IBM WebSphere Business Connection

Web Services Development Tutorial

Version 1.1.0

Note!
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 33.

First Edition (September 2002)

This edition applies to Version 1, Release 1, Modification 0, of *IBM® WebSphere® Business Connection* (5724-D26) and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. You can send them to the following address:

IBM Canada Ltd. Laboratory
Information Development
8200 Warden Avenue
Markham, Ontario, Canada L6G 1C7

Include the title and order number of this book, and the page number or topic related to your comment.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Web services tutorial

In the Basic samples document, a list of steps for developing CrossWorlds(R) Web services connectivity was presented. This list of steps is used as the basis of a tutorial example.

The tutorial guides you through developing an outbound CrossWorlds collaboration that uses a Web service provided by another CrossWorlds collaboration. However, before you begin, make sure you have followed the steps for installing WebSphere(R) and the Web Services Gateway, as described in the Installation and Configuration guide. In particular, note that CrossWorlds 4.1.1 with the Web Services 1.0.1 (or later) upgrade applied, and WebSphere 4.02 Advanced Edition, must be installed before the sample or this tutorial can be used. You should also load the Web Services sample (imported from file BCT_WS_Samples.in) into your CrossWorlds repository, because some of its parts are reused by the tutorial.

The steps from the Web Services Overview and Samples document are listed below for review. The steps are:

1. Define the generic business objects on the source and destination side.
2. Define the SOAP application-specific business objects for the SOAP connector and SAI.
3. Define the meta-objects for the SOAP application-specific business objects.
4. Define the mappings to convert from a generic business object to a SOAP application-specific business objects and from a SOAP application-specific business object to a generic business object.
5. Define the collaboration templates.
6. Define the collaboration objects.
7. Run the WSGenUtility to create the Java(TM) proxy class for invoking the SAI.
8. Create the SOAP Service.
9. Deploy the CrossWorlds artifacts and configure:
   a. The URL for the SOAP message to be sent to.
   b. The supported business objects used by the SOAP connector and other connectors involved.
   c. The meta-objects used by the SOAP connector and SAI so that they have references to your specific meta-objects.
10. Deploy the SOAP Service and configure:
    a. Log4j for tracing.
    b. Information for the Java proxy to use.
11. If the Web Services Gateway is being used, deploy the WSDL generated by the WSGenUtility into Web Services Gateway.

The tutorial is divided into three parts:

- Part I of the tutorial covers steps 1 through 7 of the list above.

  You use CWGenUtility to produce CrossWorlds business-object definitions. Then you use IBM CrossWorlds tooling to develop maps, collaboration templates, and

collaboration objects. Once these are finished, you use the IBM CrossWorlds WSGenUtility to produce the Java proxy, WSDL for the Web service, and related files.

- Part II of the tutorial covers steps 8 through 10 of the list above.

  In Part II, you deploy the proxy in a SOAP-enabled application server. Then you can test this by calling from the outbound collaboration using the SOAP connector to the inbound collaboration.

- Part III of the tutorial covers step 11 of the list above.

  In Part III, you deploy the WSDL in the Web Services Gateway so the service is provided as a gateway service. Then you can test calling from the outbound collaboration using the SOAP connector to the inbound collaboration through the Web Services Gateway. Note, however, that this testing is done using just one machine, rather than with two machines as described in the Samples User Guide.

Part I: Producing the CrossWorlds business objects and Java proxy artifacts

In this part of the tutorial, you will define the CrossWorlds and Java proxy artifacts.

First, you use the CWGenUtility program that comes with the Business Connection offering to produce a file containing the business-object definitions. You must supply the utility with a generic business-object name and a verb. Based on these names, the naming convention, and the design pattern for Web-services messaging, the utility produces an import file containing definitions for the generic business object, the application-specific business objects, and the meta-objects.

Defining the business-object definitions

You invoke CWGenUtility by running a batch file named runcwgenutility. Do the following:

1. Open a command prompt and make the current directory **\bctws\cw\cwgenutil.**

   In this directory are support files for the utility plus a batch file named runcwgenutility.bat. This batch file assumes that a Java runtime is available in the path and that the environment variable WAS_HOME is set by a WebSphere installation. If these assumptions are untrue, you can edit the file to correct the problems.

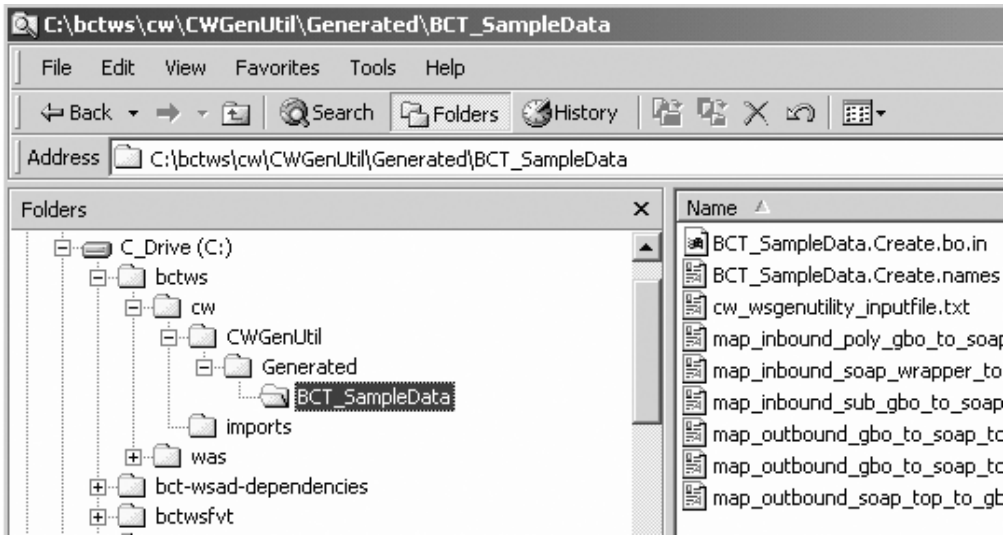
   The batch file requires these command line arguments:

   - The root directory of your CrossWorlds installation (for example, **d:\Crossworlds)**
   - The generic business-object name (in this case, **BCT_SampleData)**
   - The verb (in this case, **Create)**
   - The name of your Interchange Server (for example, **SSHCrossWorlds)**
   - The URL that you plan to use for the SOAP-enabled application server that hosts the Web service (for example, **http://localhost/crossworlds/servlet/messagerouter)**

2. Invoke this batch file (type as all one line), passing the required parameters. (Assume that all entries are case-sensitive.)

   ```
   runcwgenutility d:/crossworlds -BCT_SampleData_Create SSHCrossWorlds
   http://localhost/crossworlds/servlet/messagerouter
   ```

After the utility runs successfully, the output files are in directories under the CWGenUtil directory.



For each generic business object, a new directory is made under the **Generated** directory. If you are following the tutorial, you now have a **BCT_SampleData** directory. In this directory you will find the output of CWGenUtility, including:

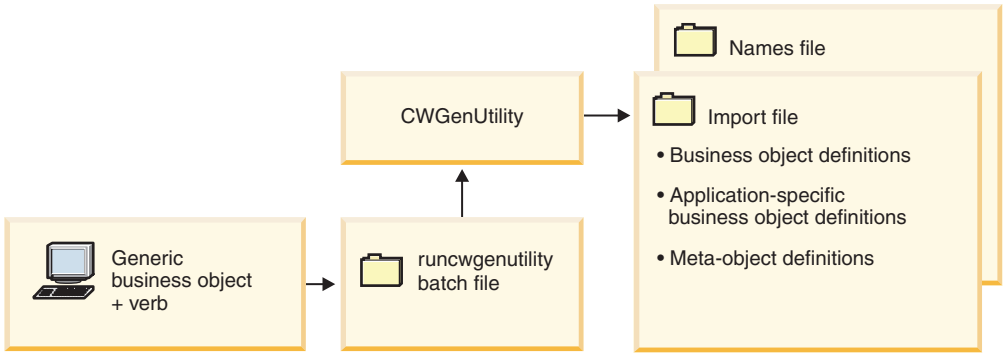
- **BCT_SampleDate.Create.bo.in**: The import file containing business-object, application-specific business object, and meta-object definitions.
- Several txt files containing code for the maps used in the outbound-to-inbound scenario.
- **BCT_SampleData**.**Create.names.txt**: A file listing all the names for the artifacts used in the outbound-to-inbound scenario. You might want to print this file to use as a reference as you go through the tutorial.

  The file includes names you need to know to complete the inbound-to-outbound message flow. The first few lines of this file are copied below for your reference. You will use the actual file as you follow the rest of the tutorial.

  Open this file in an editor now as you will need to copy and paste the names as you are creating the rest of the program artifacts used by the scenario.

  The names are rather long, so copy and paste is a good way to be sure you do not make mistakes entering them.

```
Filename: BCT_SampleData.Create.names.txt
*
*
*
*****************  BUSINESS OBJECT ARTIFACTS  *****************
GBO: BCT_SampleData
SOAP Fault ASBO: BCT_SOAP_BCT_SampleData_Fault
SOAP Wrapper ASBO: BCT_SOAP_BCT_SampleData_Wrapper
SOAP Top ASBO: BCT_SOAP_BCT_SampleData
Add the SOAP Wrapper ASBO to the Response and Request attributes of the SOAP
Top ASBO
Add the SOAP Fault ASBO to the Fault attribute of the SOAP Top ASBO
Add the GBO to the detail attribute of the SOAP Fault ASBO
Add the GBO as single child to the SOAP Wrapper ASBO
*
*
*****************************************************************
```

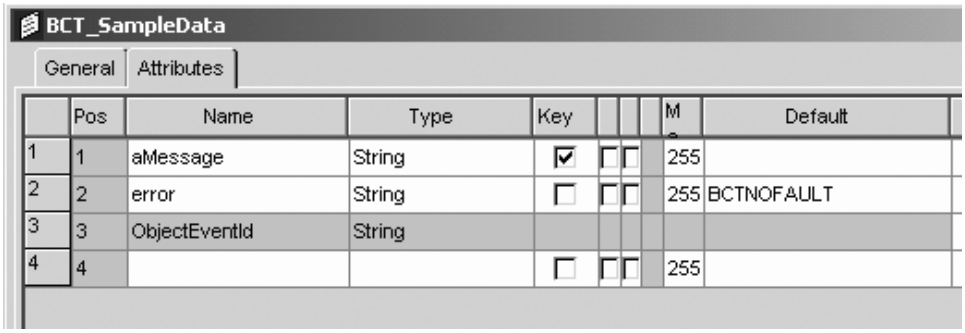


Note: The imported business objects refer to your generic business-object *type*, but the import does not include your generic business object. The import will fail unless you define your generic business object in the CrossWorlds repository first.

Defining the generic business object

Before you import the business objection definitions, define your generic business object.

1. Open the IBM CrossWorlds Business Object Designer.
2. Click **File > New**.
3. Enter the business object name (**BCT_SampleData**) and leave the application-specific information field blank.
4. Use the following illustration to complete the definition of the business object.



The generic business object for the tutorial contains two attributes.

- The attribute named **aMessage** provides a place for you to send a String message. Normally your generic business object would contain attributes relating to the business process with which you are working.
- The attribute named **error** is required by the Business Connection design pattern. It has a default value of **BCTNOFAULT.** (To use the recommended design pattern, the generic business object of any outbound-to-inbound Web service interaction must include the error attribute with a default value of BCTNOFAULT.) If the service encounters a processing error, it should fill the **error** field with information about the error. The Business Connection mappings will produce a SOAP fault message for return from the service to the caller in this case.

5. Save the business object definition to the server.

6. Leave the Business Object Designer open, because you will use it in the next section.

Importing the business-object definitions file

After you have created the generic business object, you import the business-object definitions file as follows:

1. From the CrossWorlds System Manager, click **File > Open from file**.
2. Navigate to the \bctws\cw\CWGenUtil\Generate\BCT_SampleData directory and click **BCT_SampleData.Create.BO.in**.
3. Click **Open**.

After importing the file, filter the Business Objects display using ***SampleData*** to see your generic business object (which you defined before importing) plus all of the necessary application-specific business objects and meta objects for the inbound-to-outbound scenario.

The CWGenUtility handles much of the complexity of the Web-services scenario for you by producing these business objects.

The next step is to update the system meta-objects. This must be done manually; it is not handled by the import file.

Updating the system meta-objects

You will next add the generated meta-object definitions to the meta-objects that are used by the Interchange Server (ICS). The meta-objects are used to determine how business objects are converted to and from SOAP messages.

There are four system meta-objects that you need to update. When the instructions direct you to add information to the fields, cut and paste the names from the names files described in the previous section.

Note that if you have imported the Web Services samples, there are already meta-objects in your repository. These are named with the word "Sample" prefixed to the base meta-object names. Use these meta-object definitions if you are set up to use the samples (per the Web Services Overview and Samples document).

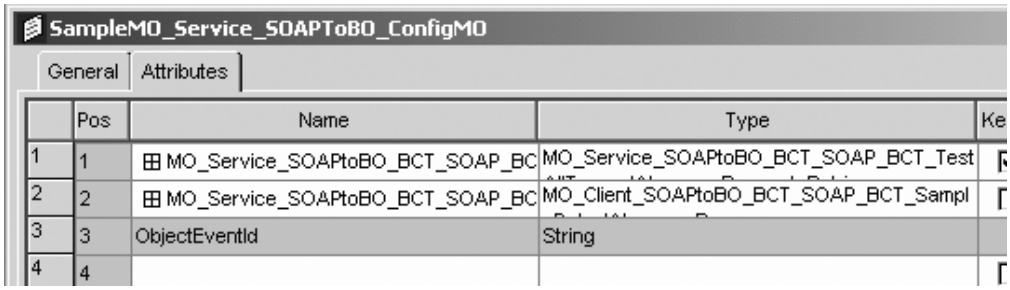
Updating MO_Service_SOAPToBO_ConfigMO

1. From the Business Object Designer, click **MO_Service_SOAPToBO_ConfigMO** to open it.
2. Click the **Attributes** tab.
3. Add the following in the **Name** field:

   `MO_Service_SOAPtoBO_BCT_SOAP_BCT_SampleData_Wrapper_Request_Create`

   Note: Copy and paste this name if possible to prevent typing errors. Remember that a names file was provided by the CwGenUtility that you can use for this purpose.
4. Into the **Type** field, add:

   `MO_Service_SOAPtoBO_BCT_SOAP_BCT_SampleData_Wrapper_Request_Create`
5. In the **Cardinality** field, type: 1
6. If this is the first attribute, check the **Key** box, because the meta-object must have a key defined.
7. Save the changes.



Updating MO_Service_BOToSOAP_ConfigMO

You add two lines to this meta-object. Do the following:

1. From the Business Object Designer, click **MO_Service_BOToSOAP_ConfigMO** to open it.
2. Click the **Attributes** tab.
3. Add the following in the **Name** field:

   `MO_Service_BOtoSOAP_BCT_SOAP_BCT_SampleData_Wrapper_Response_Create`
4. In the **Type** field, add:

   `MO_Service_BOtoSOAP_BCT_SOAP_BCT_SampleData_Wrapper_Response_Create`
5. In the **Cardinality** field, type: 1
6. If this is the first attribute, check the **Key** box, because the meta-object must have a key defined.
7. Add the following in the **Name** field:

   `MO_Service_BOtoSOAP_BCT_SOAP_BCT_SampleData_Fault_Fault_Create`

8. In the **Type** field, add:
   `MO_Service_BOtoSOAP_BCT_SOAP_BCT_SampleData_Fault_Fault_Create`
9. In the **Cardinality** field, type: 1
10. Save the changes.

Updating MO_Client_SOAPToBO_ConfigMO

You add two lines to this meta-object. Do the following:

1. From the Business Object Designer, click **MO_Client_SOAPToBO_ConfigMO** to open it.
2. Click the **Attributes** tab.
3. Add the following in the **Name** field:
   `MO_Client_SOAPtoBO_BCT_SOAP_BCT_SampleData_Wrapper_Response_Create`
4. In the **Type** field, add:
   `MO_Client_SOAPtoBO_BCT_SOAP_BCT_SampleData_Wrapper_Response_Create`
5. In the **Cardinality** field, type: 1
6. If this is the first attribute, check the **Key** box, because the meta-object must have a key defined.
7. Add the following in the **Name** field:
   `MO_Client_SOAPtoBO_BCT_SOAP_BCT_SampleData_Fault_Fault_Create`
8. In the **Type** field, add:
   `MO_Client_SOAPtoBO_BCT_SOAP_BCT_SampleData_Fault_Fault_Create`
9. In the **Cardinality** field, type: 1
10. Save the changes.

Updating MO_Client_BOToSOAP_ConfigMO

To update this file:

1. From the Business Object Designer, click **MO_Client_BOToSOAP_ConfigMO** to open it.
2. Click the **Attributes** tab.
3. Add the following in the **Name** field:
   `MO_Client_BOtoSOAP_BCT_SOAP_BCT_SampleData_Wrapper_Request_Create`
4. In the **Type** field, add:
   `MO_Client_BOtoSOAP_BCT_SOAP_BCT_SampleData_Wrapper_Request_Create`
5. In the **Cardinality** field, type: 1
6. If this is the first attribute, check the **Key** box, because the meta-object must have a key defined.
7. Save the changes.

Developing the maps

To create the maps, you use the CrossWorlds Map Designer tool. Notice that the naming convention for the maps uses the source and target business-object type names. For example, the Map for the outbound request has a source object of type BCT_SampleData and a target of BCT_SOAP_BCT_SampleData. Here are the five types of maps and the names generated by CWGenUtility. Refer to the generated names file for these name references if you want to copy and paste them.

- MAP Outbound Request:
  `BCT_SampleData_to_BCT_SOAP_BCT_SampleData`

- MAP Outbound Response:
  `BCT_SOAP_BCT_SampleData_to_BCT_SampleData`
- MAP Inbound Request:
  `BCT_SOAP_BCT_SampleData_Wrapper_to_BCT_SampleData`
- MAP Sub Fault Inbound Response:
  `BCT_SampleData_to_BCT_SOAP_BCT_SampleData_Fault`
- MAP Poly Inbound Response:
  `Poly_BCT_SampleData_to_BCT_SOAP_BCT_SampleData_Wrapper`

Use the information conveyed in the name as you create each map with the Map Designer.

Creating the outbound request map

1. To create the **Outbound Request** map, open the Map Designer.
2. Click **File > New.**
3. Choose the source and then the target types as suggested by the name of the map. Note that the source is the first business object named in the map name and the target is the second business object named in the map name.

   When you get to the dialog where you enter the map name, you should copy and paste the name. After naming the map, the Map Designer screen is displayed.

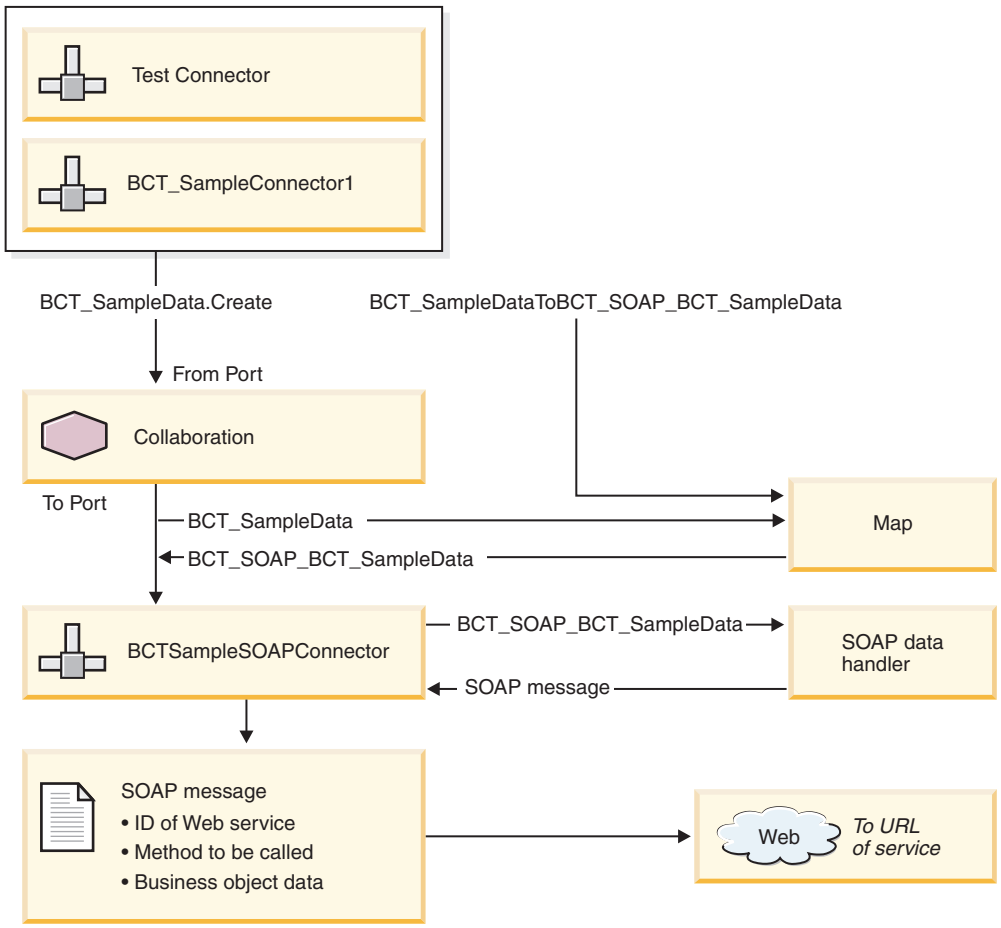
4. Click the **Diagram** tab of this screen.

   There are four fields to set in the Top SOAP application-specific business object, which is the target object of this map.
5. Set the Rule for the **URL** field to **Custom**.
6. Copy and paste the entire contents of the generated file named map_outbound_gbo_to_soap_top_url_request.txt as the custom code.
7. Set a value of **xml/soap** in the **MimeType** field. When entering the value, do *not* use quotation marks.
8. Set a value of **CxIgnore** in the **BOprefix** field; again, do not use quotation marks when entering the value.
9. Set the Rule for the **Request** field to **Custom**.
10. Copy and paste the entire contents of the generated file named map_outbound_gbo_to_soap_top_request_request.txt as the custom code.

    Finally, every time you make a new map, you must set the verb of the target object; otherwise the target object will not have any verb, and you will have a failure that is logged at runtime.

11. Set the verb by clicking the **Verbs** tab.
12. From the Verb list double-click the **Get From** choice that names the **ObjBCT_SampleData** source.
13. Save the map, compiling it when requested to do so.

The figure below shows you the map you created and how it will be used in the processing of the outbound request.



Creating the outbound response map

Next you will create the outbound response map. The names file shows that it is named BCT_SOAP_BCT_SampleData_to_BCT_SampleData, so you can see the source and target business object types.

1. Create the map as before, choosing the types and naming it.
2. Again select the **Diagram** tab on the Map Designer page. There is only one mapping field needed.
3. Select the field on the target object named **aMessage** and set a **Custom** rule. Generally, just pick the first field in the target for the placement of the Custom code.
4. Copy and paste the entire contents of the generated file named map_outbound_soap_top_to_gbo_response.txt as the custom code.
5. Click the **Verbs** tab.
6. Set the verb for the target object to use the verb of the source object.
7. Save the map, compiling it when requested to do so.

Creating the inbound response map

Next you will create the inbound request map named **BCT_SOAP_BCT_SampleData_Wrapper_to_BCT_SampleData.**

1. Create the map as before using the same technique. There is only one mapping field needed.
2. Select the field on the target object named **aMessage** and set a **Custom** rule.

3. Copy and paste the entire contents of the generated file named map_inbound_soap_wrapper_to_gbo_request.txt.
4. Click the **Verbs** tab.
5. Set the verb for the target object to use the verb of the source object.
6. Save the map, compiling it when requested to do so.

Creating the inbound request sub-map

Next you will create the inbound request sub-map named **BCT_SampleData_to_BCT_SOAP_BCT_SampleData_Fault.**

1. Create the map as before using the same technique. There are three mapping fields needed.
2. Move the value of the source **error** attribute to the target **faultstring** attribute. You move it by pressing Ctrl while dragging the **error** attribute and then dropping it onto the **faultstring** attribute.
3. Set a value of **env:Client** in the faultcode field. When entering the value, do *not* use quotation marks.
4. Set the Rule for the detail field to **Custom**.
5. Copy and paste the entire contents of the generated file named map_inbound_sub_gbo_to_soap_fault_response.txt as the custom code.
6. Click the **Verbs** tab.
7. Set the verb for the target object to use the verb of the source object.
8. Save the map, compiling it when requested to do so.

Creating the polymorphic map

Finally, you will create the inbound request map named **Poly_BCT_SampleData_to_BCT_SOAP_BCT_SampleData_Wrapper.** This is a *Polymap*, which means that it has more than one type of target business object. Only one type is mentioned when the target is named. This map names the wrapper type, which is the type used when no error occurs processing the message. The other type that can be produced is the SOAP fault type (in this case, **BCT_SOAP_BCT_SampleData_Fault**.)

1. When creating the map, choose one source type (**BCT_SampleData**) and *two* target types (**BCT_SOAP_BCT_SampleData_Wrapper** and **BCT_SOAP_BCT_SampleData_Fault**). There is only one mapping field needed.
2. Select any field on either target object and set a **Custom** rule.
3. Copy and paste the entire contents of the generated file named map_inbound_poly_gbo_to_soap_wrapper_response.txt as the custom code.

   Set the Verb for the target objects. This is a *Poly map*, which differs from the maps you have created so far. A polymorphic map can return more than one type of object. For this poly map, the verbs of all the possible return objects must be set. There are two possible return objects.

4. Click the **Verbs** tab
5. Set both targets using the source verb as shown below:
6. Save the map, compiling it when requested to do so.

All the maps are now complete.

Developing collaboration templates

Refer again to the generated names file. The names of the inbound and outbound collaboration templates are:

- Collaboration template inbound:

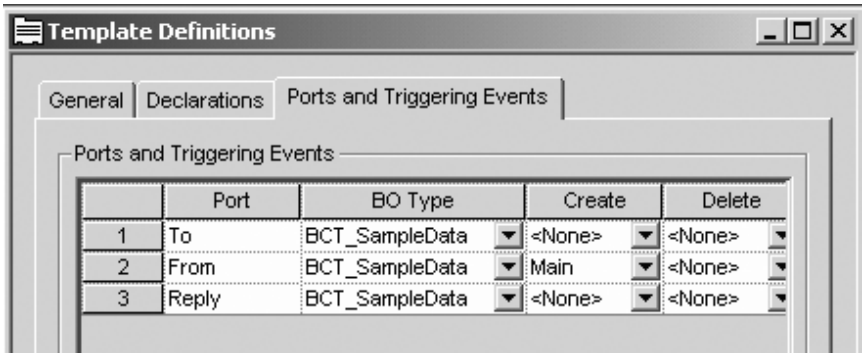
  `BCT_SampleDataCreateInbound`
- Collaboration template outbound:

  `BCT_SampleDataCreateOutbound`

To produce these templates, you can open and rename sample collaboration templates that are provided with the Business Connection offering samples.

1. From the CrossWorlds Process Designer, open the sample template named **BCT_TestAllTypesOutbound**.
2. After it is open, select **File > Save as >To server,** and save it with the name **BCT_SampleDataCreateOutbound**.

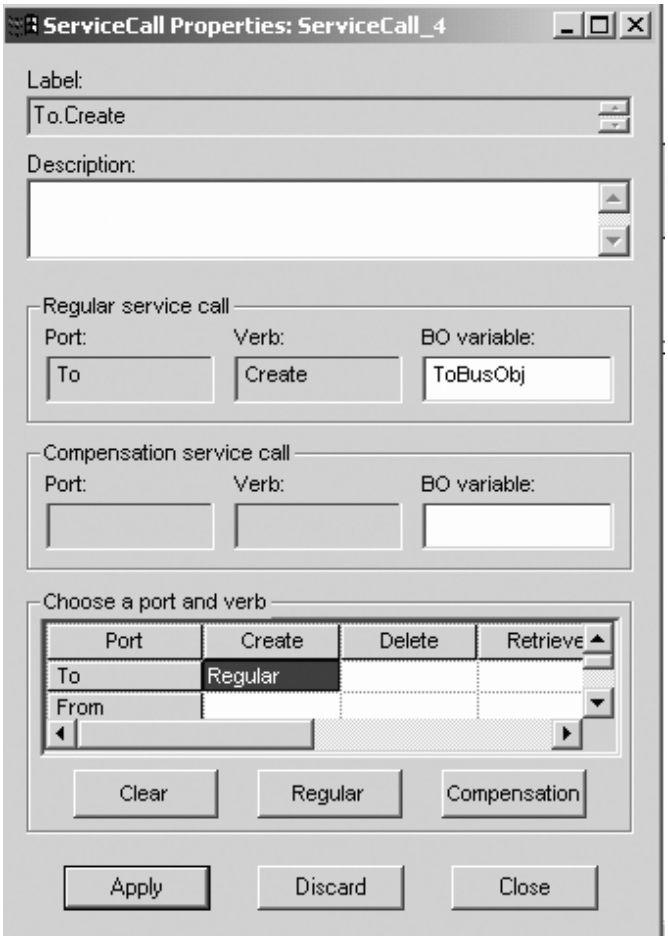
A **BCT_TestAllTypes** object with the **Retrieve** verb triggers the sample you just renamed. But a BCT_SampleData object with the Create verb triggers the tutorial collaboration. You must set this behavior next.



1. Open the **Definitions** of the collaboration template.
2. Click the **Ports and Triggering Events** tab.
3. Change the business object Type for all three ports to **BCT_SampleData**.
4. For the **From** port, click **Main** for the **Create** verb.
5. Click **Apply** and then close the window.

   This establishes a subscription for the Main scenario to **BCT_SampleData.Create** events.

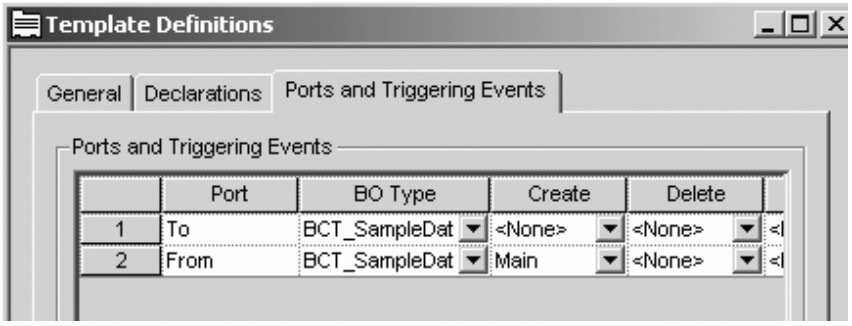
6. Change the name of the action node just below the start node from **Retrieve** to **Create**.
7. Change the **To** and **Reply** service calls to use the **Create**verb. Double-click each service call and change its properties as shown below. You will need to clear the word Regular from the Retrieve column and add it to the Create column by clicking on **Regular**.



Now repeat the process for the inbound collaboration template.

1. Open **BCT_TestAllTypesInbound.**
2. Select **File > Save as >To Server,** and save it with the name **BCT_SampleDataCreateInbound**.

   The **Template Definitions** should look like this when you are finished:



3. Change the **To** service call to reflect the use of the **Create** verb. See the earlier steps for the outbound collaboration if you don't recall how to do this step.
4. Compile and save the template.

Defining collaboration objects

A collaboration object is an instance of a collaboration template that is bound to ports. Before creating the collaboration objects, you will prepare connectors so they can be bound to the ports of the collaboration objects.

1. Open **BCT_SampleConnector1** and select the **Supported Business Objects** tab.
2. Add **BCT_SampleData** to the list, checking the **Agent Support** check box.
3. Save and close the window.
4. Repeat steps 1 through 4 for **BCTSampleConnector2**.
5. Open the **BCTSampleSOAPConnector**.
6. Add **BCT_SampleData** ***without*** checking **Agent Support**.
7. Add **BCT_SOAP_BCT_SampleData** and ***do*** check **Agent Support**.
8. Save and close the window.

You can find the names of the collaboration objects in the names file:

- Collaboration object outbound:
  `BCTSampleConnector1_to_BCTSampleSOAPConnector_BCT_SampleDataCreateOutbound`
- Collaboration Object Inbound:
  `SAI_to_BCTSampleConnector2_BCT_SampleDataCreateInbound`

Creating the outbound collaboration object

To create the object:

1. Right-click the **Collaboration Objects** folder that is under the **Integration Components** folder.
2. Select **New Collaboration Object.**
3. Select the template named **BCT_SampleDataCreateOutbound** and copy and paste the name for the outbound collaboration object.
4. Click **Next** (and continue to click it without completing any panels) until the **Finish** button is displayed.
5. Click **Finish**.
6. Bind the **From** and **Reply** ports to **BCTSampleConnector1.**
7. Bind the **To** port to the **BCTSampleSOAPConnector**.

Now you are finished with the outbound collaboration object. No explicit save is necessary.

Creating the inbound collaboration object

To create the object:

1. Right-click the **Collaboration Objects** folder that is under the **Integration Components** folder.
2. Select **New Collaboration Object.**
3. Select the template named **BCT_SampleDataCreateInbound** and copy and paste the name for the inbound collaboration object.
4. Click **Next** (and continue to click it without completing any panels) until the **Finish** button is displayed.
5. Click **Finish**.
6. Bind the **From** port to the **External Connector.**
7. Bind the **To** port to **BCTSampleConnector2.**

8. You must add maps to the **From** port. You do this by double-clicking the **From** port. This opens a window where you can drag-and-drop maps for the request and response.
9. Drag the map named **BCT_SOAP_BCT_SampleData_Wrapper_to_BCT_SampleData** and drop it as the Incoming Map.
10. Drag the **Poly_BCT_SampleData_to_BCT_SOAP_BCT_SampleData_Wrapper** and drop it as the Outgoing Map.
11. Because you have changed map-supported business objects, you must stop the Interchange Server and restart it to refresh all references to the connector properties in the system. Do so now.

Developing Java proxy and WSDL using WSGenUtility

The CrossWorlds WSGenUtility generates a Java proxy class that can be deployed as a Web service. The proxy uses a configuration file to obtain a connection to the Interchange Server so it can pass SOAP messages for processing by an inbound collaboration. The WSGenUtility also produces a WSDL that describes the structure and content of the SOAP messages that can be handled by the inbound collaboration.

Now that all of the CrossWorlds inbound artifacts are complete, the WSGenUtility can be used.

1. Display a command prompt and navigate to directory <crossworlds>\DevelopmentKits\WebServices.
2. Enter the following command to start the utility.

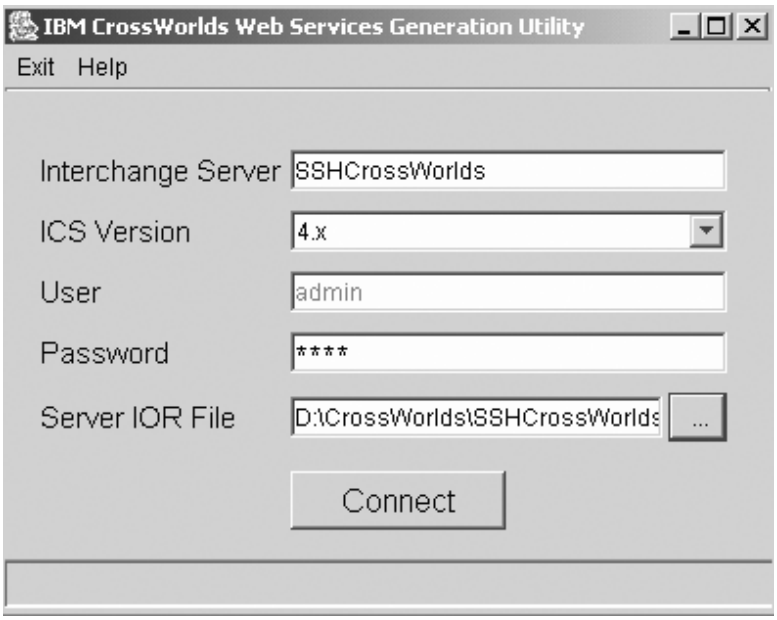
   ```
   WSGenUtility
   ```

   The WSGenUtility requires that you provide some information. The names required by the **WSGenUtility** are provided by the **CWGenUtility** in a file named cw_wsgenutility_inputfile.txt.
3. Bring this file up in an editor now so you can copy and paste names from it.

The contents of **cw_wsgenutility_inputfile.txt** are copied below for your reference. Note that your file will differ from the sample below in the specific names used on your system.

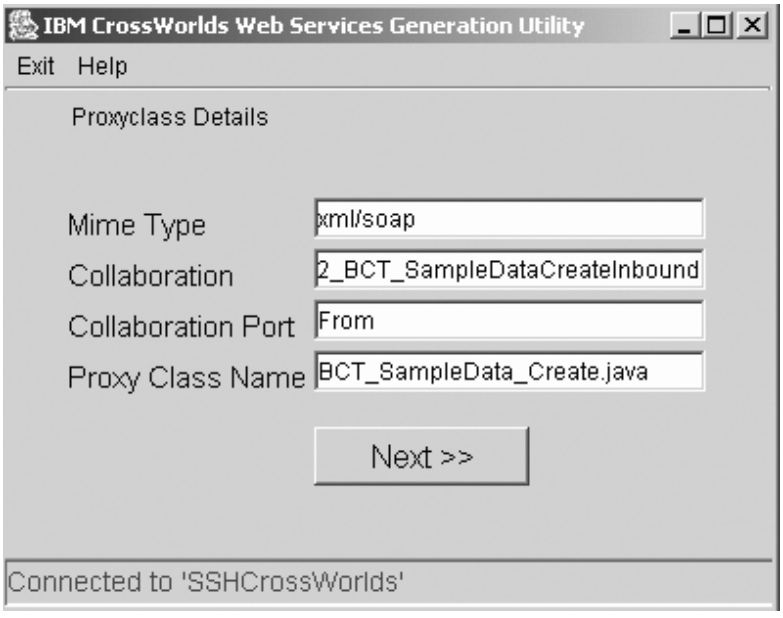
```
INTERCHANGESERVER=SSHCrossWorlds
ICS_VERSION=4.X
ICS_USER=admin
ICS_PASSWORD=null
ICS_SERVER_IOR_FILE=d:/crossworlds/SSHCrossWorldsInterchangeServer.ior

MIME_TYPE=xml/soap
COLLABORATION=SAI_to_BCTSampleConnector2_BCT_SampleDataCreateInbound
COLLABORATION_PORT=From
PROXYCLASSNAME=BCT_SampleData_Create.java

SERVICENAME=BCT_SampleData_Create
TARGETNAMESPACE=urn:ibmwsgw#BCT_SampleData_Create
BINDINGURL=http://localhost/crossworlds/servlet/messagerouter
CHECKTYPESINWSDL=true
INPUTMESSAGEMETAOBJECT1=MO_Service_SOAPtoBO_BCT_SOAP_BCT_SampleData
  _Wrapper_Request_Create
OUTPUTMESSAGEMETAOBJECT1=MO_Service_BOtoSOAP_BCT_SOAP_BCT_SampleData
  _Wrapper_Response_Create
FAULTMESSAGEMETAOBJECT1=MO_Service_BOtoSOAP_BCT_SOAP_BCT_SampleData
  _Fault_Fault_Create
```

1. Fill in the fields in the first dialog according to your configuration. Note that you cannot type in the location of the server IOR file; instead you must use the file dialog button to select the IOR file.



2. Press the **Connect** button.

3. Copy and paste the information into the second dialog.



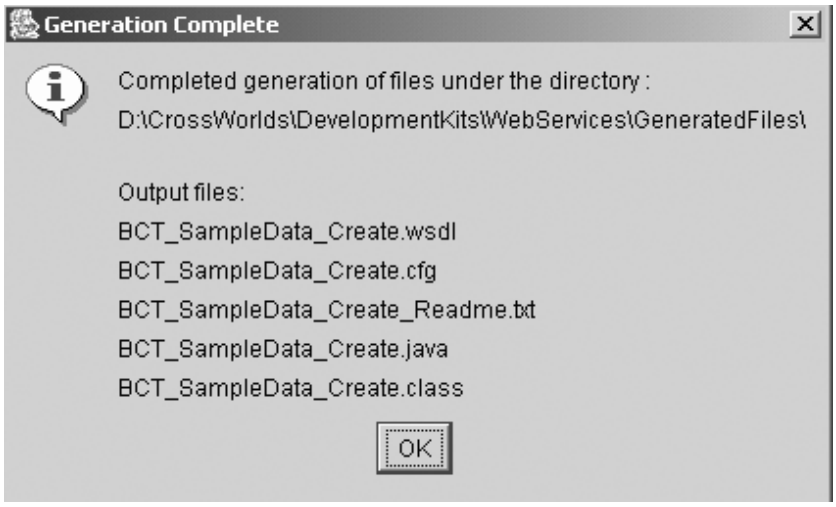
Note that you can have a package qualifier on the Proxy Class Name, although you do not have one in this example. Also note that you must have the **java** file extension on the class name you enter.

4. Press the **Next** button to display the final window.
5. In this window, you fill in the fields with values from the file **cw_wsgenutility_inputfile.txt.** The names are long and it is easy to type them

incorrectly, so copy and paste is especially useful here. Be sure to click **Types in the same WSDL,** because it simplifies deployment into the Web Services Gateway.

6. Click **Generate**.

The next window shows you the result of using this utility:



The class and cfg files created by the utility are used to deploy the Web service in a SOAP-enabled application server. These files are saved in the **<crossworlds>\DevelopmentKits\WebServices\Generated**subdirectory.

You are now finished with the first part of the tutorial.

Part I review

During Part 1, you did the following:

1. You used CWGenUtility (by running the runcwgenutility batch file and entering a business-object name and other information) to create the following:
   a. Generic business-object definitions.
   b. SOAP application-specific business objects.
   c. Meta-objects for the SOAP application-specific business objects.
2. You used the Business Object Designer to create the generic business object.
3. You used the Business Object Designer to update the meta-objects.
4. You used Map Designer to define mappings.
5. You used the Process Designer to define collaboration templates.
6. You created collaboration instances.
7. You used WSGenUtility to create the Java proxy class and WSDL file.

You will deploy the Java proxy Web service in Part II of the tutorial.

The WSDL file created by the utility is used to deploy the Web service in the Web Services Gateway. You will do this in Part III of the tutorial.

Part II: Deploying the CrossWorlds Web service

This section describes how to:

- Create a SOAP Service
- Deploy the CrossWorlds artifacts and configure:
  - The URL for the SOAP message to be sent to
  - The supported business objects used by the SOAP connector and other connectors involved
  - The meta-objects used by the SOAP connector and SAI so that they have references to your specific meta-objects
- Deploy the SOAP Service and configure:
  - Log4j for tracing
  - Information for the Java proxy to use

Creating a Web service

You start by using the WebSphere Application Assembly Tool. This tool is described in more detail in Web Services Technical Reference.

You will create the service by modifying a template and then saving the new file under another name.

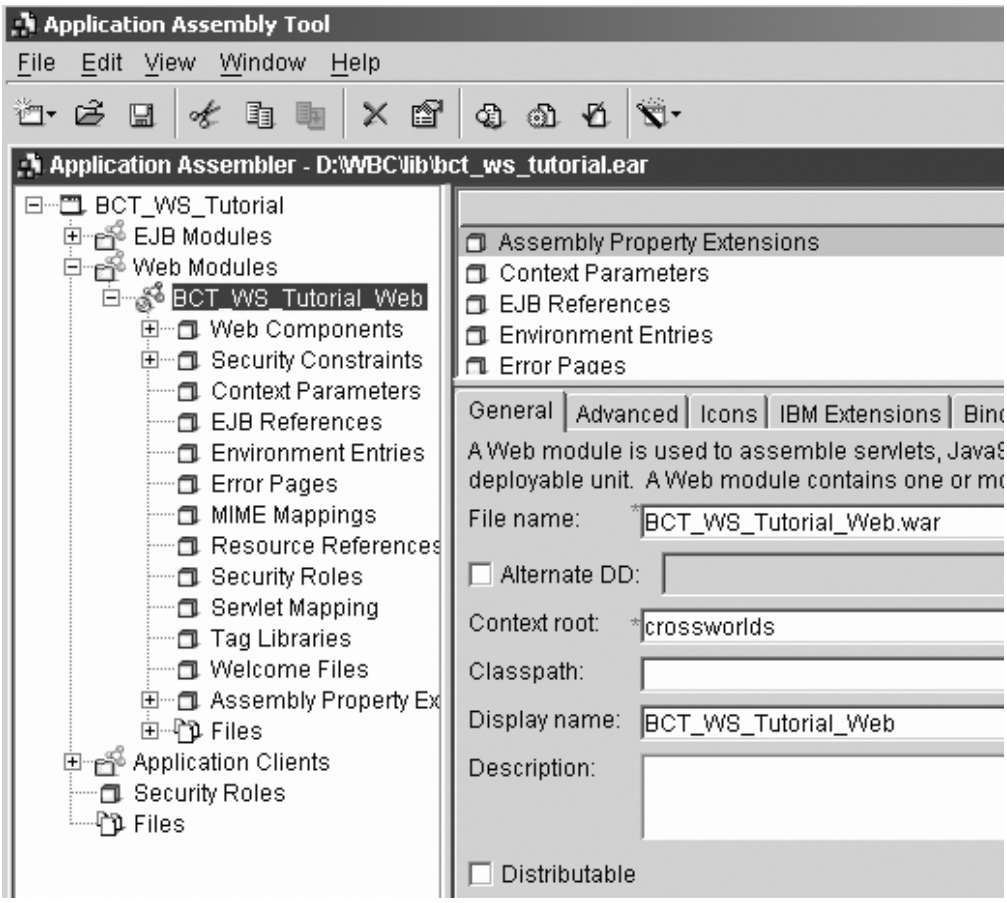
1. Open the WebSphere Application Assembly Tool.
2. Open **<*BCT_HOME*>\lib\bctwswebservicetemplate.ear**.
3. Click **BCT_WS_WebService_Template_EAR**.

   This is a template that you will change to reflect the service you are building.

4. Click the **General** tab.
5. In the **Display name** field, change the value to **BCT_WS_Tutorial**
6. Click **Apply**.
7. In the left pane, click **Web Modules > BCT_WS_WebService_Template**.
8. Click the **General** tab.
9. In the File name field, change the value to **BCT_WS_Tutorial_Web.war**
10. In the Display Name field, change the value to **BCT_WS_Tutorial_Web**
11. In the Context root field, change the value to **crossworlds**
12. Click **Apply**.
13. Click **File > Save as**.
14. Type:

    ```
    \bctws\was\installableapps\BCT_WS_Tutorial.ear
    ```

After changing the display names of the EAR and WAR, changing the context root of the WAR, and saving with the new name, the application looks like this in the Application Assembly Tool:



Importing the EAR file into WebSphere Studio Application Developer

Next, you import the new EAR file into WebSphere Studio Application Developer.

1. Open WebSphere Application Developer.
2. Click **File > Import > EAR file**.
3. Using the File dialog, select **bct_ws_tutorial.ear**.
4. Name the Enterprise Project:
   `BCT_WS_Tutorial_EAR`

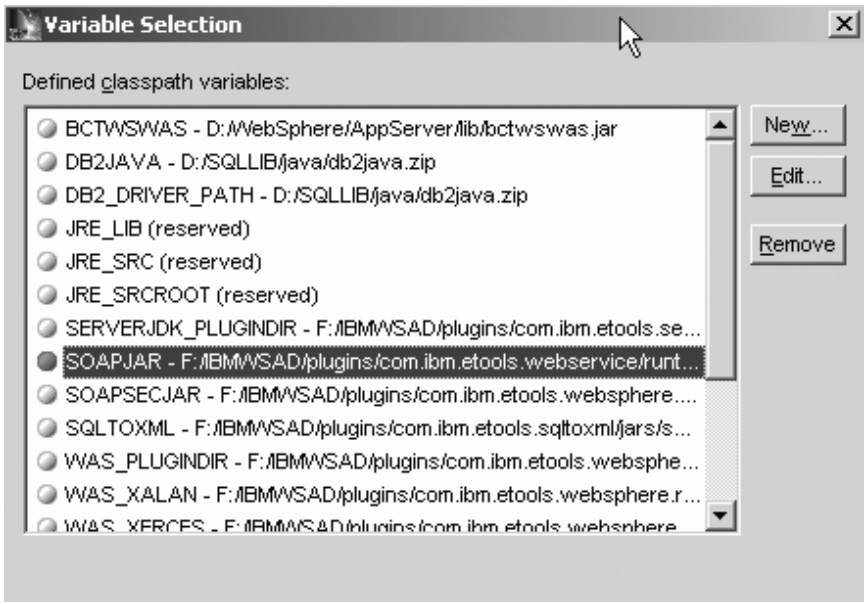
Note: You can do all of the deployment just with the Application Assembly Tool, but this tutorial does not describe this process. The WebSphere Application Developer process can serve as a guide to the necessary steps if you want to use the Application Assembly Tool.

After importing the EAR file, you will see some errors that need to be fixed. To correct the errors, you will add soap.jar to the build path of the Web project, as follows:

1. Switch to the **Navigator** view of the J2EE perspective.
2. Right-click the **BCT_WS_Tutorial_Web** and select **Properties** in the context menu.
3. Click **Java build path.**
4. Click the **Libraries** tab.

The problem is that there is no reference to **soap.jar**. This must be added.

5. Click **Add Variable.**
6. Click the **Browse** button.
7. Click the **New** button.
8. Add the name **SOAPJAR** and the file **<*WebSphere_Application_Developer_HOME*> /plugins/com.ibm.etools.webservice/runtime/soap.jar**
9. Click **OK** until you return to the list of libraries, and verify that **SOAPJAR** is in the list.



10. Click **OK** on each dialog to finish the process.

    The Web project will then rebuild, correcting the errors.

Importing the proxy

Next, import the proxy that you generated using WSGenUtility in Part I.

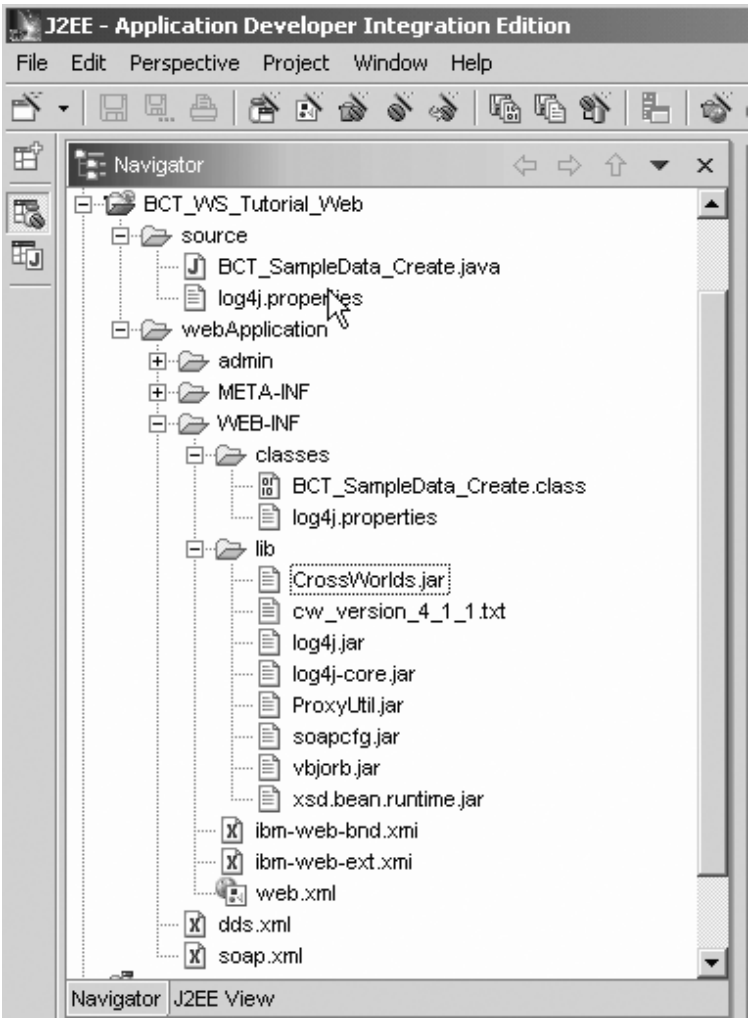
1. In the **Navigator** view of the J2EE perspective, expand the Web module until you locate the source folder.
2. Select the source folder and then click **File > Import** from the menu bar of WebSphere Application Developer.
3. Select **File System** as the source of the import.
4. For the import source, browse to **<crossworlds>\DevelopmentKits\WebServices\GeneratedFiles**.
5. Click *on the word* **GeneratedFiles** in the left pane. Then check the box next to **BCT_SampleData_Create.java** on the right pane.
6. Click **Finish** to import the file.

   After importing the source file, WebSphere Application Developer compiles it and places the class file into the classes directory under the WEB-INF directory.

7. Expand the lib directory and remove the file named **BCT_WS_Tutorial_Web.jar**.

   This file appears because you imported the EAR file into WebSphere Application Developer. It is not needed.

After you have done these steps, your Web project should look like the following:



Editing the SOAP deployment descriptor file

Next you edit the SOAP deployment descriptor file, dds.xml, which is located in the WEB-INF directory.

1. Open the file by right-clicking on **dds.xml** and then clicking **Open with > Default text editor**.
2. Read the prologue in dds.xml, which tells you what to do to add the service descriptor for the Web service.
3. Edit the readme file produced by the WSGenUtility.

   The readme file is in the GeneratedFiles directory with the other files produced by WSGenUtility. The file lists the names of the namespace, class, and method that you must use in the service descriptor.

For the tutorial example, the service descriptor should be added between the root element tags as shown below:

```
<root>
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
  id="urn:ibmwsgw#BCT_SampleData_Create"
 type="message" checkMustUnderstands="false">
  <isd:provider type="java" scope="Application"
 methods="m_BCT_SampleData">
    <isd:java class="BCT_SampleData_Create" static="false"/>
  </isd:provider>
</isd:service>

</root>
```

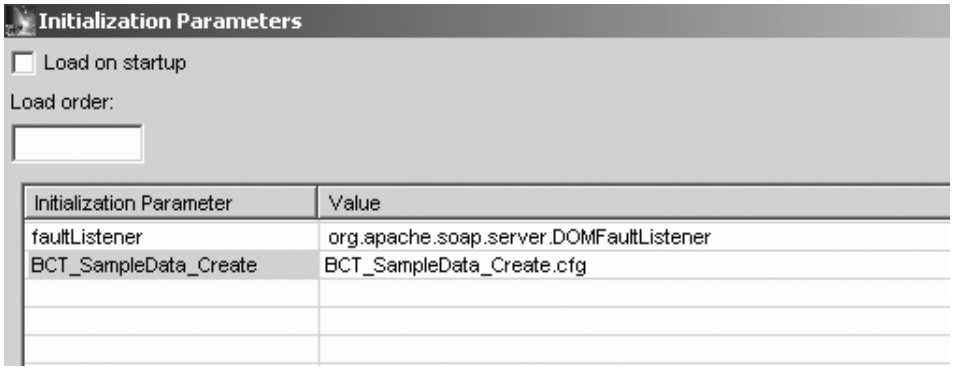
4. Save dds.xml and close it.

   Next, update the Servlet initialization parameters for the messagerouter servlet. This provides the location of the WSGenUtility proxy cfg file to the SOAP runtime engine.

5. Double-click the **web.xml** file that is in the WEB-INF directory.
6. Click the **Servlets** tab.
7. Click the **messagerouter servlet** to highlight it.
8. Click **Initialization**.

   Refer again to the readme produced by WSGenUtility. It tells you the key value to use for the initialization.

9. Add this as the initialization parameter, and add the cfg file as shown, with no path.



   This causes WebSphere to load the cfg file from its Working Directory.

10. Click **OK**.
11. Save and close web.xml.

Editing the log4j.properties file

Next, you will edit the file named **log4j.properties** in the **WEB-INF\source** directory. Again, refer to the readme produced by WSGenUtility for the category name to add.

1. Copy and paste the two sample lines for a service, and edit them with the correct name.

2. Save and close the **log4j.properties.**
   Now you are finished adding the service to the EAR file.
3. All editors (dds.xml, log4j.properties, and web.xml) should be closed. If they are open, close them now.

Exporting the EAR file

Export the EAR with the same name in the same directory from which you imported it:

1. Click **File > Export** from the WebSphere Application Developer menu bar.
2. Click **EAR file**.
3. Click **Next**.
4. Click the EAR file you just edited, and select the original location and name from which it was imported.
5. Click **Finish.**
6. When prompted, choose to over-write the original.

Deploying the EAR file

Now you will deploy the EAR file to WebSphere. The tutorial procedure assumes that you have followed the sample installation procedure. If you have not already done this, you must create the app server in WebSphere. This provides the sample application servers that are needed for the rest of the tutorial. Deploy the bct_ws_tutorial.ear file into WebSphere, associating it with the CrossWorlds app server.

Recall that WSGenUtility created a file named **BCT_SampleData_Create.cfg**. This file has connection information used by the Java proxy to connect to the ICS where the inbound collaboration resides. The contents of this file are similar to the following:

```
#Configuration Properties for BCT_SampleData_Create.java
 #Sun Jun 09 00:23:57 EDT 2002
 CollabName=SAI_to_BCTSampleConnector2_BCT_SampleDataCreateInbound
 PassWord=null
 ICSName=SSHCrossWorlds
 CWLDVersion=4.x
 UserName=admin
 MimeType=xml/soap
 CollabPort=From
 IORFileName=D\:\\CrossWorlds\\SSHCrossWorldsInterchangeServer.ior
```

Notice that if the ICS for the inbound collaboration was running on a different machine from WebSphere, you could configure the connection to the ICS machine using this file. You would copy the IOR file from the other machine to a local drive on the WebSphere machine, specify the path in the **IORFileName** line, and specify any other connection information as necessary.

You must copy the cfg file from **<crossworlds>\DevelopmentKits\WebServices\GeneratedFiles** to the working directory of the **CrossWorlds App Server** so the proxy class (which runs in the app server's JVM) can find it.

1. Copy this file now before starting the application server.

   Recall that an initialization parameter of the messagerouter servlet points to this directory for this file.

Note: If you do not copy the file, the proxy will use default values that were coded into it when WSGenUtility created it.

2. Install the tutorial EAR file that you created by right-clicking the **Enterprise Applications** and clicking **Install Enterprise Application.**
3. Choose the **CrossWorlds App Server** when you are installing the EAR file.
4. Start the **CrossWorlds App Server**.

Testing basic CrossWorlds connectivity

Now you can test the flow from the outbound collaboration, through the BCTSampleSOAPConnector, to the inbound collaboration.

Recall that in the CrossWorlds **start_server.bat** file, you added a definition of a Java system property to specify the location of a configuration file named **bct_ws_configuration.txt.** This file is used by the outbound collaboration's request map to look up the URL where the SOAP message will be sent.

This file is in directory **\bctws\cw.** You will edit the file.

Editing the configuration file

To edit the file:

1. Open the file in a text editor.
2. Add the following lines to it:

```
BCT_WS_BCT_SampleData_Create_URL=http://localhost/crossworlds/servlet/messagerouter
BCT_WS_BCT_SampleData_DestIDIdentifier=APartnerID
BCT_WS_BCT_SampleData_DestIDType=PartnerName
```

These lines provide the outbound request map with the URL to send to. They also provide the Destination Trading Partner ID.

Production collaborations could use another means of obtaining this information, but for the tutorial and the Business Connection offering samples, this file (in addition to a class that uses it for looking up this information) is provided. As you can see, the generic business-object type and verb are combined with a prefix and suffix to provide keys that the outbound request map can find using the look-up class. Look at the custom code in the outbound request map to see how this look-up is performed.

Adding these lines to the file and saving it enables the SOAP top-level application-specific business object to pick up this URL. The header in the message will pick up the partner information. This is for use later by the Web Services Gateway authentication and routing filters.

Creating an instance of the business object

1. Display the TestConnectors by clicking **Start > Programs > IBM CrossWorlds > Connectors > Test Connector**.

If you have gone through the Business Connection sample scenario that uses BCT_TestAllTypes, you should already have profiles in the Test Connector for

BCT_SampleConnector1 and BCT_SampleConnector2. If not, read the sample user guide and set these profiles up now.

2. Open one of the Test Connectors to BCT_SampleConnector1 and the other to BCT_SampleConnector2. Connect them to their respective agents.
3. Based on its supported business objects, BCT_SampleConnector1 will have your generic business object (BCT_SampleData) in its left-hand pane after it is connected to the agent.
4. Click **BCT_SampleData** and then click **Edit > Create BO**.

   A dialog will open where you can create an instance of **BCT_SampleData** for testing.
5. Select **Create** from the Verb list.
6. Type a name for the object.
7. Double-click the value field for **aMessage**.
8. Enter a message of your choice.
9. Click **OK**.

Sending a test message

1. Before continuing, display the **BCT_SampleSOAPConnector** connector agent. See the Business Connection samples instructions for setting up and starting this agent process.
2. Before sending the message, be sure both collaboration objects and all of their connectors are started. Also, be sure that the **CrossWorlds App Server** is started.
3. To send the message, click your generic business object instance in the left-hand pane of the Test Connector window to highlight it. Then click **Request > Send** to send it.

   After sending the message, you should see the message trace information in the **BCT_SampleSOAPConnector** agent window. This is because tracing was pre-configured for this connector. The agent window will stop showing the message xml. When the reply (or an error) occurs, the agent window will trace the response (or fault).

   Note: If you send a message and do not respond within approximately three minutes, an error will occur. This is because a default timeout in the IBM Http Server will close the socket for the response. This timeout can be changed using Http Server administration; however, be aware that such a timeout is necessary to manage the resources of the server.

Modifying the test message

1. Look at the Test Connector for **BCT_SampleConnector2**.
2. Click on **Request/Accept**. You should see the request message in the right-hand pane.
3. Double-click the message and change the value for **aMessage** before responding.
4. To respond, click the message in the right-hand pane (to highlight it) and click **Request > Reply Success**.
   The agent window will respond with tracing of the response.

Receiving the test message

1. To complete the flow, return to the Test Connector for **BCT_SampleConnector1**.
2. Click **Request/Accept** to receive the response message.
3. Double-click to see the response value in **aMessage**.
4. Click **Request > Reply Success** to complete the flow.

Note: If you do not perform all of these steps in order, the Test Connector can get out of sync and fail to work correctly. Disconnecting and reconnecting will correct this condition. Also, unresolved flows will result if you do not follow all steps in sequence every time.

After completing a flow, you can look at the **CrossWorlds App Server's** standard output file (\bctws\was\cw\logs\cwstdout.log) to see the Log4J tracing done by the Java proxy. For each interaction, you will see information similar to the lines that follow:

```
[6/10/02 11:00:42:746 EDT] 24040003 SystemOut     U [2002-06-10 11:00:42,656]
INFO160394[Servlet.Engine.Transports:25](ProxyClassParam.java:220) - Loading
parameters from config-file: BCT_SampleData_Create.cfg

[6/10/02 11:00:42:826 EDT] 24040003 SystemOut     U [2002-06-10 11:00:42,746]
INFO160484[Servlet.Engine.Transports:25](WebServicesAccessClient.java:129) -
Initializing Session with ICS using parameters:cwldVersion= 4.x icsName=
SSHCrossWorlds userName= admin passWord= null collabName
=SAI_to_BCTSampleConnector2_BCT_SampleDataCreateInbound collabPort= From
mimeType= xml/soap boVerb=Create iorFileName=
  D:\CrossWorlds\SSHCrossWorldsInterchangeServer.ior

[6/10/02 11:00:44:048 EDT] 24040003 SystemOut     U [2002-06-10 11:00:43,968]
INFO161706[Servlet.Engine.Transports:25](WebServicesAccessClient.java:221) -
Entering executingCollab. CollabName:
SAI_to_BCTSampleConnector2_BCT_SampleDataCreateInbound


[6/10/02 11:00:44:128 EDT] 24040003 SystemOut     U [2002-06-10 11:00:44,048]
INFO161786[Servlet.Engine.Transports:25](WebServicesAccessClient.java:230) -
Executing collaboration with parameters: cwldVersion= 4.x icsName=
SSHCrossWorlds userName= admin passWord= null collabName
=SAI_to_BCTSampleConnector2_BCT_SampleDataCreateInbound collabPort= From
mimeType= xml/soap boVerb=Create iorFileName=
  D:\CrossWorlds\SSHCrossWorldsInterchangeServer.ior


[6/10/02 11:00:46:812 EDT] 24040003 SystemOut     U [2002-06-10 11:00:46,732]
INFO164470[Servlet.Engine.Transports:25](WebServicesAccessClient.java:199) -
Closing Access Session. CollabName:
SAI_to_BCTSampleConnector2_BCT_SampleDataCreateInbound
```

Part II review

During Part II, you did the following:

1. From the WebSphere Application Assembly Tool, using the BCT_WS_WebService_Template_EAR file as a template, you created the BCT_WS_Tutorial EAR file. The EAR file defines the Web service.
2. Next, you imported the EAR file into WebSphere Application Developer.
3. You imported the Java proxy (which you created in Part I of the tutorial) into WebSphere Application Developer.
4. You edited the deployment descriptor file and the log4jproperties file for use with this sample.

5. You deployed the EAR file in WebSphere.
6. Finally, you tested the Web service by:
   a. Creating an instance of the business object, which contains a message.
   b. Sending a message from the outbound collaboration.
   c. Receiving the message to the inbound collaboration.
   d. Modifying the response message.
   e. Sending the modified message back to the outbound collaboration.
   f. Inspecting the message to see that it was actually changed.

In Part III, you will use the WSDL file (created by WSGenUtility in Part I of the tutorial) to deploy the Web service in the Web Services Gateway.

Part III: Deploying the service in Web Services Gateway

Part III of the tutorial describes how you deploy services in the Web Services Gateway.

The Web Services Gateway is provided as a collection of enterprise applications and JAR files. Installing Web Services Gateway is covered in the Installation and Configuration guide. If you have not already installed Web Services Gateway, refer to that documentation and install it now. When installing for the tutorial, you should use the Web Services Gateway App Server that was imported into your WebSphere node earlier.

The Business Connection offering uses the two SOAP 2.2 channels provided with Web Services Gateway. One channel should be considered a Public channel and the other a Private channel. See the Web Services Overview and Samples document for more about these channels and their deployment.

If you have deployed according to the sample setup instructions, you will have two channels in your Gateway. By convention the channel with URL **http://hostname/wsgwsoap2** is the Public channel. In the tutorial, everything is done on just one machine, so only the Public channel will be used.

Refer to the Web Services Overview and Samples document for discussion and examples using two gateways.

You will first deploy the service into the Web Services Gateway. Then you'll run a test to confirm that it was actually deployed.

Deploying the WSDL file

To deploy the WSDL:

1. Copy the file to a more permanent location, such as

   \bctws\was\cw\wsdl

   Important: The GeneratedFiles directory is likely to be over-written or erased. The reason you need to keep the WSDL file in a permanent place is that the Web Services Gateway holds a reference to it and does not make a copy of it. This means if you deploy it and then erase it, your service will be gone.

2. Deploy the WSDL that was generated by the WSGenUtility into the Public SOAP channel. This file is located in directory **<crossworlds>\DevelopmentKits\GeneratedFiles** and is named **BCT_SampleData_Create.wsdl**. To deploy the WSDL:
   a. Use a browser and navigate to the Gateway Service Deployment Screen.
   b. In the Gateway Service Name field, type:

      ```
      BCT_SampleData_Create
      ```

   c. In the Message part representation field, type:

      ```
      Generic classes
      ```

   d. In the Channels field, click: **ApacheSOAPChannel** to select it.
   e. In the WSDL Location field, type:

      ```
      c:/bctws/was/cw/wsdl/BCT_SampleData_Create.wsdl
      ```

f. In the Location type field, type: URL
g. Click **OK.**

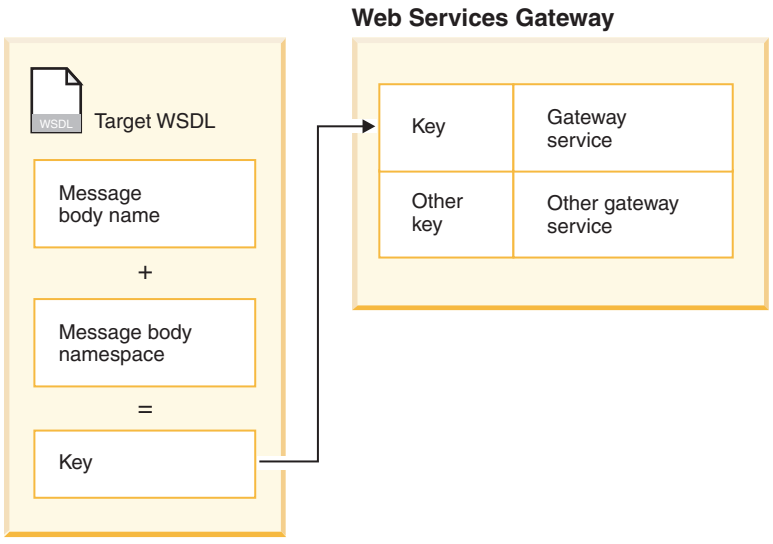
Changing the destination URL

After deploying the service in the Web Services Gateway SOAP channel, you can change the configuration file to direct the BCTSampleSOAPConnector to send its messages to the URL of the Web Services Gateway Channel rather than the URL of the CrossWorlds App Server.

To change the URL:

1. Open the \bctws\cw\bct_ws_configuration.txt file for edit.
2. Comment the line with the direct-to-CrossWorlds URL.
3. Add this line below it:
   BCT_WS_BCT_SampleData_Create_URL=http://localhost/wsgwsoap2
4. Save the changes.

Now messages whose generic business object is of type BCT_SampleData and whose verb is **Create** will be sent to the inbound channel of the Web Services Gateway. The Web Services Gateway will find the gateway service for the messages based on the message BodyName and BodyNS.



Once the gateway service is located, the target service will be invoked by WSIF using the reference to the original WSDL file, which specifies the URL of the CrossWorlds App Server.

Testing the change

To test the change you made:

1. Make sure both app servers are running and use the two Test Connectors as before.

   The fact that the Web Services Gateway is used is transparent to you because there is no security, logging, or routing service being performed by the Web Services Gateway.

2. Stop the **Web Services Gateway App Server.**

   Your test should fail at this point, which confirms that the Web Services Gateway was used in the service deployment.

If you want to use any of the logging or routing services, see Web Services Advanced Topics, which discusses the services. You can add these services to the tutorial yourself.

In Part III, you deployed the Web service you created into the Web Services Gateway. This completes the tutorial.

Now that you have seen how to build and deploy a web service, refer to Web Services Advanced Topics

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

WebSphere Business Connection Lab Director
IBM RTP Laboratory
3039 Cornwallis Road
P.O. BOX 12195

Raleigh, NC 27709-2195
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Programming interface information

Programming interface information is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM
alphaWorks
AIX
CrossWorlds
DB2
DB2 OLAP Server
DB2 Universal Database
DeveloperWorks
MQSeries
SecureWay
WebSphere

Lotus is a trademark of International Business Machines Corporation and Lotus Development Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.