



Hello World Part 1: Getting Started

Hello World Part 1: Getting Started

Note

Before using this information and the product it supports, read the information in “Notices” on page 25.

Contents

Chapter 1. Introduction 1

Overview 1

Concepts 2

Chapter 2. Build it yourself 5

Import the existing Web service into your workspace 5

Create a library project 6

Copy the Web service file 6

Create the new service interface 7

Create a mediation module project that references the library 9

Assemble the mediation module. 9

Create the mediation flow implementation 12

Chapter 3. Run the sample 17

Deploy the modules to the server 17

Test the module 19

Remove the module from the server 20

Chapter 4. Import 23

Notices 25

Terms of use 29

Chapter 1. Introduction

This sample shows you how to use WebSphere Integration Developer to create, deploy, and run a mediated call to an existing Web service.

In the sample, you learn how to complete the following activities:

- Navigate the workbench.
- Create a library project for shared artifacts.
- Create a mediation module project for integration logic.
- Invoke a Web service.
- Mediate a Web service call with a mediation flow, to map its interface to another one.
- Expose a mediation flow as a new service with a different interface.
- Deploy a mediation module.
- Run and test a mediation module.

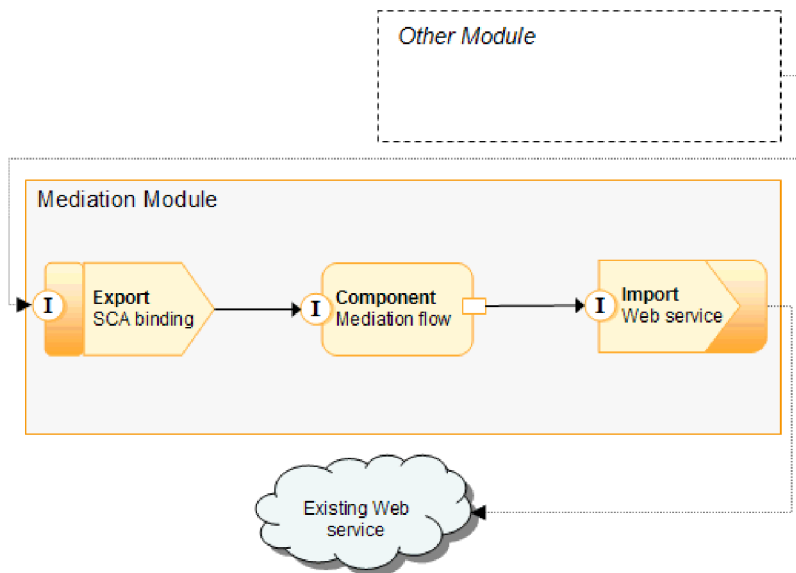
Overview

In this sample, your goal is to invoke an existing Web service by using a Web service import. This existing Web service has already been implemented and supplied for you, but you need to bring it into your workspace and deploy it to your server. It is implemented using a mediation module, but you need not know or care how it is implemented. You only need to know that there is a WSDL file defining not only its interface but where and how to invoke it with SOAP over HTTP.

Your objective is to allow other integration developers to invoke the Web service by calling your mediation module so they are not calling the Web service directly and so that you have the flexibility and resilience to change and evolve that Web service without impacting the clients that call it; all changes are absorbed in the mediation module. Furthermore, using a mediation module allows you to expose a different interface for that Web service, such as you will do in this scenario.

Because the interface that this sample exposes through an export with an SCA binding is different than the interface of the Web service it uses through an import with a Web service binding, you need to map the parameters (request) and what is returned (response) in a mediation flow component, with one flow for the request and another for the response. Mediation flows are built from mediation primitives that are wired together. Each primitive is a pre-supplied capability that acts on or processes the message flowing through it. Typically, the message contains the request and response parameters that are passed to or returned from an external call.

A high-level overview of this sample is shown in the following figure:



Concepts

WebSphere Integration Developer is a business integration product that enables you to create integration logic for invoking and exposing services, and to create business processes that integrate applications and data.

A mediation flow intercepts and modifies messages that are passed between existing services (providers) and clients (requesters) that want to use those services. Mediation flows can be implemented in modules or mediation modules. Mediation modules can be deployed to WebSphere® Enterprise Service Bus as well as WebSphere Process Server. In this sample, you use a mediation module so it is applicable to all WebSphere Integration Developer users.

A business process is a defined set of business activities that represent the steps required to achieve a business objective. Business processes can only be implemented in modules versus mediation modules, and can only be deployed to WebSphere Process Server. (The Hello World Part 2 sample introduces business processes.)

Modules and mediation modules in WebSphere Integration Developer are composed of components that call each other in a loosely coupled way. This loose coupling is achieved by each component declaring not only the interfaces by which it can be invoked, but also the interfaces of the components it wants to call or reference. By only defining the interfaces of these required components and not the actual components, it enables you to easily change the component used to satisfy those references.

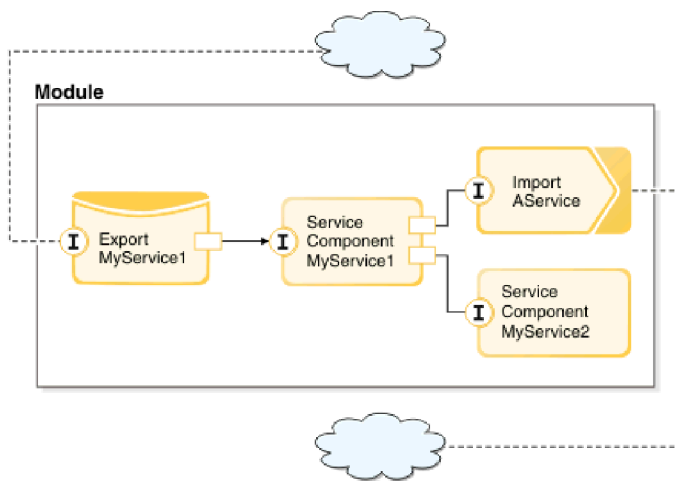
These components are implemented in a number of supported ways, such as with a mediation flow, or with a business process, or with Java. Each kind of implementation has its own editor, and in this sample you will be introduced to the mediation flow editor.

These components are defined and wired together within a module using the assembly editor. Defining a component means supplying it with a name, and identifying its interfaces and references, as well as its implementation type, and opening it in the appropriate editor to define its implementation. Wiring components means connecting one component's references with other components to satisfy its requirements.

To group together components for deployment to a particular server, you use a module. For a client application or another module to invoke a deployed module, you use other kinds of assembly diagram nodes called *exports* to export one of the components in the module so that it can be invoked remotely.

There are a number of options you can use to expose a component beyond its module boundary. You could expose a component as a Web service or by using a queuing technology such as Java Message Service. These options are called bindings. For module-to-module communication, there is an optimized option called an SCA binding. SCA (Service Component Architecture) is the standard upon which this loose-coupling capability is built. You will use an SCA binding in this sample, so that the module in the Hello World Part 2 sample can invoke the module created here in the Hello World Part 1 sample.

Sometimes components within modules also need to invoke existing services that are external to the module, and this is done through another kind of assembly diagram node called an *import* to import external services into the module so that they can be used to satisfy component references, just like any local component. The following figure shows a component MyService1 in the middle that has two references: one satisfied by an import of an external service, and the other satisfied by another local component. The first component is also exported so that other services that are external to the module can invoke the first component remotely, as shown in the following figure:



In this sample, you will use an import with a Web service binding to enable the mediation flow component to invoke an existing but external Web service that is supplied for you.

All components in WebSphere Integration Developer, including imports and exports, declare interfaces so other components or clients know to invoke them. Usually these interfaces are defined with the Web Services Description Language (WSDL), although Java components can also use Java interfaces. You use the interface editor to declare WSDL interfaces in WebSphere Integration Developer, and you can also import existing WSDL files into your library and module projects. In this sample, you will create a new interface and bring in an existing interface.

Interfaces contain one or more operations, each of which contains one or more input and output parameters, which use standard built-in data types or user-defined data types known as business objects. You will create a business object in this sample.

Chapter 2. Build it yourself

You can use the many tools of WebSphere Integration Developer to build the sample yourself.

To build the sample, you will complete the following steps:

1. Import the existing Web service into your workspace.
2. Create a library project.
3. Copy the Web service WSDL file. This is a concrete WSDL file for the supplied Web service that describes its interface and its SOAP binding.
4. Create the new service interface. In the library you will create a new business object and new interface.
5. Create a mediation module project that references the library.
6. Create an assembly with a mediation flow component. That component is exported with an SCA binding, and has a reference that is fulfilled by a Web Service import.
7. Create the mediation flow implementation, which includes mapping the request parameters between the exported and imported services, and the returned result.

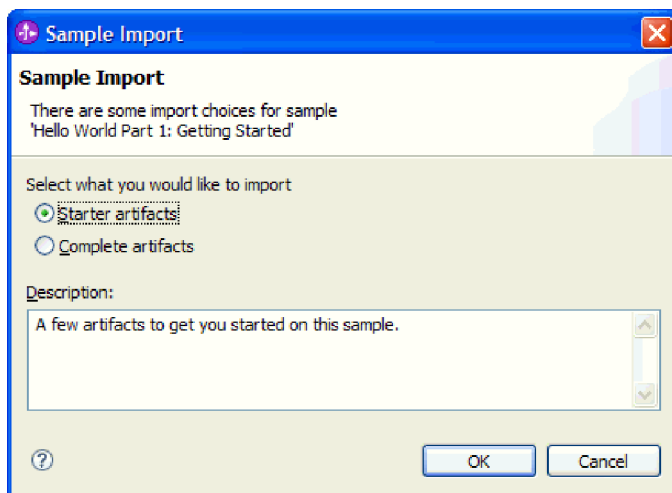
Import the existing Web service into your workspace

To build this sample, you need the HelloService sample Web service. This is pre-supplied for you and represents an existing Web service you want to call, which takes a string and returns “Hello string”.

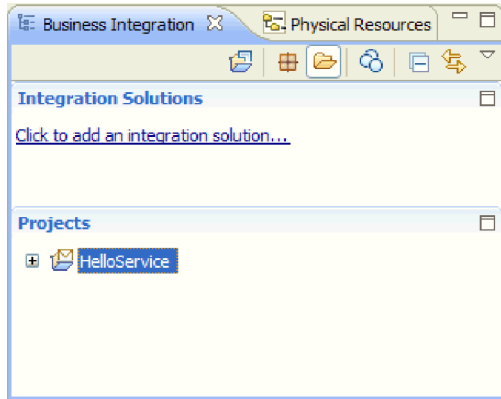
You do not need to know how the service is implemented, but you will notice it was done with a mediation module with a simple Java component that is exported with a Web service export binding.

To add the ready-made HelloService Web service implementation to your workspace:

1. Start with a fresh workspace. In WebSphere Integration Developer, ensure the Welcome page is open. If not, select **Help > Welcome** to open it.
2. Click **Samples and Tutorials**. The Samples and Tutorials page opens.
3. Under the **Hello World Part 1: Getting Started** section, click the **Import** link. You are presented with two possible modules to import, as shown here:



4. Select **Starter artifacts** and click **OK**. You should now see a single project named **HelloService** in your Business Integration view, as shown here:



5. Close the Welcome page by clicking on the **X** in its tab. Similarly close the Samples and Tutorials page.

Create a library project

In WebSphere Integration Developer, a library is a project where you can place files that are needed by more than one module. Because you need to eventually share your new service's interface and business object artifacts with the Hello World Part 2 sample, you need a library to hold them. Otherwise, you could create them directly into your module.

To create the library project:

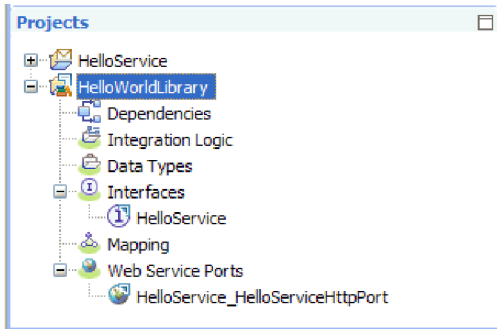
1. In the Business Integration view, right-click the **HelloService** project and select **New > Project > Library**. The New Library wizard opens.
2. In the **Library name** field, enter **HelloWorldLibrary** and click **Finish**. The library project appears in the Business Integration view.

Copy the Web service file

One use for the new library is to contain the endpoint WSDL file for the supplied Web service you want to invoke.

To copy the Web service file into the library:

1. In the Business Integration view, expand the **HelloService** project, and then the **Web Service Ports** category.
2. Right-click **HelloServiceExport_HelloServiceHttpPort** and select **Show Files**. This action gives focus to the Physical Resources view. This view shows the underlying files in your projects.
3. Right-click the file **HelloService_HelloService.wsdl**, and select **Copy**. This action places the WSDL file in the clipboard.
4. Return to the Business Integration view. Right-click **HelloWorldLibrary** and select **Paste**. This copies the WSDL file into your library project.
5. Collapse the **HelloService** project because you do not need to see it anymore, and expand the **HelloWorldLibrary** project. Then expand its **Interface** and **Web Service Ports** categories to see what has been copied in from that WSDL file, as shown here:



Note: When you have your own existing Web service, you can similarly copy it to a library or module project by using the clipboard or by dragging and dropping it from the Windows file system, or by using **File > Import > Business Integration > WSDL/Interface**. The latter option is suggested for more complicated WSDL files that include references to other files.

Note: You might be wondering what the annotation is on the upper right of the icons for the interface and port. This annotation indicates that the interface and port are part of a WSDL file that contains multiple objects that can be extracted by right-clicking and selecting **Refactor > Extract WSDL Components**. However, you do not need to perform that task in this sample.

Create the new service interface

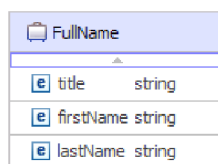
Your service will concatenate three input strings; namely, a *title*, a *first name*, and a *last name*. To hold these fields you need to create a business object and then an interface that takes one of these business objects as input and returns the concatenated string "Hello *title firstname lastname*".

To create the service interface:

1. In the Business Integration view, within the **HelloWorldLibrary**, right-click the **Data Types** category. Select **New > Business Object**. The New Business Object wizard opens.
2. In the **Name** field, enter **FullName** and click **Finish**. The business object editor opens.
3. To create a new field, click the little F button in the local toolbar (or right-click the **FullName** box and select **Add Field**), as shown in the following figure:



4. Type over the generated field's name of **field1** and replace it with **title**. If the name is not selected, then first click on it to select it.
5. Repeat the previous step to create two more fields; one named **firstName** and the other named **lastName**. The final business object should look like this:



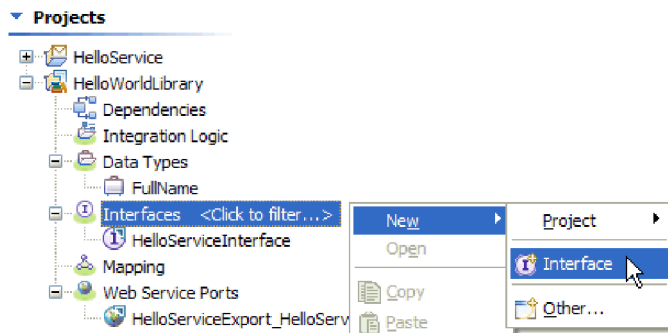
Optional: Select one of the string cells in the **Type** column. A list of types appears. Although you only need fields of type **string** for this sample, this is where you can specify other types. Press the **Esc** key to close the list.

Optional: Select a field and look at the Properties view below the editor. Although you do not need to set any of these fields for this sample, this is where you can specify certain properties for fields, such as specifying field repetition or maximum length.

6. Press **Ctrl-S** to save your work, and then close the business object editor.

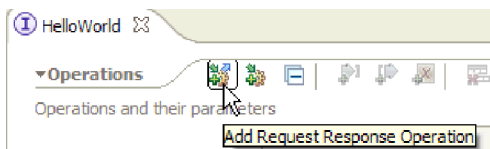
Optional: Under the covers, you have just created a new XSD or XML schema file with a complex type in it. If you are curious, you can see the file by right-clicking the business object and selecting **Open With > XML Schema Editor**, and then choosing the **Source** tab.

7. Back in the Business Integration view, within the **HelloWorldLibrary**, right-click the **Interfaces** category. Select **New > Interface**, as shown here:

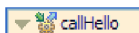


The New Interface wizard opens.

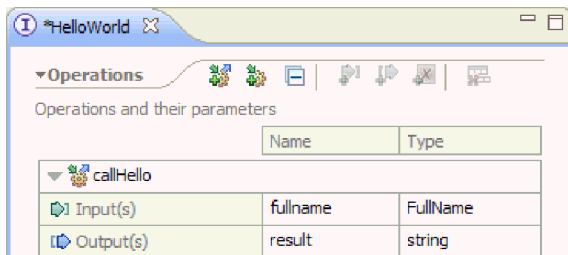
8. In the **Name** field, enter **HelloWorld** and click **Finish**. The interface editor opens.
9. To add a request response operation, click the **Add Request Response Operation** icon in the local toolbar, or right-click and select **Add Request Response Operation**.



10. Double-click the generated operation name **operation1** and type over it with **callHello** as shown here:



11. Double-click the generated parameter name **input1** and type over it with **fullname**.
12. Click in the type string cell of the table, in the **Input(s)** row, to change the type. In the pop-up list, scroll to the bottom and select **FullName**, which is the business object you recently created.
13. Double-click the generated parameter name **output1** and type over it with **result**. The interface should look like this:



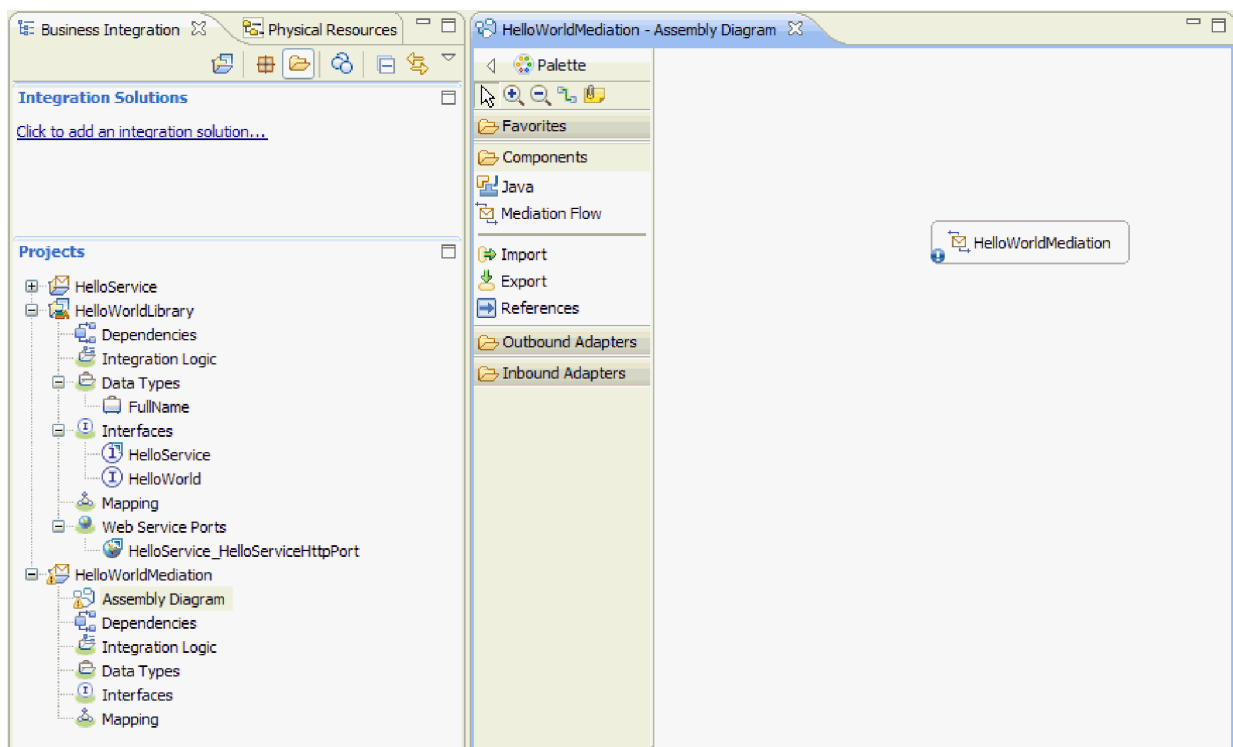
14. Save and close the interface editor.

Create a mediation module project that references the library

A mediation module is a project containing service integration logic and it can be deployed to either WebSphere Process Server or WebSphere Enterprise Service Bus.

You will use a mediation module to access the supplied **HelloService** Web service and expose it as a service to other modules. Because the interface of the service you invoke (HelloService) is different than the interface of the service you expose (HelloWorld), you will need to map between them.

1. Right-click the **HelloWorldLibrary** library and select **New > Project > Mediation Module**. The New Mediation Module wizard is displayed.
2. In the **Module name** field, enter HelloWorldMediation and click **Next**. The **Select required libraries** page shows.
3. Ensure that the **HelloWorldLibrary** project is selected. This selection associates the library with this new module so that this module can use any artifacts in that library. Click **Finish**. The assembly editor opens for the new module and it contains a mediation flow component, as shown here:



Assemble the mediation module

Next, you create an export that allows other modules to call the mediation flow component, and create an import that invokes the HelloWorld service. Wire together the export, mediation flow, and import to produce a deployable module.

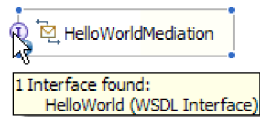
To assemble the mediation module:

1. The **HelloWorldMediation** component will ultimately be implemented by a mediation flow, but first you need to give it an interface so it can be invoked by other components. Select the component, and notice the hover bar that comes up above it, as shown here



Note: If you accidentally double-click on the component, an Open window will ask if you want to create the implementation. If the window opens, close it.

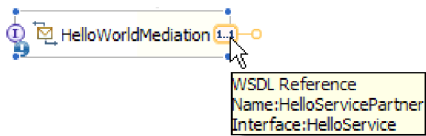
Select the circled **I** from that hover bar to create an interface (or right-click the component and select **Add > Interface**). The Add Interface window opens. Select **HelloWorld**, which is the interface that you recently created. Click **OK**. You will now see a circled **I** on the left edge of the component, and its hover help displays the interface type, as shown in the following figure:



2. This mediation can now be invoked, but you also need to identify what services it will invoke. Select the component again, and this time select the right arrow from the hover bar to add a reference, as shown in the following figure:



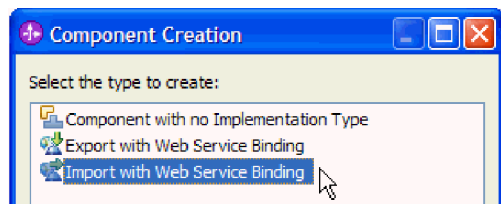
The **Add Reference** window opens. Select **HelloService**, which is the interface of the supplied service that will be invoked. Click **OK**. You will now see a little box on the right edge of the component, with “1..1” in it, which represents a required service that this component has declared it invokes, as shown in the following figure:



You have not yet identified the actual service to be invoked. At this point all you know is the interface or shape of that service, (The “1..1” is the *multiplicity* of the reference and it indicates that this reference must be satisfied with exactly one wire to another component. It is possible to configure the reference to allow multiple wires.)

This is the elegance of Service Component Architecture: regardless of the implementation details, applications are defined as a series of components that expose interfaces and consume other components or services through references.

3. Now you need to add an import component for the invocation of the supplied **HelloService** Web Service whose endpoint WSDL you copied into your library. In the Business Integration view, expand **HelloWorldLibrary**. In the **Web Service Ports** category, drag the port (HelloService_HelloServiceHttpPort) anywhere in the assembly diagram canvas. The Component Creation dialog opens, as shown here:



4. Select **Import with Web Service Binding**, and click **OK**. The Transport Selection window opens.
5. Select **SOAP1.1/HTTP for JAX-RPC** for the transport, and click **OK**. An import named **HelloServiceImport1** is created.
6. Click on the import name to enter edit mode, and change the name to **HelloServiceImport** (that is, delete the '1' suffix).

Note: If you accidentally double-click the component, an Open window will ask whether you want to create the implementation. If the window opens, close it.

7. Hover over the reference box on the right border of the **HelloWorldMediation** component until you see a yellow border and a yellow circle to the right, as shown in the following figure:



Grab the circle with your mouse and drag it to the circled **I** on the left edge of the **HelloServiceImport** component to wire the two components together, as shown in the figure below:



You have just resolved the reference of the first component with an actual provider of the service, which in this case is an external Web service. This means that when the implementation of the first component invokes that reference, it will really invoke the HelloService Web service. (Soon, you will create the implementation of the first component.)

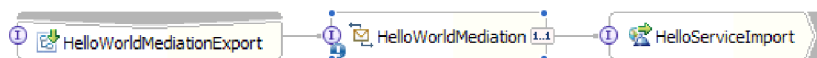
Note: A quick way to create a Web service import is by dragging **Import** from the palette to the assembly diagram and then configuring it. You would create an import this way to invoke other types of services, such as those invoked over the native SCA binding, or by sending a message over HTTP, JMS or MQ or by invoking a remote enterprise Java bean. Beyond these built-in import bindings, you can also use supplied adapters to invoke external services by way of the J2EE Connector Architecture (J2C) standard to do things like write to a file or send an email. The services accessible by some of the adapters are referred to as Enterprise Integration Services or EIS.

Optional: In the palette, expand the **Outbound Adapters** category and use the hover help to see what external services are available.

8. In addition to invoking a Web service, you also want to expose your mediation so it can be invoked from other modules in preparation for Hello World Part 2. Because this is the only client you need to support, you use the SCA export binding. (Only other modules can invoke an SCA export component by using an SCA import component in their own module). In the assembly diagram canvas, right-click the **HelloWorldMediation** component and select **Generate Export > SCA Binding**, as shown here:



This action creates a new export component with an SCA binding that is wired to the **HelloWorldMediation** component. Now your component can be accessed outside of this module.



Note: SCA is only one way to expose a component so that it can be invoked outside of the module. As you can see in the **Generate Export** cascading menu, others include HTTP, JMS, MQ and Web services. As with imports, these are all built-in bindings supported natively, but in addition there are supplied adapters for invoking a module using the J2C standard.

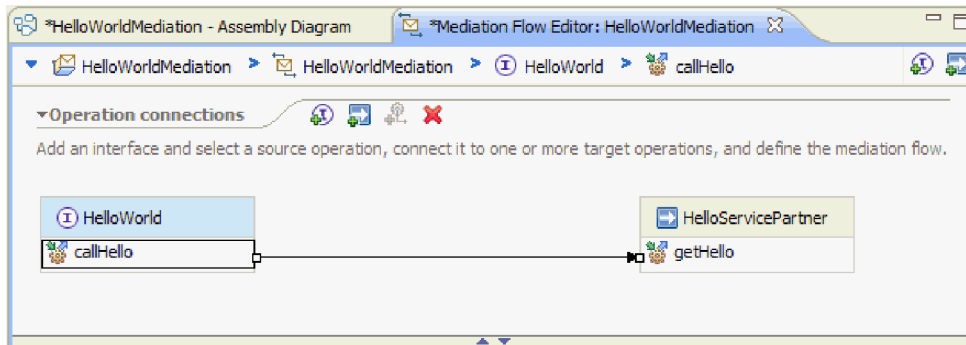
Optional: In the palette, expand the **Inbound Adapters** category and use the hover help to see what other ways of invoking the module are supported, such as receiving an e-mail or contents appearing in a flat file.

9. Press **Ctrl-S** to save your work in the assembly diagram.

Create the mediation flow implementation

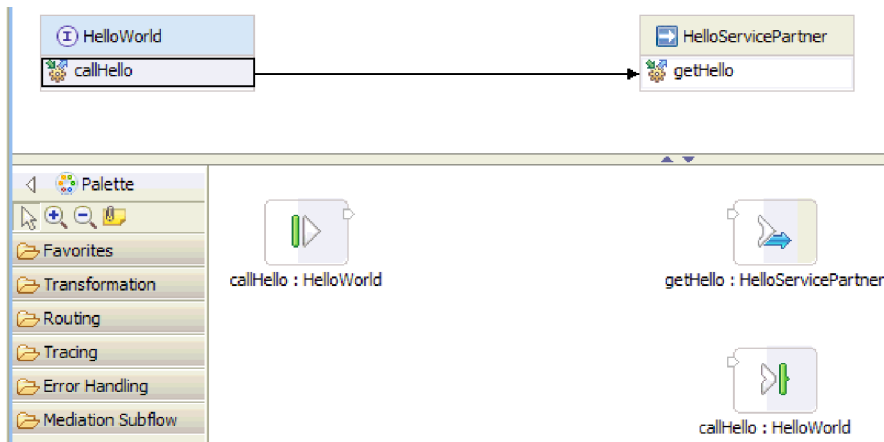
Finally, it is time to create the implementation for the HelloWorldMediation component. This component is a mediation flow and was created for you when you created the mediation module, although you could also create one by dragging a mediation flow from the palette. Because of the type of the component, you will use the mediation flow editor to implement it.

1. Double-click on the **HelloWorldMediation** component and click **Yes** in the Open window and click **OK** in the Generate Implementation window. The mediation flow editor opens.
2. At the top of the mediation flow editor, you see **callHello** on the left and **getHello** on the right. This is the single operation from this component's interface and the single operation from this component's reference, respectively. (Note that it is valid to have multiple interfaces and references for each component, and multiple operations for each interface and reference. But this *is* Hello World). Select the **callHello** interface operation and drag the yellow wire to the **getHello** reference operation, as shown here:



What you have indicated here is that you will be invoking the **getHello** operation as part of implementing the **callHello** operation for this component.

3. Next you want to implement the *flow* for the **callHello** operation. Because this is a request response operation, you ultimately need one flow for the request and another for the response, but you start first with the request. Click **callHello** to see the flow in the section below, as shown here:

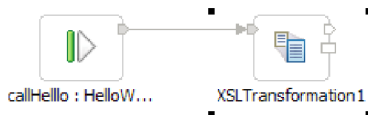


In the flow area, you see an input node on the left, which represents control reaching your request flow by way of the **callHello** operation being invoked. On the right, there are two output nodes: the top one, which represents calling out to the **getHello** reference operation and the bottom one that represents returning to the caller of this flow. If you wire to the top *callout* output node, you will invoke whatever that reference is wired to, which in this case is an external Web service. If you wire to the bottom *input response* output node, you will end the flow and send a response back to

whatever is wired to this component, which in this case is an SCA export component. However, typically the result is returned in the response flow versus this input response node.

The flow editor allows you to place *primitives* between the input and output nodes and wire them together, producing a flow upon which the *message* travels. The message is the parameter data for the input operation, which in this case is the payload for callHello – namely, a FullName business object.

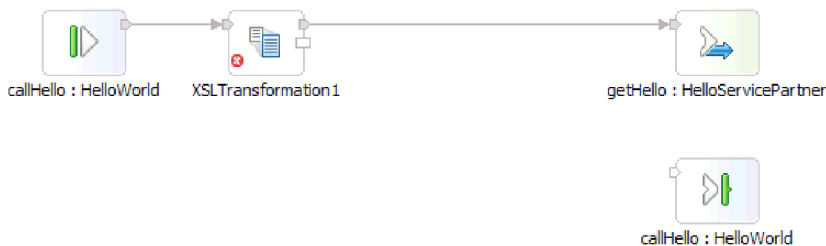
4. You want to wire the input node to the callout output node, but you cannot do so directly from the input node because the type of the message coming in is different than the type of the message going out. (That is, the parameters of the callHello input operation are different than the parameters of the getHello output operation.) So you need to map between the two. In the palette, expand **Transformation**, and drag the **XSL Transformation** primitive to the canvas.
5. Wire from the right edge of the input node's *output terminal* to the left edge of the **XSLTransformation1** primitive's *input terminal*, as shown here:



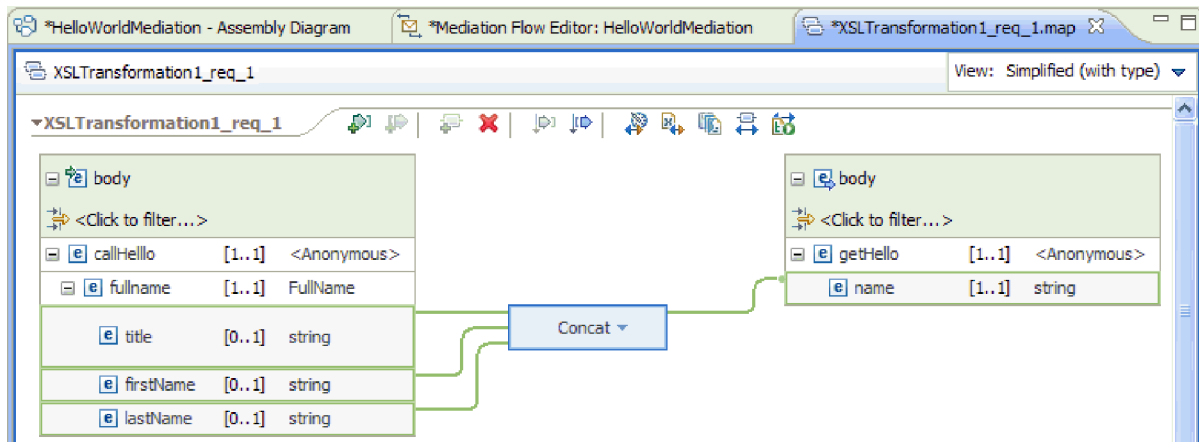
6. Similarly, wire the right edge of the topmost output terminal of the **XSLTransformation1** primitive to the input terminal of the callout output node (which is the right-most node).

Note: The bottom output terminal is the “fail” terminal, and normally you would wire it to primitives to handle the situation where this primitive failed for some reason. For example, if you received unexpected input, the map might fail.

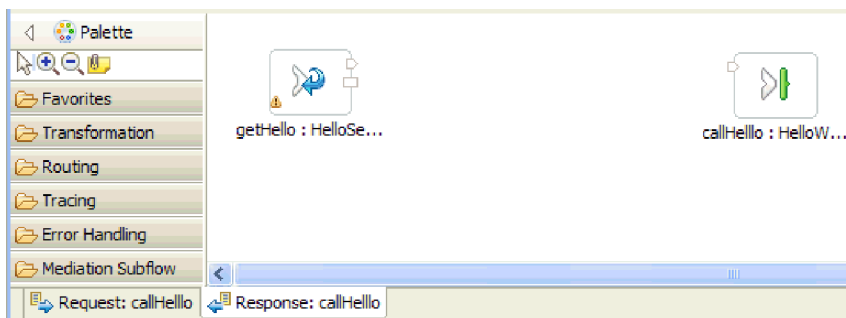
7. Press **Ctrl-S** to save your work. Note the red X on the **XSLTransformation1** primitive because you don't have a map defined yet, as shown in the following figure:



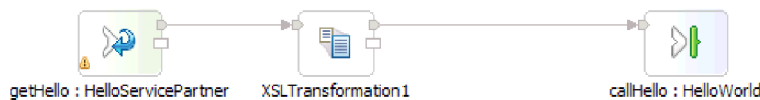
8. Double-click the **XSLTransformation1** primitive to create its map. The New XML Mapping window opens. Click **Finish**. The XML mapping editor opens.
9. In the XML mapping editor, fully expand the left and right trees. Wire **title** on the left to **name** on the right. This creates a **Move** map operation.
10. Wire **firstName** on the left to the **Move** operation in the middle. This changes the operation into a **Concat** operation.
11. Wire **lastName** on the left to the **Concat** operation in the middle. You now see three wires coming in, and one going out, as shown here:



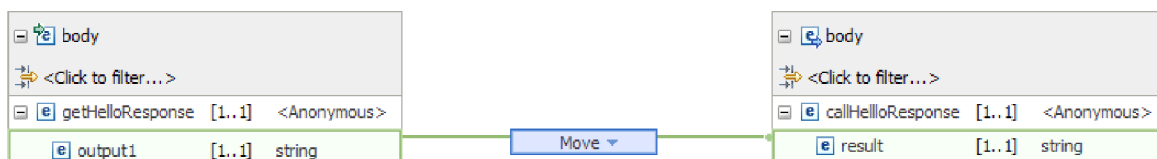
12. Select the **Concat** operation and then go to the **Properties** view and click the **General** tab.
13. In the table, click in the **Delimiter** cell for **title** and enter a blank. Press the **Enter** key after typing. Repeat for the **Delimiter** cell for **firstname**.
14. Save and close the XML mapping editor.
15. Save the mediation flow editor and notice that the error marker goes away.
16. Now you must wire the mediation *response* flow, to process the response from the Web service you invoked in the request flow and turn it into a response to the caller of this component. Select the **Response:callHello** tab at the bottom of the flow editor, as shown here:



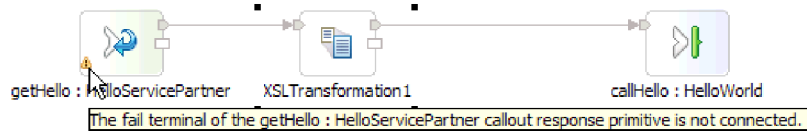
17. This time use a shortcut. Wire the input node on the left to the output node on the right. Because the types do not match, an XSL Transformation primitive is automatically inserted on the wire for you, as shown in the following figure:



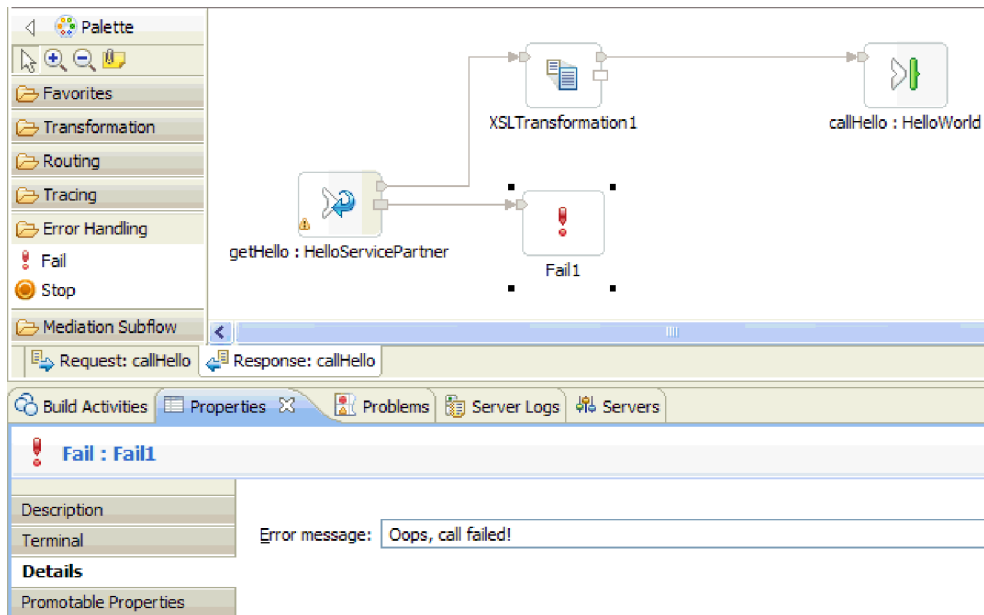
18. Double-click the **XSLTransformation1** primitive and create the map. Map the incoming **output1** field to the outgoing **result** field, as shown in the following figure:



19. Save and close the XML mapping editor.
20. Finally, you want to get rid of the warning that you have not wired the fail terminal for the callout response node, as shown in the following figure:



The warning occurs because you have not dealt with the case when the call to the Web service fails. Expand the **Error Handling** category in the palette, and drag the **Fail** primitive to the canvas, and then wire the **Callout Response** fail terminal to this primitive. Then from the context menu of the Fail primitive, select **Show in Properties**. Go to the **Details** property page and enter a string like **“Oops, call failed!”** to include in the fault that gets sent to the caller, as shown in the following figure:



21. Save and close the mediation flow editor and the assembly editor.

Congratulations – the authoring steps are done! Now it is time to test.

Chapter 3. Run the sample

After you have finished building the sample, you can run it.

To run the sample, complete the following steps:

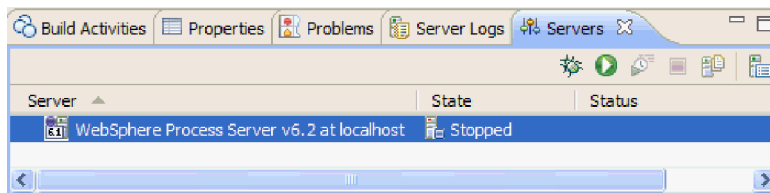
- Deploy the mediation module.
- Test the mediation module.

Deploy the modules to the server

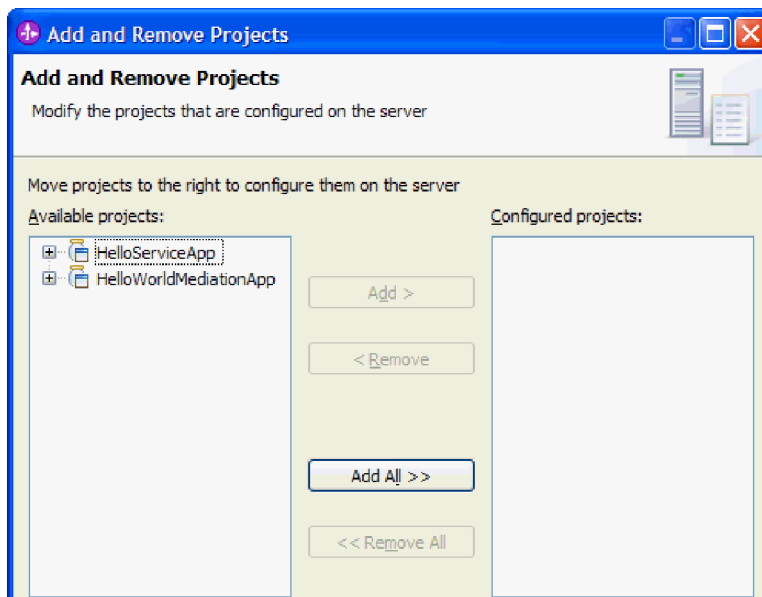
You must deploy (also referred to as *publish*) your modules to your test environment server before you can run and test your mediation.

To deploy the module to the server:

1. If you are in the Samples and Tutorials page of the Welcome, click the **Go to the Business Integration perspective** icon in the upper right corner of the page. The Business Integration perspective opens. This is where you perform most of your development tasks in WebSphere Integration Developer.
2. Click the **Servers** tab. The Servers view opens, as shown in the following figure:

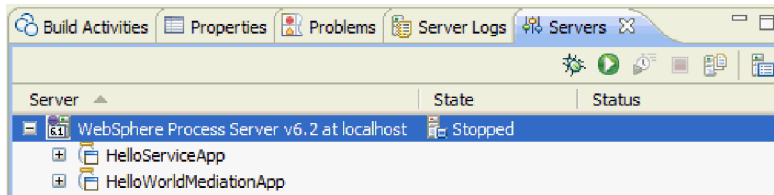


3. In the Servers view, right-click **WebSphere Process Server** or **WebSphere Enterprise Service Bus** and select **Add and Remove Projects**. The Add and Remove Projects window opens, as shown here:

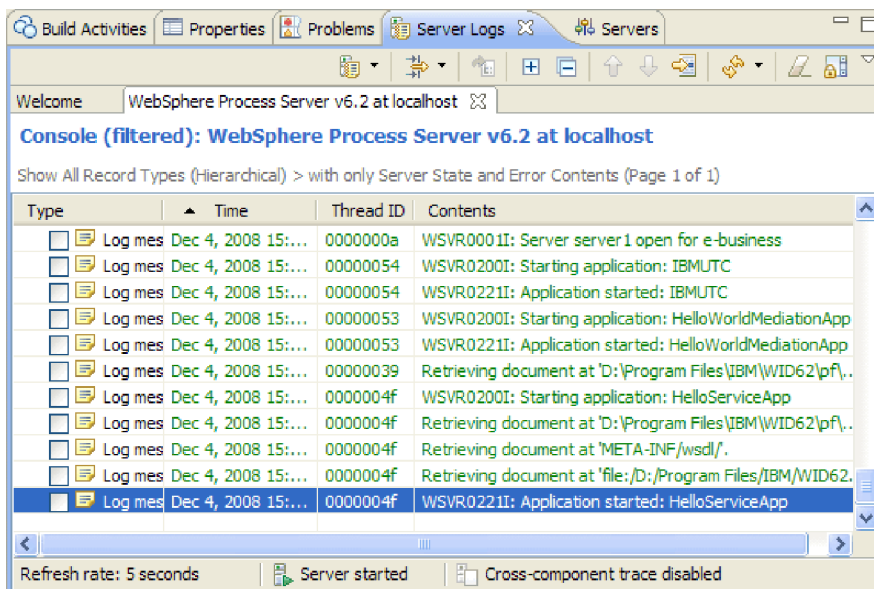


4. In the **Available projects** list, select the **HelloServiceApp** application.
5. Click **Add**. The **HelloServiceApp** application is added to the **Configured projects** list.
6. Similarly add the **HelloWorldMediationApp** application to the **Configured projects** list.

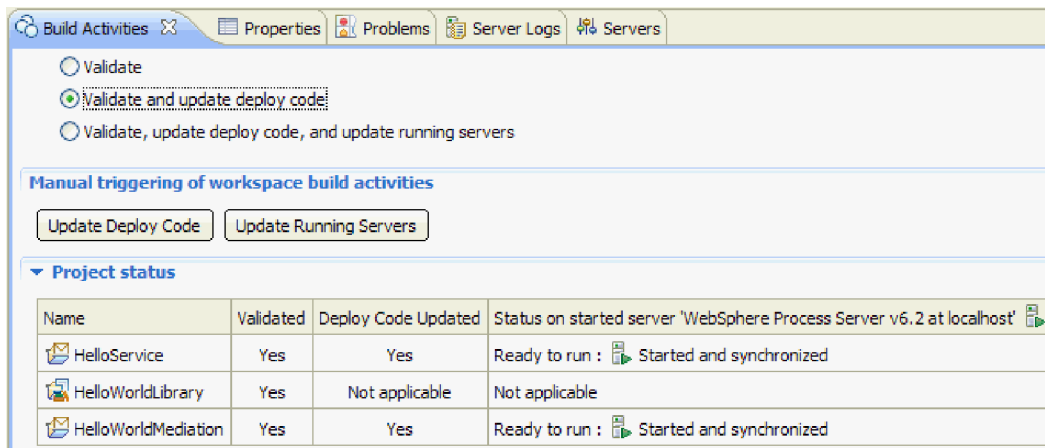
- Click **Finish**. The **HelloServiceApp** application and the **HelloWorldMediationApp** application now both appear under the expanded server in the **Servers** view, as shown in the following figure:



- If the **State** column in the **Servers** view shows this server is **Stopped**, right-click on the server and click **Start**. Wait until the Servers view shows a state of **Started**. This may take a few moments.
- Optional:** The Server Logs view shows the messages emitted by the server. Double-click the **Server Logs** tab to give it focus and expand it to full size. Scroll to the bottom to see the logged messages that the two applications have started. Also notice that the bottom status line shows that the server is started, as shown here:



- Optional:** The Build Activities view shows you the status of your projects, and after the initial association of a project with a server, it is a good place to do subsequent publishes after making changes. Click the **Build Activities** tab and expand **Project Status** to ensure your projects display a status of **Ready to run**, as shown in the following figure:



In the future, to quickly re-publish modules that have a status of **Started but requires republishing**, as they will after subsequently editing your library or module projects, you can simply click **Update Running Servers**.


Test the module

The next task is to run and test the module you just deployed. You will use the integration test client to test the module by giving it sample data and viewing the result.

To test the module:

1. In the Business Integration view, expand **HelloWorldMediation** and double-click **Assembly Diagram**. The assembly diagram opens.
2. Right click on the **HelloWorldMediationExport** export component and select **Test Component**. The integration test client opens.
3. In the **Value** column of the value editor table (located in the lower right corner of the test client), double-click a cell to enter edit mode and then enter Mr for **title**, Phil for **firstName**, and Bar for **lastName**. **Tip:** Click the down arrow and press the **Enter** key after typing. The value editor is shown in the following figure:

Name	Type	Value
fullname	FullName	✓
title	string	✓ Mr
firstName	string	✓ Phil
lastName	string	✓ Bar

4. At the top of the events list in the test client, click the **Continue** icon .
5. The Deployment Location dialog box opens. Ensure that your server is selected and click **Finish**.
6. The **User Login** window opens. If you did not change the default user ID and password of the server during installation, click **OK**. Otherwise, type the user ID and password that you specified during installation and click **OK**. The test client code that runs on the server is started, and if necessary any modules with changes are published, and the test is run. You see events in the **Events** list showing execution flowing through the components in the assembly diagram and fine-grained events of the execution flowing through the primitives in the mediation request and response flows. The result returned should be the string "Hello Mr Phil Bar", as shown here:

Events

- Invoke (HelloWorldMediationExport:callHello)
- Invoke started
- Binding (SCA:HelloWorldMediationExport)
- Invoke (HelloWorldMediationExport:callHello)
- Request (HelloWorldMediationExport --> HelloWorldMediation)
- Fine-Grained Trace (HelloWorldMediation:HelloWorld)
- callHello : HelloWorld
- XSLTransformationI
- getHello : HelloServicePartner
- Request (HelloWorldMediation --> HelloService)
- Response (HelloWorldMediation <-- HelloService)
- Fine-Grained Trace (HelloWorldMediation:HelloWorld)
- getHello : HelloServicePartner
- XSLTransformationI
- callHello : HelloWorld
- Response (HelloWorldMediationExport <-- HelloWorldMediation)
- Return (HelloWorldMediationExport:callHello)
- Binding (SCA:HelloWorldMediationExport)
- Invoke returned

General Properties

Detailed Properties

Module: [HelloWorldMediation](#)

Component: [HelloWorldMediationExport](#)

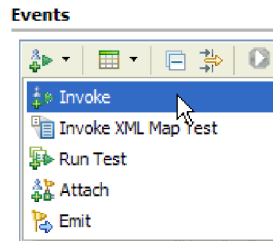
Interface: [HelloWorld](#)

Operation: [callHello](#)

Return parameters:

Name	Type	Value
result	string	✓ Hello Mr Phil Bar

7. Optional: You can continue testing. Select the little down arrow icon beside the first icon in the toolbar above the Events list and then select **Invoke**, as shown here:



A new Invoke event appears in the events list, and the original input data for that test shows in the **Initial request parameters** value editor. Change Bar to BarAgain and rerun the test, again by clicking the **Continue** button.

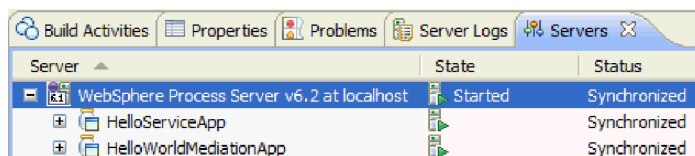
8. Use **File > Close All** to close all open editors. When prompted to save your test client session, click **No**.

Remove the module from the server

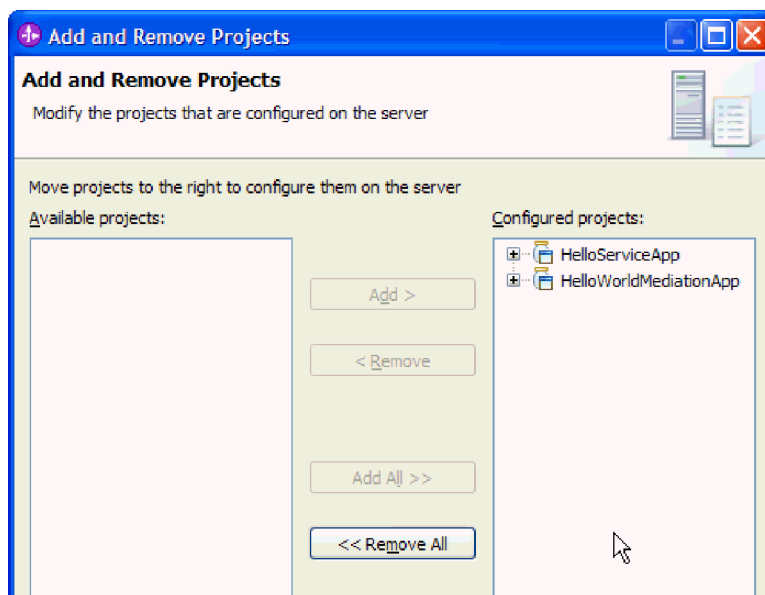
Generally, when you have finished testing a module, you should remove it from the server. This will ensure that the only modules that are deployed to the server are those that you are preparing to test, which will reduce the load on the server and enhance its performance.

To remove the module from the server:

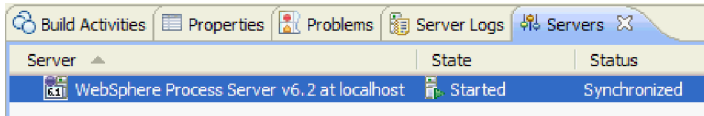
1. Click the **Servers** tab. The Servers view opens, as shown in the following figure:



2. In the Servers view, right-click **WebSphere Process Server** and select **Add and Remove Projects**. The Add and Remove Projects dialog box opens, as shown in the following figure:



3. Click **Remove All**. The applications are removed from the Configured projects list.
4. Click **Finish**. When a dialog informs you that the project is being removed from the server, click **OK**. The applications no longer appear under the server in the Servers view, as shown here:



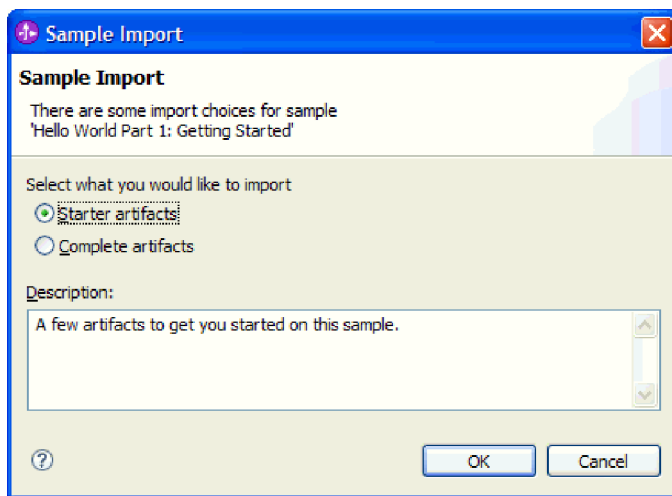
Congratulations! You have completed the Hello World Part 1: Getting Started sample.

Chapter 4. Import

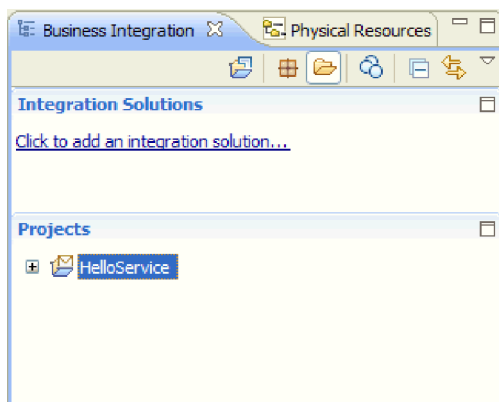
You can either import a complete ready-made version of the Hello World Part 1: Getting Started sample, or you can import starter artifacts and build the sample yourself.

To import the sample:

1. Open WebSphere Integration Developer and select a new workspace.
2. Select **Help > Welcome** to open the product Welcome.
3. Click the **Samples / Tutorials** icon. The Samples / Tutorials page opens.
4. Under the **Hello World Part 1: Getting Started** section, click the **Import** link. You are presented with two options, as shown here:



5. If you want to build the sample yourself, select **Starter artifacts** and click **OK**. You should now see a single project named HelloService in your Business Integration view, as shown here:



Open the "Build it yourself" instructions and begin with the topic "Create a library project".

6. If you want to import the complete ready-made sample, select the option **Complete artifacts** and click **OK**. You will see the following projects in the Business Integration view:
 - A mediation module named HelloServices.
 - A mediation module named HelloWorldMediation.
 - A library named HelloWorldLibrary.

Instructions for running the sample are found in the topic "Run the Sample".

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other 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 documentation does not grant 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.*

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

*IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan*

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 the 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:

*Intellectual Property Dept. for WebSphere Software
IBM Corporation
3600 Steeles Ave. East
Markham, Ontario
Canada L3R 9Z7*

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 measurements 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.

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

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples 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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate 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.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, 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

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

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

Linux is a registered trademark of Linus Torvalds 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.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Terms of use

Permissions for the use of publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, non commercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

© Copyright IBM Corporation 2005, 2008. All Rights Reserved.

Readers' Comments — We'd Like to Hear from You

Integration Developer
Version 6.2
Hello World Part 1: Getting Started
Version 6 Release 2

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Send your comments to the address on the reverse side of this form.

If you would like a response from IBM, please fill in the following information:

Name

Address

Company or Organization

Phone No.

E-mail address



Cut or Fold
Along Line

Fold and Tape

Please do not staple

Fold and Tape

PLACE
POSTAGE
STAMP
HERE

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

Fold and Tape

Please do not staple

Fold and Tape

Cut or Fold
Along Line



Printed in Canada