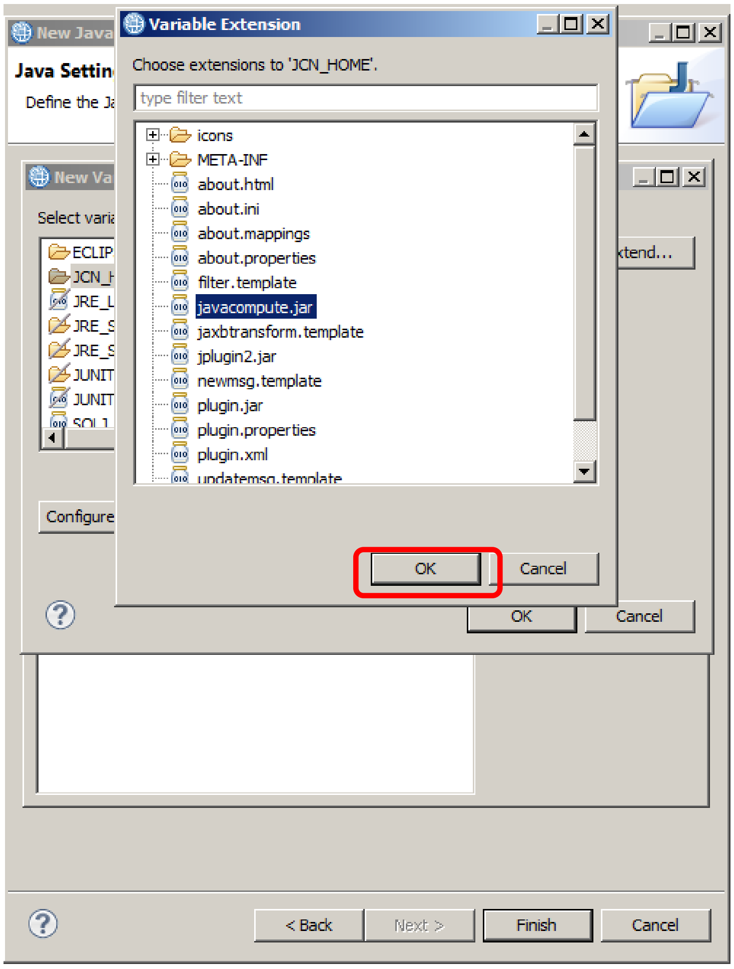


4. Choose “JCN_HOME” from the list of variables and click “Extend”:

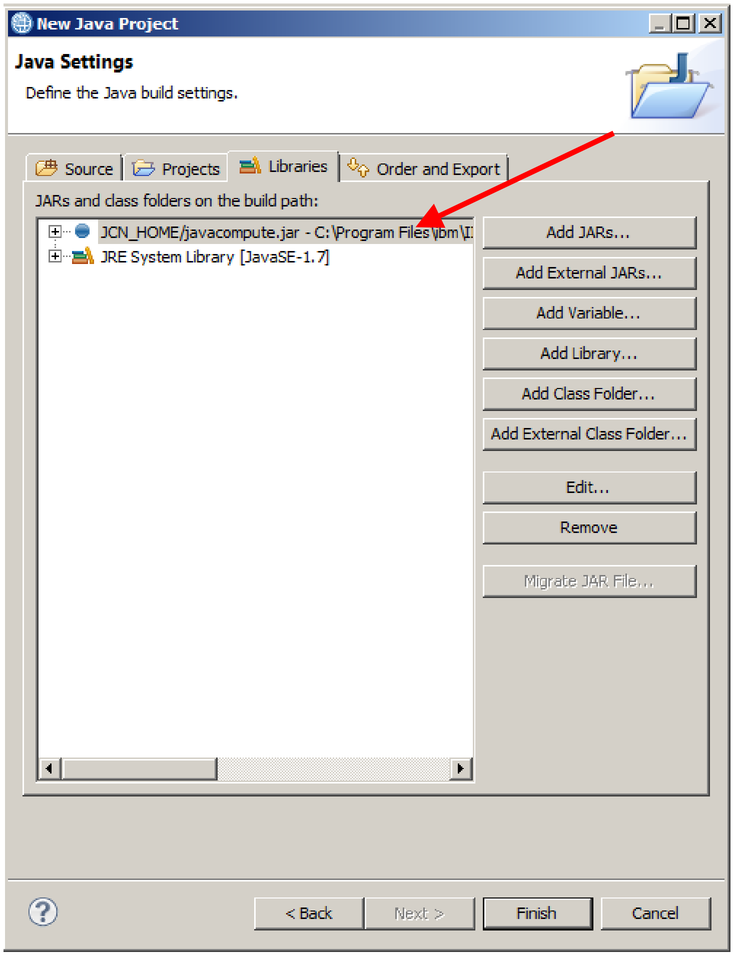

New Java Project
Java Settings
Define the Java build settings.
New Variable Classpath Entry
Select variables to add to build path:
Extend...
Configure Variables...
OK
Cancel
< Back
Next >
Finish
Cancel

ECLIPSE_HOME - C:\Program Files\ibm\IIB\10.0.0.0\tools
JCN_HOME - C:\Program Files\ibm\IIB\10.0.0.0\tools\plugins\com.ibr
JRE_LIB (deprecated) - C:\Program Files\ibm\IIB\10.0.0.0\common\j
JRE_SRC (deprecated) - (empty)
JRE_SRCROOT (deprecated) - (empty)
JUNIT_HOME (deprecated) - C:\Program Files\ibm\IIB\10.0.0.0\tools
JUNIT_SRC_HOME (deprecated) - C:\Program Files\ibm\IIB\10.0.0.0
SQLJ_JAR - C:\IBM\SQLLIB\java\sqlj.zip

5. From the list of extensions choose “javacompute.jar” from the list of options and click OK:



6. This will add the javacompute.jar to the build path:

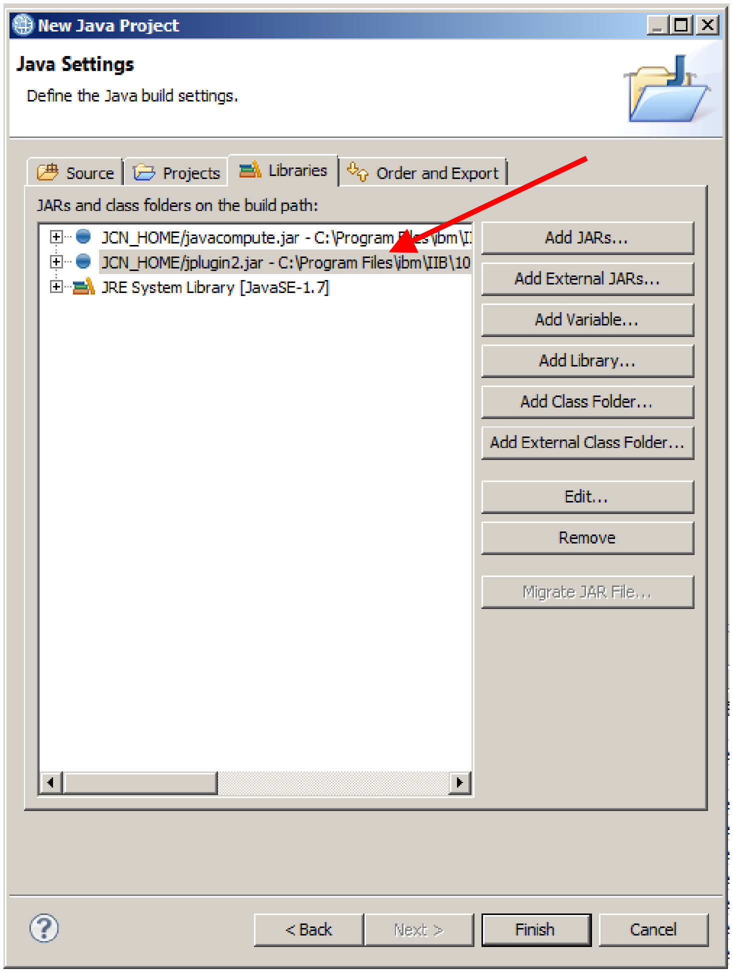


JARs and class folders on the build path:

JCN_HOME/javacompute.jar - C:\Program Files\ibm\I

JRE System Library [JavaSE-1.7]

7. Repeat the above steps to also include “jplugin2.jar” to the build path. The resulting JARs on the buildpath should look like this:



New Java Project

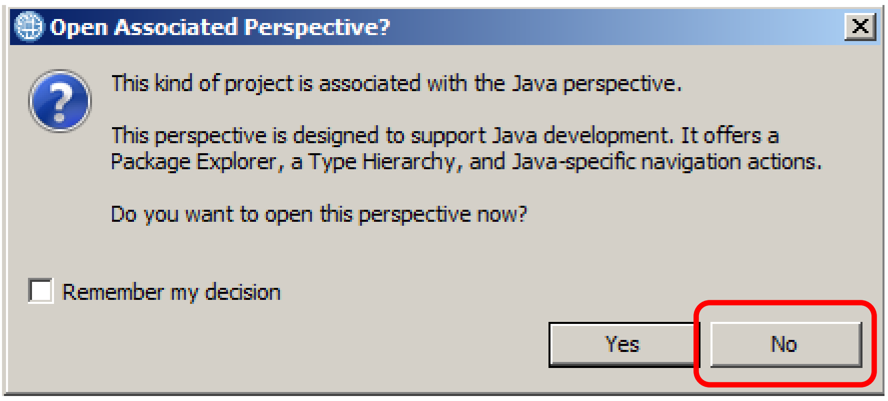
Java Settings

Define the Java build settings.

JARs and class folders on the build path:

- JCN_HOME/javacompute.jar - C:\Program Files\ibm\I
- JCN_HOME/jplugin2.jar - C:\Program Files\ibm\IIB\10
- JRE System Library [JavaSE-1.7]

8. Click Finish to create the project. Click “No” when asked about opening the Java perspective:



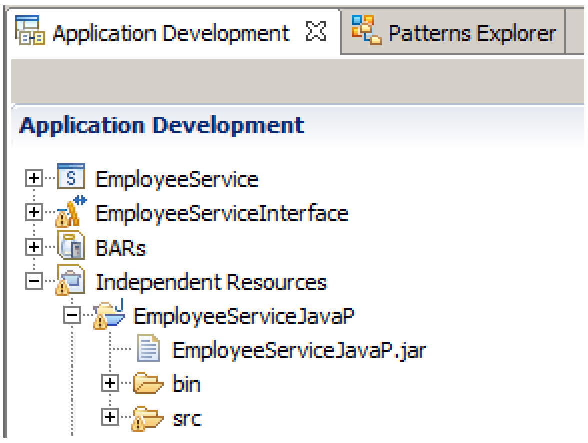
Open Associated Perspective?

This kind of project is associated with the Java perspective.

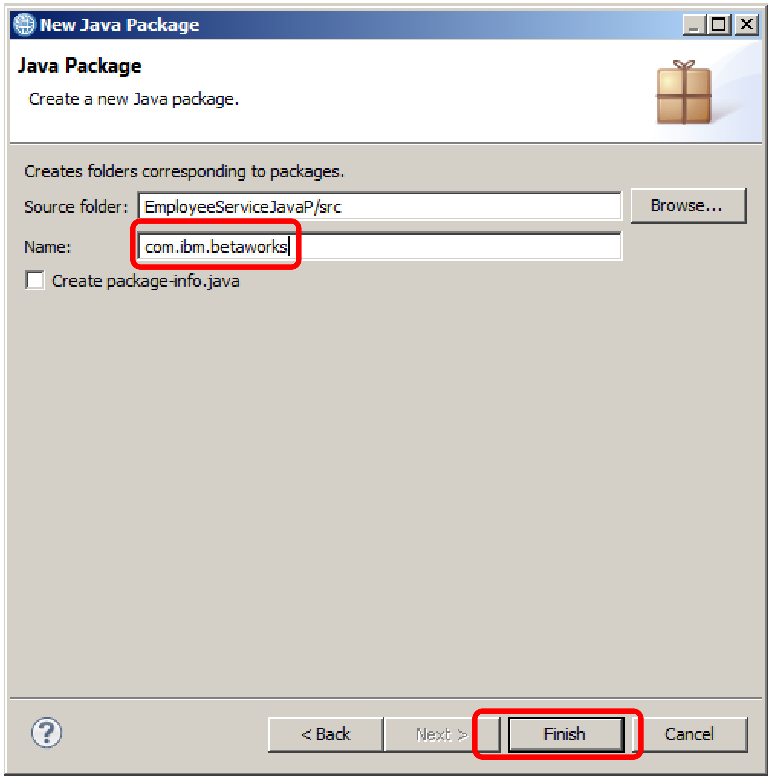
This perspective is designed to support Java development. It offers a Package Explorer, a Type Hierarchy, and Java-specific navigation actions.

Do you want to open this perspective now?

The EmployeeServiceJavaP project will be created in your workspace under “Independent Resources”:



9. Right click on the project and click new and chose “Other”. From the Select wizard expand the Java folder and chose “Package”. The Create new Java package wizard will open. Specify “com.ibm.betaworks” as the package name and click Finish:


New Java Package

Java Package

Create a new Java package.

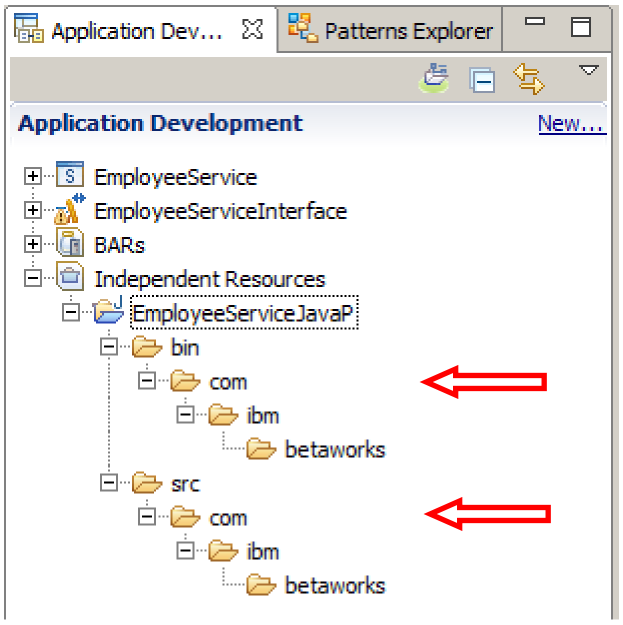
Creates folders corresponding to packages.

Source folder: EmployeeServiceJavaP/src

Name: com.ibm.betaworks

☐ Create package-info.java

10. Note that the “src” and “bin” folders now contain a hierarchy of folders containing the names “com”, “ibm” and “betaworks”:



9.2 Creating the Java Project for code called by the JCN

To create the EmployeeServiceJavaPJCN project:

1. Repeat the instructions in the previous section, with appropriate name change for the Java Project.

2. A JavaCompute node class should always be in a Java project with JCN nature. This needs to be specified in the .project file in the workspace folder. To do this, edit the file and add the sections highlighted:

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
      <name>EmployeeServiceJavaP</name>
      <comment></comment>
      <projects>
      </projects>
      <buildSpec>
            <buildCommand>
                  <name>org.eclipse.jdt.core.javabuilder</name>
                  <arguments>
                  </arguments>
            </buildCommand>
            <buildCommand>

      <name>com.ibm.etools.mft.java.builder.javabuilder</name>
                  <arguments>
                  </arguments>
            </buildCommand>
            <buildCommand>
                  <name>com.ibm.etools.mft.bar.ext.barbuilder</name>
                  <arguments>
                  </arguments>
            </buildCommand>
            <buildCommand>
                  <name>com.ibm.etools.mft.jcn.jcnbuilder</name>
                  <arguments>
                  </arguments>
            </buildCommand>
      </buildSpec>
      <natures>
            <nature>org.eclipse.jdt.core.javanature</nature>
            <nature>com.ibm.etools.mft.bar.ext.barnature</nature>
            <nature>com.ibm.etools.mft.jcn.jcnnature</nature>
      </natures>
</projectDescription>
```