IBM WEBSPHERE 5.0 SKILL TRANSFER - LAB EXERCISE

Building a Business Process with Faults

What This Exercise is About

Business Processes can handle exception situations (such as no seats available when trying to make a flight reservation) by different means. Through Faults, Business Processes have a very convenient and flexible way to handle the exception situations. Faults are comparable to Java exceptions except at a Business Process level. WebSphere Studio Application Developer Integration Edition provides the wizards and tools to work with faults and to specify if and how they should be handled. This exercise will guide you through creating a Business Process which contains Fault terminals and Fault nodes, handling Business Process exceptions.

User Requirement

Windows 2000 Professional Service Pack 2 is required for this lab exercise. IBM WebSphere Studio Application Developer Integration Edition v5.0 (beta) is also required. WebSphere Application Server v5 Enterprise will also need to be installed. The lab source files (eelabfiles50.zip) must be extracted to the root directory (i.e., C:\). Experience with previous versions of WebSphere Studio Application Developer Integration is required. Familiarity with Web Services and Service-Oriented Architectures will also be helpful.

What You Should Be Able to Do

During this lab you should be able to create a simple Business Process based on independent Services. The Business Process will be built using WebSphere Studio Application Developer Integration Edition. Within Integration Edition, you will start by importing in the different WSDL files which describe the Services. Next you will connect the different Services together to create a Business Process and, more importantly, you will add Fault activities to cope with exception situations which may occur from the Services. Finally you will deploy the Business Process , install in WebSphere Application Server v5 Enterprise, and test the application.

Introduction

In this exercise you will build a simple Business Process which can be called stand alone or be part of a larger Business Processes. The Business Process is focused on working with exceptional situations (faults). This scenario is common and very generic and could be applied to many situations. This Business Process will be created in the context of a travel application. As reservations are made for a flight or a hotel, problems may occur preventing the reservation from being completed. The two services FlightEJBService and HotelEJBService raise faults should an incorrect flight destination be specified such as the mythical city of Atlantis or the requested hotel, such as AcmeHotel, is not on the preferred list. You have two basic means of handling a fault, either performing a correctional activity and continue with the Business Process or terminating the process and passing the information related to the exception onto another service. In this application, should AcmeHotel be specified, you will handle the Fault and continue the Business Process not interrupting the flight reservation as a different hotel can be specified without necessarily cancelling the flight reservation. Should an incorrect destination such as the Atlantis be specified, the complete Business Process will end as a hotel room will not be able to be reserved in the mythical city.

Exercise Instructions

**** NOTE ** Solution instructions are at the end. The solution is provided in case you do not wish to perform the exercise steps, or are unable to complete the lab and want to see the final solution. If you intend to go through the lab, start at Part One -- assuming you have met the requirements in the section “User Requirements” stated above.**

Part One: Lab Setup

__1. Start WebSphere Studio Application Developer Integration Edition.

__ a. From the Windows Taskbar, select the menu **Start** > **Programs** > **IBM WebSphere Studio** > **Application Developer Integration Edition** .

__ b. A dialog box will be displayed allowing you to select the location where you would like the workspace directory to be stored. Enter **c:\eelabfiles50\BPFaultlab\workspace** for the location and select **OK**.

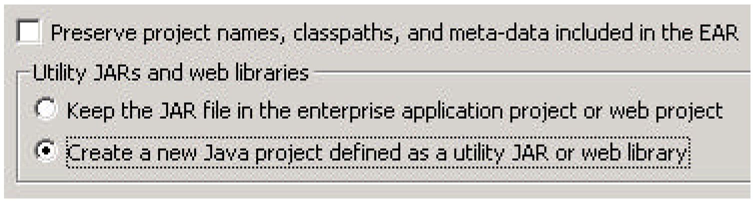
__2. Import the application MyTravelBP into Integration Edition. This application will be where you build your simple Business Process.

__ a. Select **File > Import** and select **EAR file** before selecting **Next**.

__ b. For the EAR File, select Browse... And select **c:\eelabfiles50\bpflaultlab\MyTravelBP.ear**. Select **OK**.

__ c. For the Project name, enter **MyTravelBPEAR**. Select **Next**.

__ d. In the Import Defaults window, select the radio button **Create a new Java project defined as a utility JAR or web Library**.



☐ Preserve project names, classpaths, and meta-data included in the EAR

Utility JARs and web libraries

○ Keep the JAR file in the enterprise application project or web project

● Create a new Java project defined as a utility JAR or web library

__ e. Select **Finish**.

Note: A number of items will appear in the Task view relating to unresolved references within the projects . These items are specific to your development environment and will be fixed in later steps.

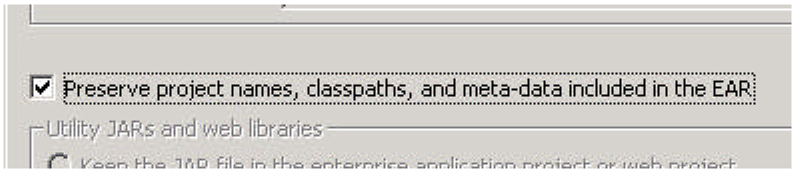
__3. Your Business Process calls a number of Web Services as the Business Process is executed. These Web Services are contained in a J2EE application which can run on WebSphere Application Server and needs to be imported into Integration Edition. The key point to note about this application is the Web Services provided and not the J2EE implementation. This application should be considered as a "black box" as the Business Process application only uses the Web Service interface when completing a Business Process. The application is only required should you want to test your Business Process within the Test Environment.

__ a. Select **File > Import** and select **EAR file** before selecting **Next**.

__ b. For the EAR File, select Browse... And select **c:\labfiles50\bpfaultlab\MyTravel.ear**. Select **Open**.

__ c. For the Project name, enter **MyTravel**. Select **Next**.

__ d. Since you will be treating this application as a "black box", select **Preserve project names, classpaths, and meta-data included in the EAR**. This setting will keep the different modules in a single JAR or WAR file and keep the implementation details hidden, enforcing the idea of a "black box".



☑ Preserve project names, classpaths, and meta-data included in the EAR

Utility JARs and web libraries

__ e. Select **Finish**.

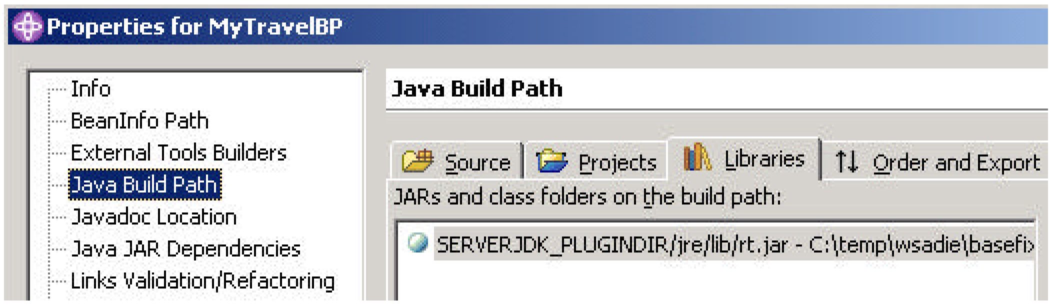
__4. You will need to update the Java Build Path to solve the unresolved references existing within the projects.

Note: These steps are only necessary within the beta version of Integration Edition and should be fixed by General Availability.

__ a. In the Business Integration perspective, select the **Package Explorer** view.

__ b. The Resource perspective will be open. Right-click on **MyTravelBP** and select **Properties**.

__ c. Select **Java Build Path** and select the **Libraries** Tab.



JARs and class folders on the build path:

SERVERJDK_PLUGINDIR/jre/lib/rt.jar - C:\temp\wsadie\basefi

__ d. Select the button **Add Variables...** and scroll down and select **WAS_50_PLUGINDIR**. Select the **Extend...**.

__ e. In the Variable Extensions window, expand **lib**. Using the Control key select the multiple files listed below.

commons-logging-api.jar

flow.jar

j2ee.jar

qname.jar

wsadiebp.jar

wsatlib.jar

wsdl4j.jar

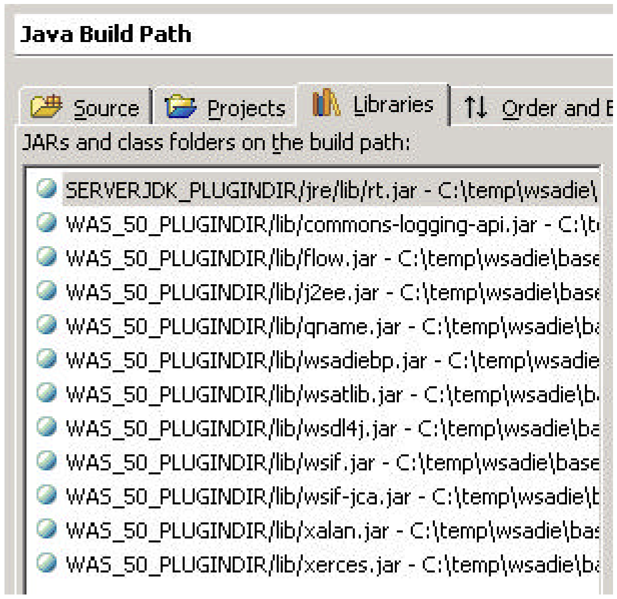
wsif.jar

wsif-jca.jar

xalan.jar

xerces.jar

__ f. Select **OK**.



JARs and class folders on the build path:

```
SERVERJDK_PLUGINDIR/jre/lib/rt.jar - C:\temp\wsadie\
WAS_50_PLUGINDIR/lib/commons-logging-api.jar - C:\t
WAS_50_PLUGINDIR/lib/flow.jar - C:\temp\wsadie\base
WAS_50_PLUGINDIR/lib/j2ee.jar - C:\temp\wsadie\base
WAS_50_PLUGINDIR/lib/qname.jar - C:\temp\wsadie\ba
WAS_50_PLUGINDIR/lib/wsadiebp.jar - C:\temp\wsadie
WAS_50_PLUGINDIR/lib/wsatlib.jar - C:\temp\wsadie\b
WAS_50_PLUGINDIR/lib/wsdl4j.jar - C:\temp\wsadie\ba
WAS_50_PLUGINDIR/lib/wsif.jar - C:\temp\wsadie\base
WAS_50_PLUGINDIR/lib/wsif-jca.jar - C:\temp\wsadie\b
WAS_50_PLUGINDIR/lib/xalan.jar - C:\temp\wsadie\bas
WAS_50_PLUGINDIR/lib/xerces.jar - C:\temp\wsadie\ba
```

__ g. Select **OK** to close the Properties window. The workbench will be rebuilt.

__ h. Repeat the same steps for the **MyTravelBPEJB** project.

Note: There will be a number of warnings still in the Task view. These can be ignored and are specific to the beta version of Integration Edition and should be fixed by General Availability

Part Two: Preparing to build a Business Process

__1. Import the dependent service interfaces.

In this step, you will import the WSDL files for the services for reserving flights and reserving hotel rooms. These Services are described by WSDL documents. (If these services were published, these WSDL files could also be obtained through a UDDI registry.)

__ a. From the main menu, select **File** > **Import**.

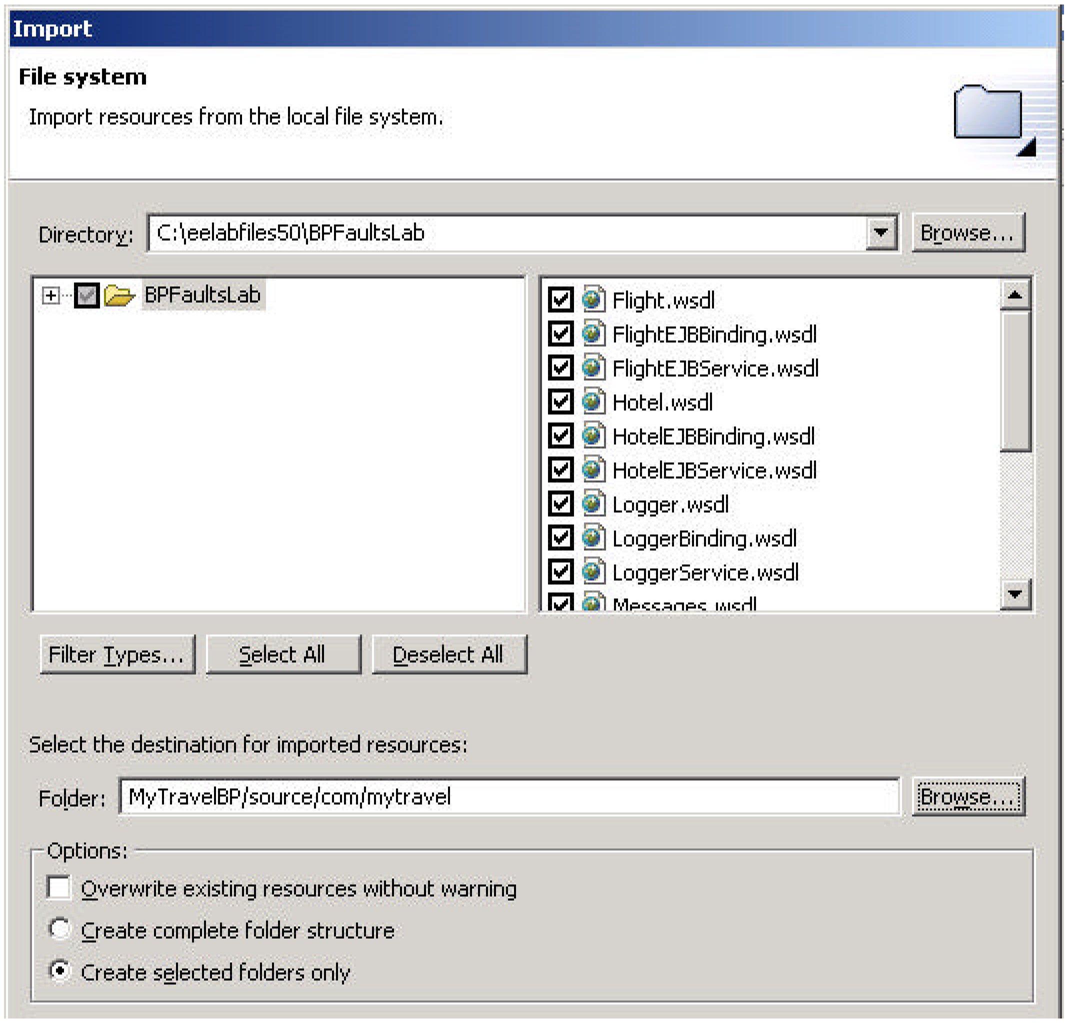
__ b. When presented with the **Import** wizard, select **File system**. Click the **Next** button to proceed.

__ c. When presented with the **Import** panel, click the **Browse** button that is located to the right of the **Directory** input field.

__ d. When presented with the **Browse for Folder** navigation window, navigate to and select the **C:\eelabfiles\BPFaultLab** subdirectory. Click the **OK** button to proceed.

__ e. From the **Import** panel, on the right-hand side of the panel, select the **Flight.wsdl**, **FlightEJBBinding.wsdl**, **FlightEJBService.wsdl**, **Hotel.wsdl**, **HotelEJBBinding.wsdl**, **HotelEJBService.wsdl**, **Logger.wsdl, LoggerBinding.wsdl, LoggerService.wsdl, messages.wsdl**, and **mybookingInterface.wsdl** check-boxes.

__ f. In the destination **Folder:** input field, enter "**MyTravelBP/source/com/mytravel**".
Select **Finish** to proceed.


Import

File system

Import resources from the local file system.

Directory: C:\eelabfiles50\BPFaultsLab

BPFaultsLab

- ☑ Flight.wsdl
- ☑ FlightEJBBinding.wsdl
- ☑ FlightEJBService.wsdl
- ☑ Hotel.wsdl
- ☑ HotelEJBBinding.wsdl
- ☑ HotelEJBService.wsdl
- ☑ Logger.wsdl
- ☑ LoggerBinding.wsdl
- ☑ LoggerService.wsdl
- ☑ Messages.wsdl

Select the destination for imported resources:

Folder: MyTravelBP/source/com/mytravel

Options:

- ☐ Overwrite existing resources without warning
- ○ Create complete folder structure
- ● Create selected folders only

__2. It is worth spending a few moments to discuss and inspect some of the files that you just imported. The WSDL files describe the different Services which will make up our Business Process.

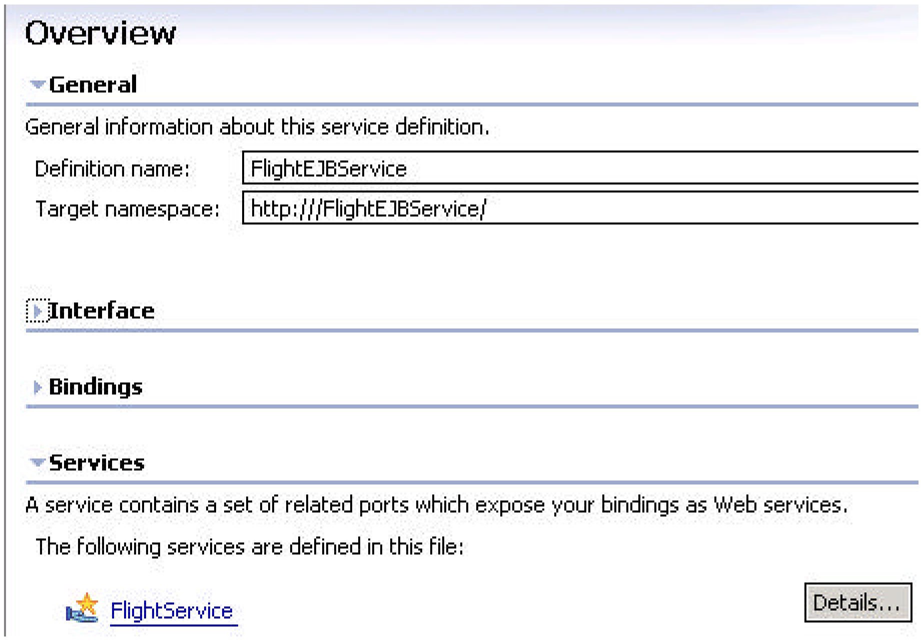
__ a. The Flight and Hotel files are for the flight and hotel services respectively. The implementation of these services are actually Enterprise beans.

These artifacts could be used directly as Enterprise beans within your Business Process rather than through the Service-Oriented Architecture approach. With the Service-Oriented approach, additional options are available such as handling Fault situations and specifying Compensation services for "undoing" a completed service. (Compensation is not covered in this lab exercise).

__ b. From the **Services** view of the **Business Integration** perspective, expand **MyTravelBP/source/com.mytravel** folder.

__ c. Open **FlightEJBService.wsdl**.

__ d. The Overview tab will be displayed. Under the Services section, the FlightService is listed. Select **FlightService**.


Overview

General

General information about this service definition.

Definition name: FlightEJBService

Target namespace: http:///FlightEJBService/

Interface

Bindings

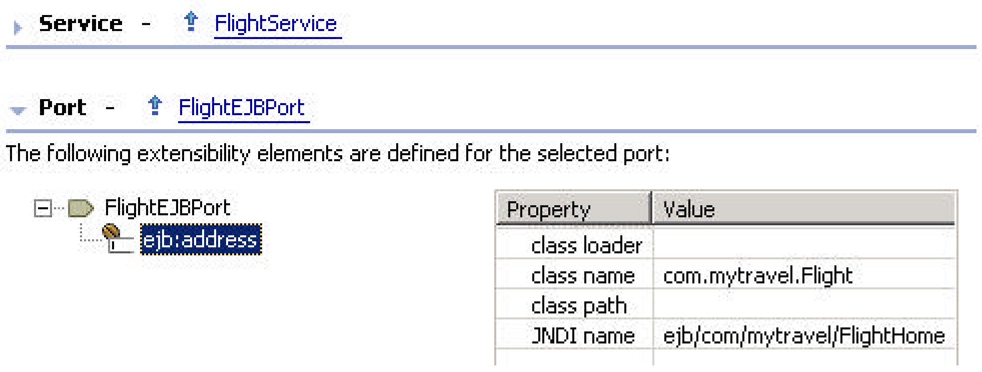
Services

A service contains a set of related ports which expose your bindings as Web services.

The following services are defined in this file:

FlightService

__ e. The Services tab will be displayed. Scroll down and expand the **Port** section. Under FlightEJBPort, select ejb:address. Notice the Enterprise bean service is accessed through its JNDI name **ejb/com/mytravel/FlightHome**.


Service - FlightService

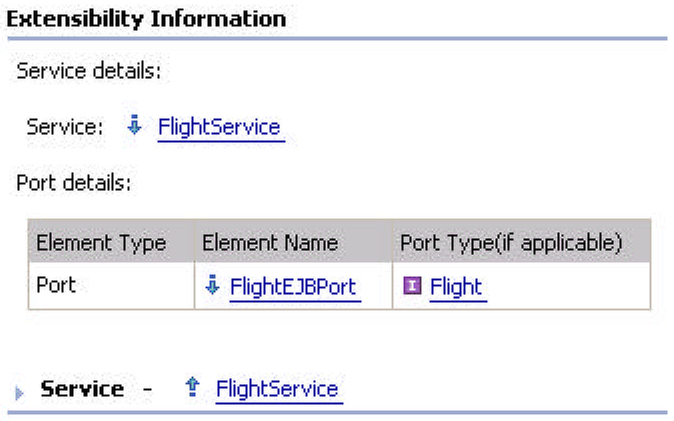
Port - FlightEJBPort

The following extensibility elements are defined for the selected port:

FlightEJBPort
ejb:address

Property	Value
class loader	
class name	com.mytravel.Flight
class path	
JNDI name	ejb/com/mytravel/FlightHome

__ f. Move up to the Extensibility Information section. Under the Port details section select the Port Type **Flight**.



Extensibility Information

Service details:

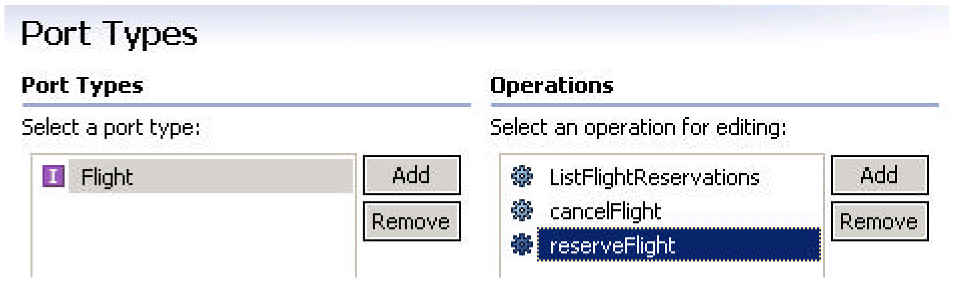
Service: FlightService

Port details:

Element Type	Element Name	Port Type(if applicable)
Port	FlightEJBPort	Flight

Service - FlightService

__ g. The **Flight.wsdl** file will be opened. This file has the interface information (operations, messages, parts, etc.) for the Flight reservation service. The Port Types tab lists the operations for the Flight Port Type. Select the **reserveFlight** operation.



Port Types

Port Types

Select a port type:

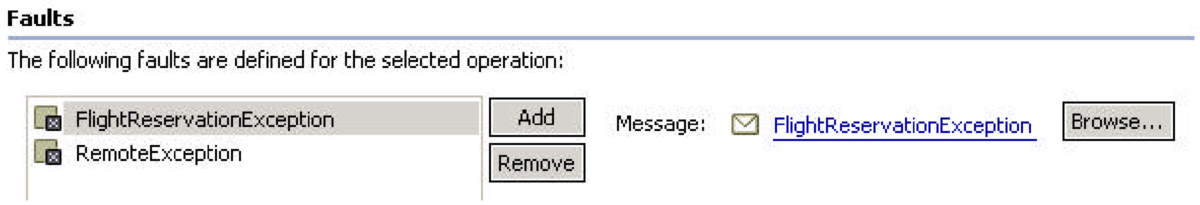
Flight

Operations

Select an operation for editing:

ListFlightReservations
cancelFlight
reserveFlight

__ h. Scroll down to the Faults section. Notice there are two Faults which can occur for this operation, **FlightReservationException** and **RemoteException**. Each Fault has a corresponding message which would be returned should the Fault occur. The message for each Fault will be used when you create the Business Process and handle the Faults in the Business Process.



Faults

The following faults are defined for the selected operation:

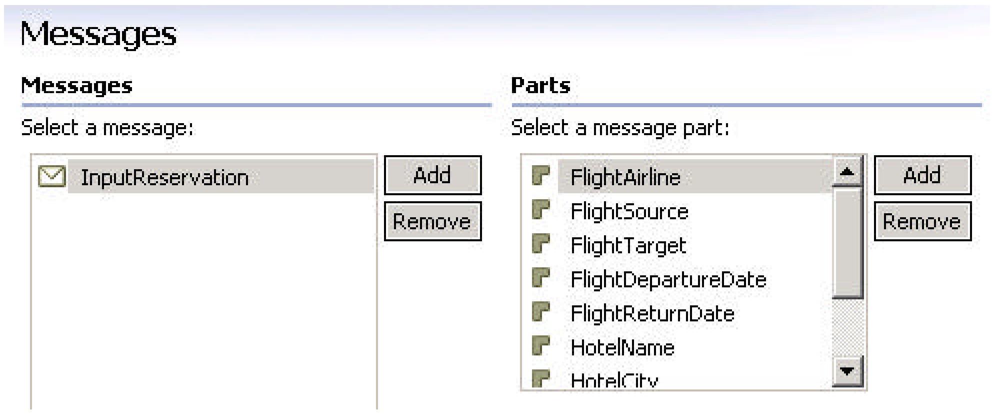
FlightReservationException
RemoteException

Message: FlightReservationException

__ i. Close the Flight.wsdl and FlightEJBService.wsdl file. You can also view the Faults for the Hotel Service by opening the HotelEJBService.wsdl and the Hotel.wsdl files.

__ j. The Logger files (**Logger.wsdl**, **LoggerBinding.wsdl**, and **LoggerService.wsdl**) describe a service which is used for logging status messages. This service will be used to record the completion of a reservation as well as failures based on Remote Exceptions and Flight Reservation Exceptions.

__ k. The **mybookingInterace.wsdl** and **Messages.wsdl** files describe a service interface and will be used as the interface for your Business Process. Open the **Messages.wsdl** file and select the **Messages** tab. For the interface, there is a single message, **InputReservation**, which is the input message. Notice the different parts which make up the message.

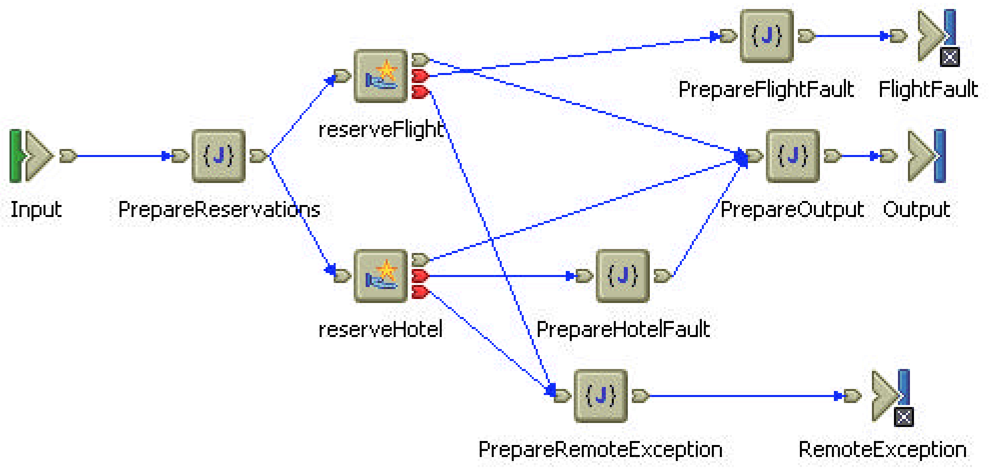


__ l. Close the Messages.wsdl file.

Part Three: Building a Business Process

__1. Create a new Business Process.

In this step, you will be creating a Business Process named mybooking.process that eventually will look like this:



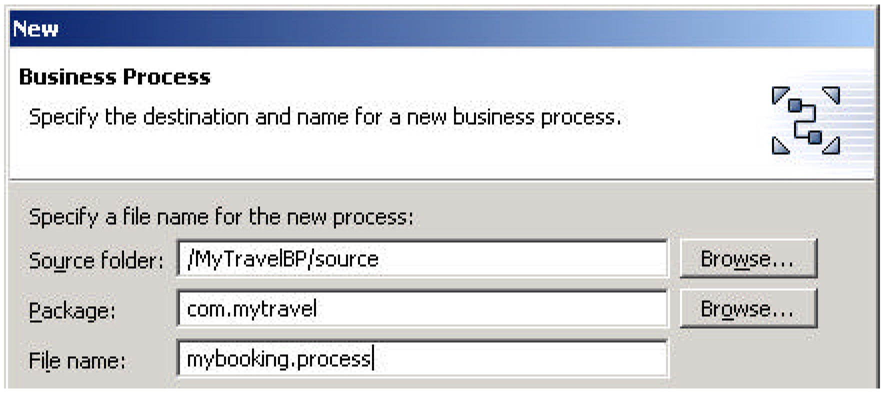
__ a. From the menu, select **File** > **New** > **Business Process**.

__ b. When presented with the **Business Process** panel, select the **Browse** button located to the right of the **Source folder** input field.

__ c. In the Folder Selection dialog-box, select the **MyTravelBP/source** folder. Select **OK**.

__ d. On the Business Process panel, for the Package, enter **com.mytravel** as the package name.

__ e. On the **Business Process** panel, enter a File name of **mybooking.process**.


New

Business Process

Specify the destination and name for a new business process.

Specify a file name for the new process:

Source folder: /MyTravelBP/source

Package: com.mytravel

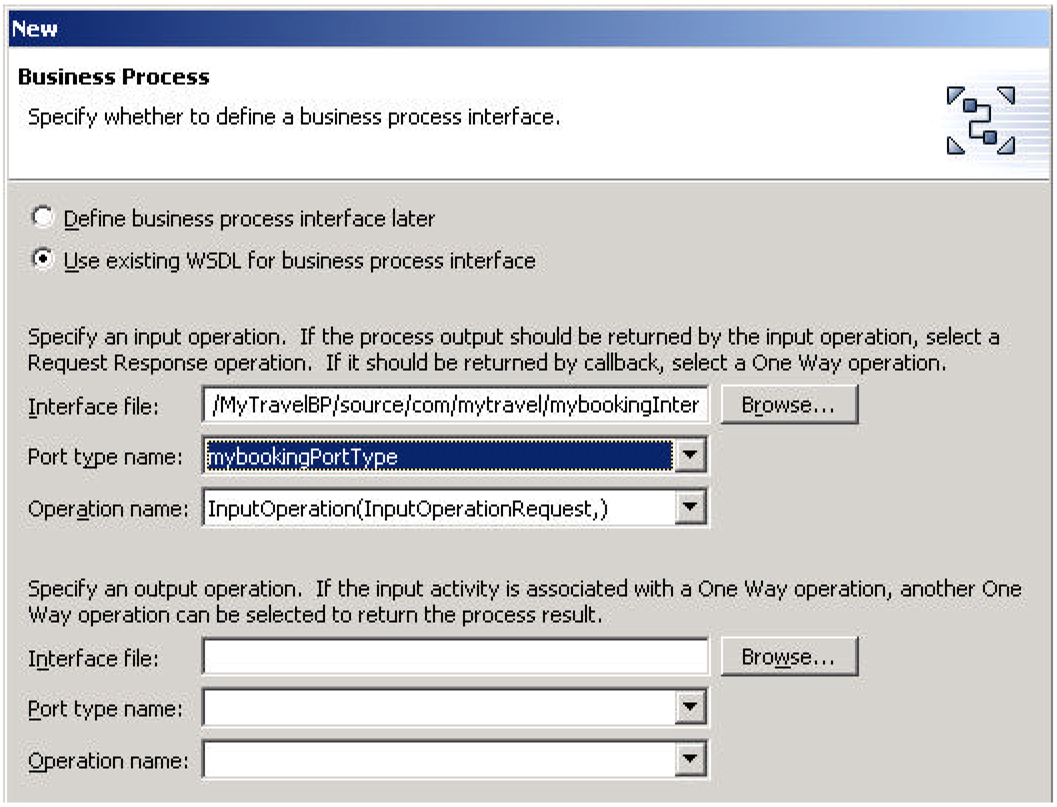
File name: mybooking.process

__ f. Select **Next** to proceed.

__ g. Every Business Process must have an input operation and an output operation. The input operation is called in order to start the Business Process. If the service being called is synchronous, the input operation can return the result(s). If the service being called is asynchronous, a separate operaton can be specified for retrieving the result(s) of the Business Process.

Choose the radio button **Use existing WSDL for business process interface**. The interface definition is contained in **mybookingInterface.wsdl** file. You will use this definition as the interface for the mybooking Business Process.

Select the **Browse...** button. In the file selection dialog expand **MyTravelBP/source/com/mytravel/mybooking**, then select the **mybookingInterface.wsdl** file and click **OK**.



New

Business Process

Specify whether to define a business process interface.

( ) Define business process interface later

(●) Use existing WSDL for business process interface

Specify an input operation. If the process output should be returned by the input operation, select a Request Response operation. If it should be returned by callback, select a One Way operation.

Interface file: /MyTravelBP/source/com/mytravel/mybookingInter

Port type name: mybookingPortType

Operation name: InputOperation(InputOperationRequest,)

Specify an output operation. If the input activity is associated with a One Way operation, another One Way operation can be selected to return the process result.

Interface file:

Port type name:

Operation name:

Click the **Finish** button to proceed.

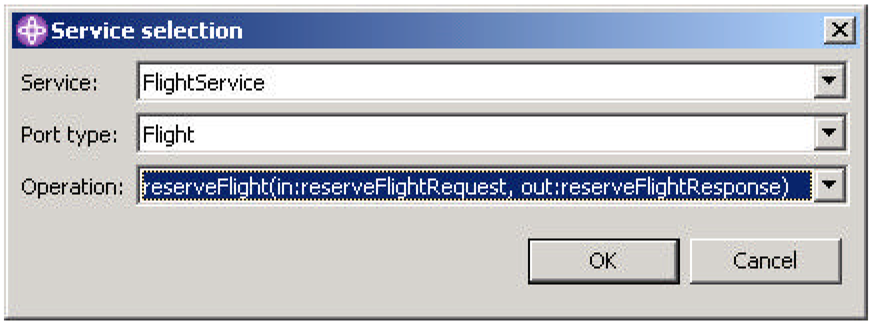
Note: Additional items will appear in the **Tasks** view. You will resolve these items shortly in upcoming steps.

__ i. Upon completion of the previous step, the **Business Process** editor will open, defaulting to the **Process** tab. The **Process** tab displays the **mybooking** Business Process in a manner that provides for graphical/drop-n-drag editing. Currently the **mybooking** Business Process only contains the default Input and Output nodes.

__2. Add the individual Services which will make up the Business Process.

__ a. In the Package Explorer view, expand **MyTravelBP/source/com.mytravel** if it is not expanded. Drag the file **FlightEJBService.wsdl** and drop it on the Business Process editor.

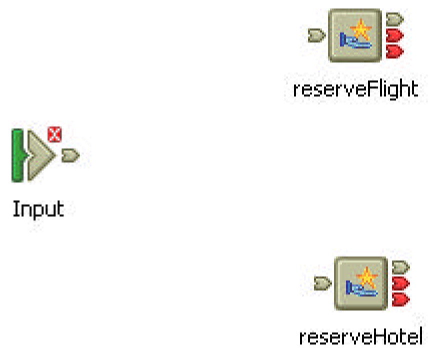
__ b. The operations which are available for the Flight Service will be displayed. Select **reserveFlight** for the Operation and select **OK**.



Service: FlightService
Port type: Flight
Operation: reserveFlight(in:reserveFlightRequest, out:reserveFlightResponse)

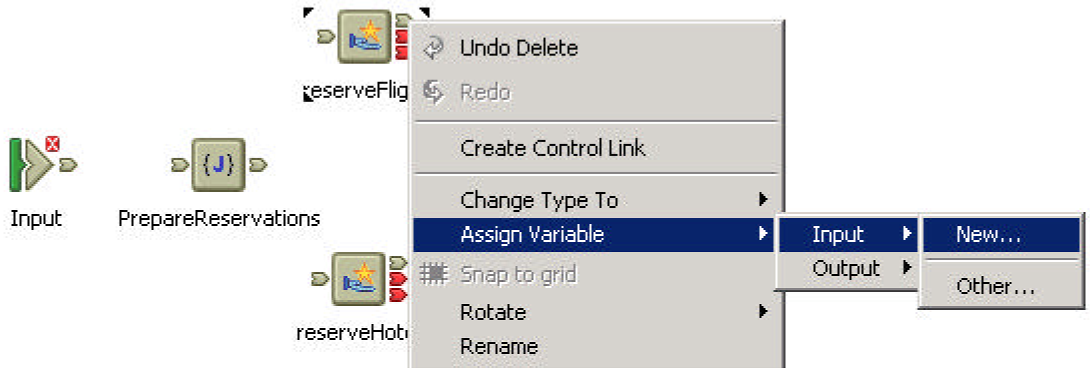
Notice there are 3 terminals on the reserveFlight Activity. The red terminals are for the Fault messages which can be returned should a exception occur in the reserveFlight activity.

__ c. Drag the file **HotelEJBService.wsdl** and drop it on the Business Process editor. Select **reserveHotel** for the Operation. Select **OK**. Your Business Process should look as follows:

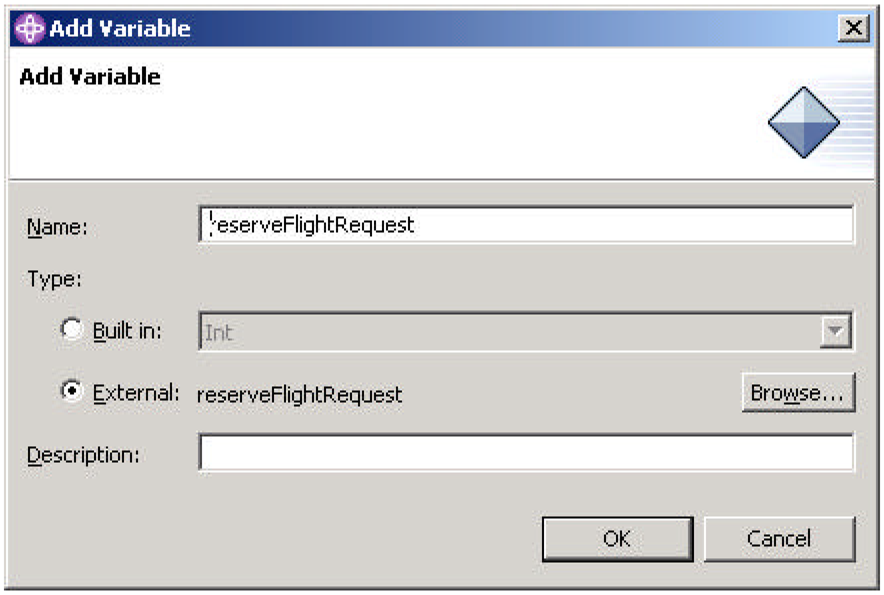


__3. With the services added to your Business Process, you will need to specify the messages which are sent into the services as well as the messages which are returned. You will also need to specify the messages that will be returned should a fault arise during execution of a service. The messages are stored in Variables of the Business Process.

__ a. Right-click on **reserveFlight** Service and select **Assign Variable > Input > New...**. (You could also right-click the Service, select Properties and then implementation).



__ b. Ensure the **External** radio button is selected and set to **reserveFlightRequest**.



Name: reserveFlightRequest

Type:

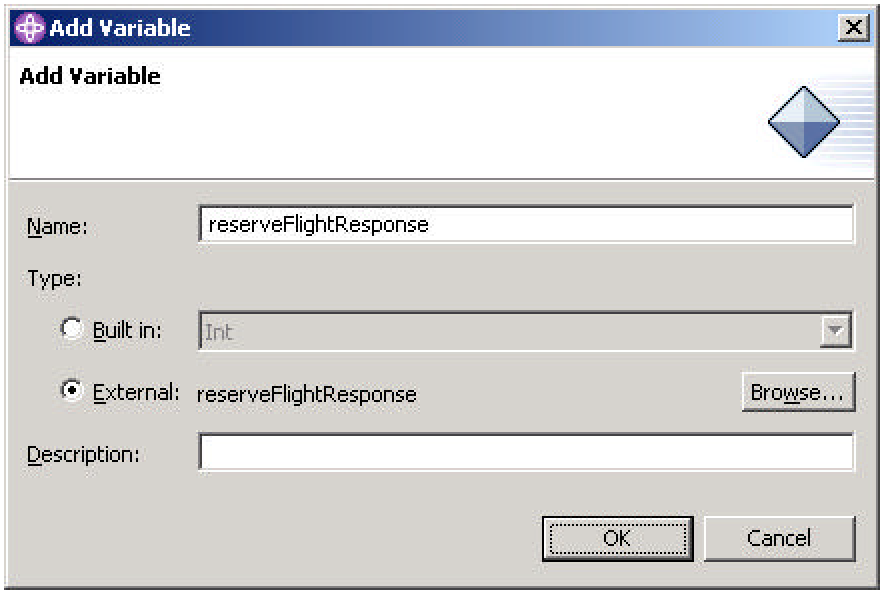
( ) Built in: Int

(•) External: reserveFlightRequest

Description:

Select **OK**.

__ c. Repeat the above two step for the Output, ensuring the External type is **ReserveFlightResponse**.



Name: reserveFlightResponse

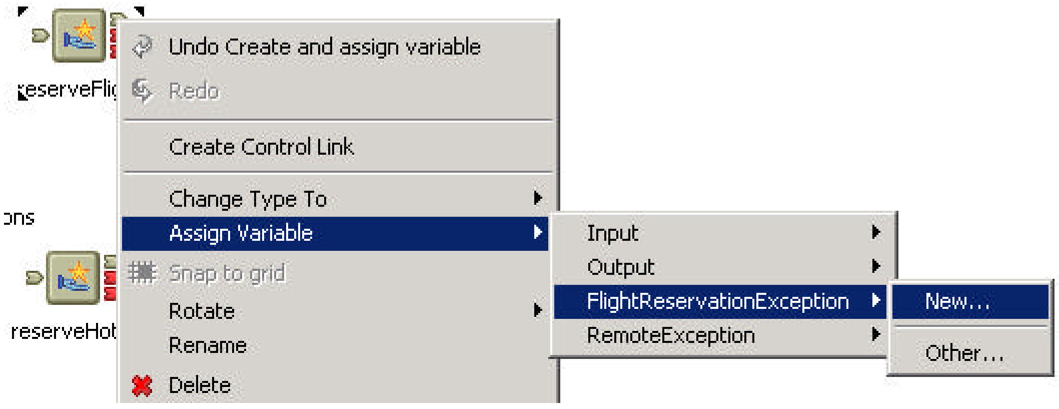
Type:

Built in: Int

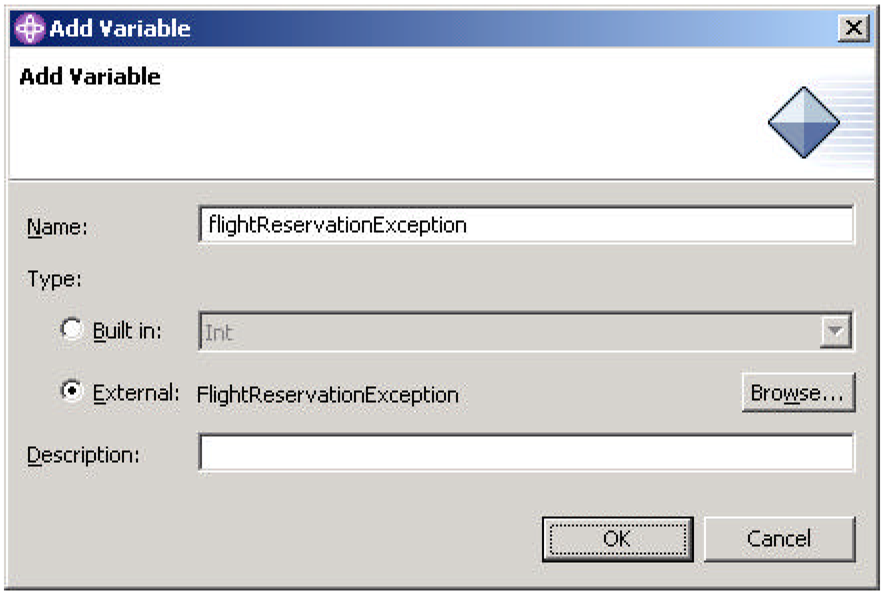
External: reserveFlightResponse

Description:

__ d. Variables can also be specified for the Fault messages which may be returned from the reserveFlight service should it fail. Right-click the reserveHotel service again, select **Add Variable > FlightReservationException > New...**.



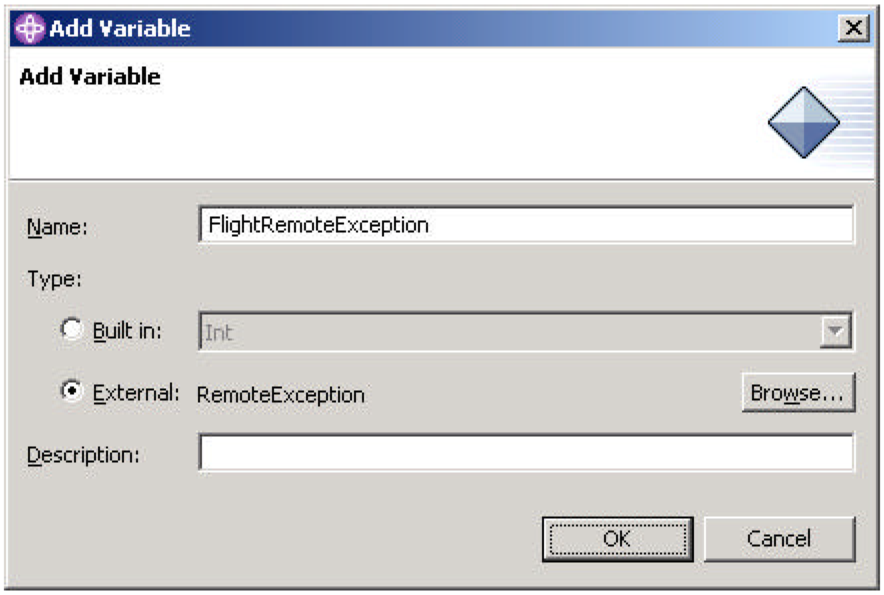
__ e. Ensure the **External** radio button is selected and set to **FlightReservationException.**



Name: flightReservationException
Type:
Built in: Int
External: FlightReservationException
Description:

Select **OK**.

__ f. Repeat above two steps for **RemoteException**. For the Variable Name, specify **FlightRemoteException**. (We need to specify a specific name to avoid conflicts from the Hotel Service which also throws an exception and can return a RemoteException.)

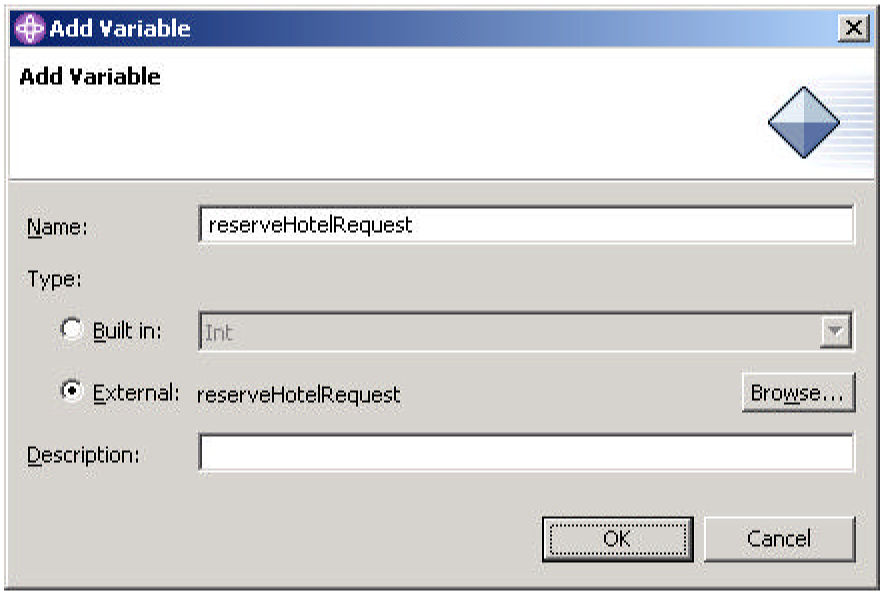


Name: FlightRemoteException
Type:
Built in: Int
External: RemoteException
Description:

Select **OK**.

__4. You will also need to specify messages for the reserveHotel service.

__ a. Right-click on reserveHotel. Select **Add Variable > Input > New...**. Ensure the External button is selected and set to **reserveHotelRequest**. Select **OK**.


Add Variable

Add Variable

Name: reserveHotelRequest

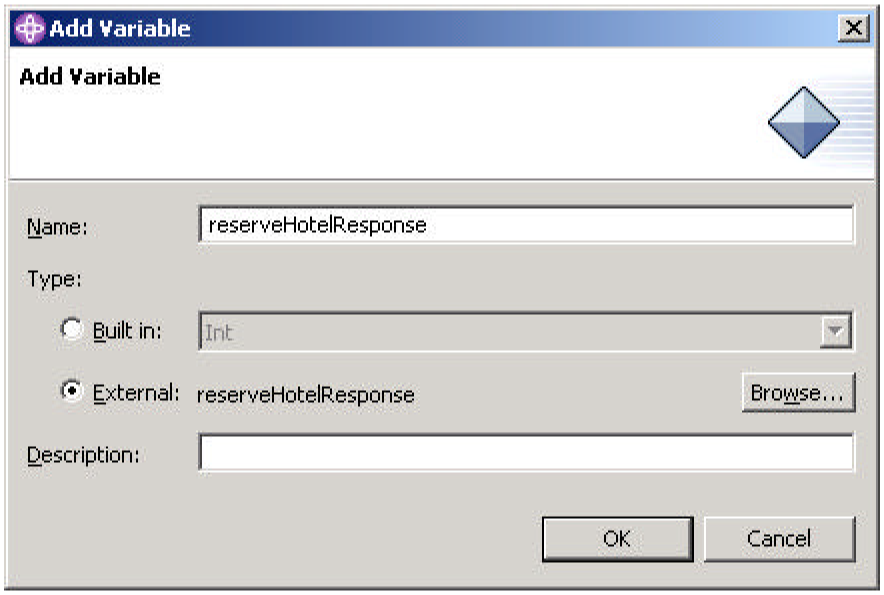
Type:

( ) Built in: Int

(●) External: reserveHotelRequest

Description:

__ b. Right-click on reserveHotel again. Select **Add Variable > Output > New...**. Ensure the External button is selected and set to **reserveHotelResponse**. Select **OK**.


Add Variable

Add Variable

Name: reserveHotelResponse

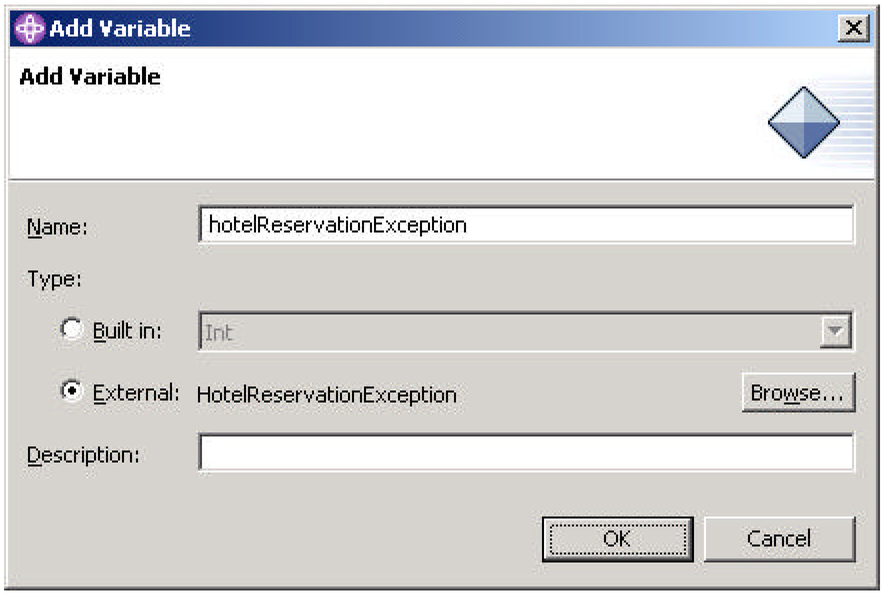
Type:

( ) Built in: Int

(●) External: reserveHotelResponse

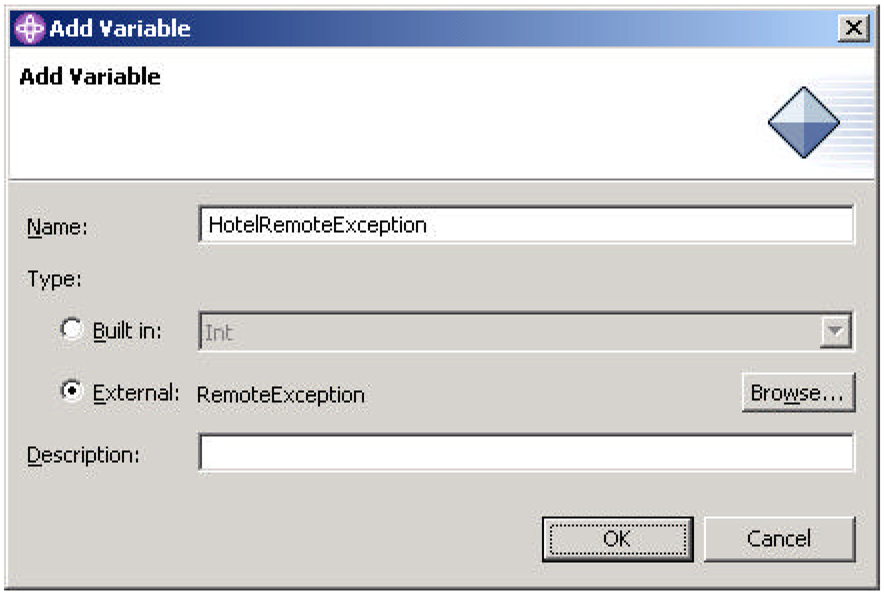
Description:

__ c. Right-click on reserve Hotel again. Select **Add Variable > HotelReservationException > New...**. Ensure the External button is selected and set to **hotelReservationException**. Select **OK**.


Add Variable
Add Variable
Name: hotelReservationException
Type:
( ) Built in: Int
(●) External: HotelReservationException Browse...
Description:
OK Cancel

__ d. Right-click on reserve Hotel again. Select **Add Variable > RemoteException > New...**. Ensure the External button is selected and set to **hotelReservationException**.

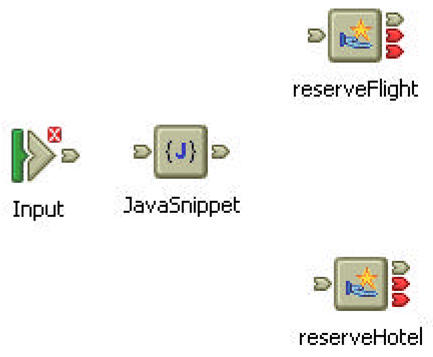
__ e. Change the name to **HotelRemoteException**.


Add Variable
Add Variable
Name: HotelRemoteException
Type:
( ) Built in: Int
(●) External: RemoteException Browse...
Description:
OK Cancel

Select **OK**.

__5. You will need to add a number of Java snippets which will be used for preparing the messages into the reserve Services and handling the responses.

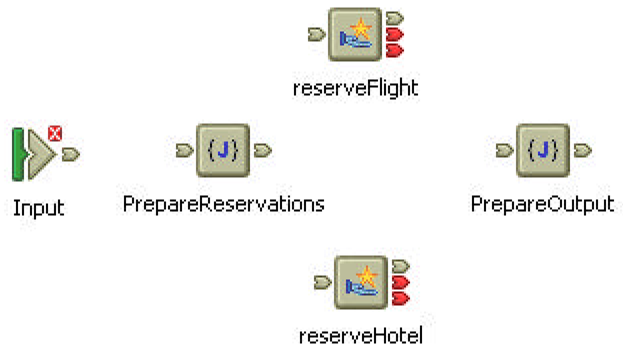
__ a. The first Java snippet will prepare the messages which will be sent to the reserveFlight and reserveHotel Services. From the pallete select Java Snippet, and click within the Business Process editor.



__ b. Right-click the Java snippet and select **Properites**. Select General and change the name to **PrepareReservations**. Select **OK**. You will add the code to prepare the messages later in the lab exercise.

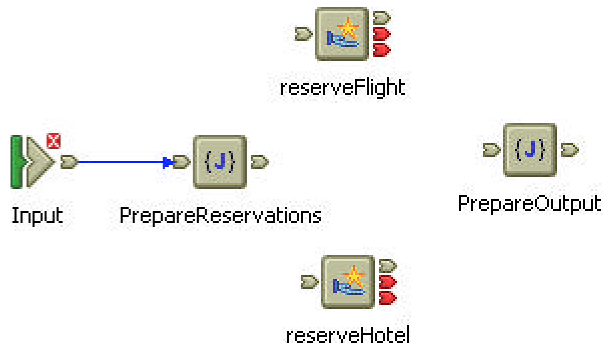
__ c. Following the previous steps, add another Java Snippet which will work with the response messages returned from the reserveFlight and reserveHotel services.

__ d. Change the name of the Java Snippet to **PrepareOutPut**.

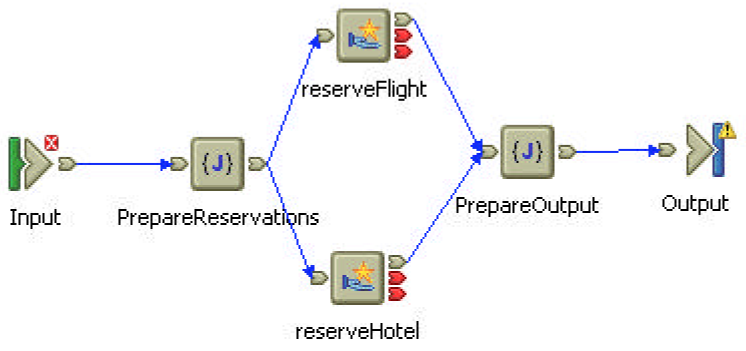


__6. You are now ready to specify the execution flow of your Business Process by connecting the different activities.

__ a. From the pallete, select **Control Link**. Starting at the Input activity, select the output terminal and connect it to the input terminal of the PrepareReservations Java snippet.



__ b. Connect the other activities as shown below. When connecting the reserveFlight and reserveHotel services, do not connect the red terminals. These terminals will be connected Shortly.



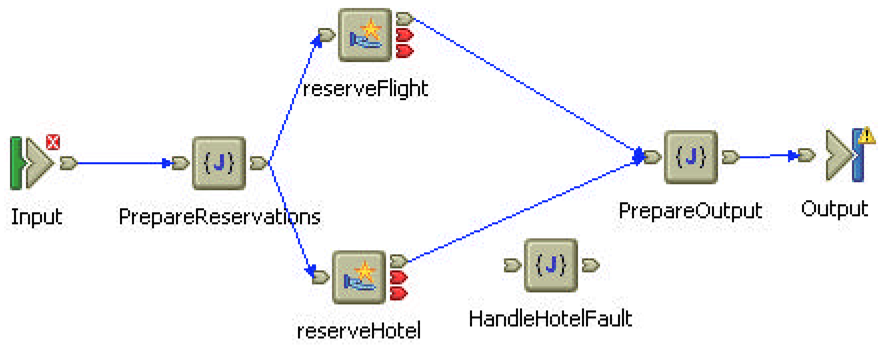
The Control Links you created would be executed during successful execution of the reserveFlight and reserveHotel services. When a reservation request comes in, a flight reservation and a hotel reservation will be made. When both reservations have been completed, a success message will be created in the PrepareOutput Java snippet and then passed onto the Output service.

__7. Although the normal execution has been set, the exception situations for the different services have not been handled. You will need to specify these for your Business Process to complete correctly and to report the exceptions correctly.

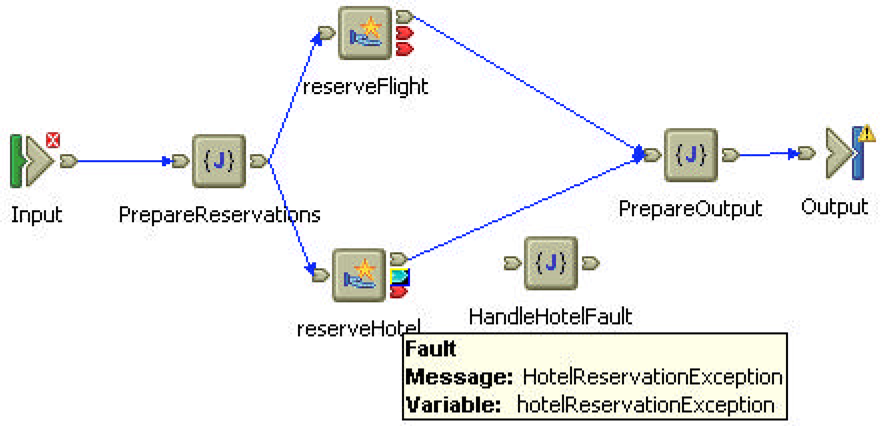
Within a BusinessProcess, there are two ways in which you can handle exception situations. You can handle the exception, performing some type of correcting activity and continuing the Business Process or you can return the exception as the result of the Business Process.

__ a. For the hotelReservationException, which is thrown if a hotel is requested that is not on the preferred list, it can be handled within the Business Process. For the scenario where a Flight has been reserved, you would not want to fail the Business Process because a hotel under a specific name cannot be found. You would want to mark the reservation as complete and possibly notify the employee to select a different hotel while keeping the flight reservation within the Output message.

To handle the exception, you can "wire" the Fault Terminal to another activity which can take the appropriate action before resuming the business process. Add a **Java snippet** to you Business Process and change the name to **HandleHotelFault** (for instructions on adding a Java snippet and changing the name are described in earlier steps).

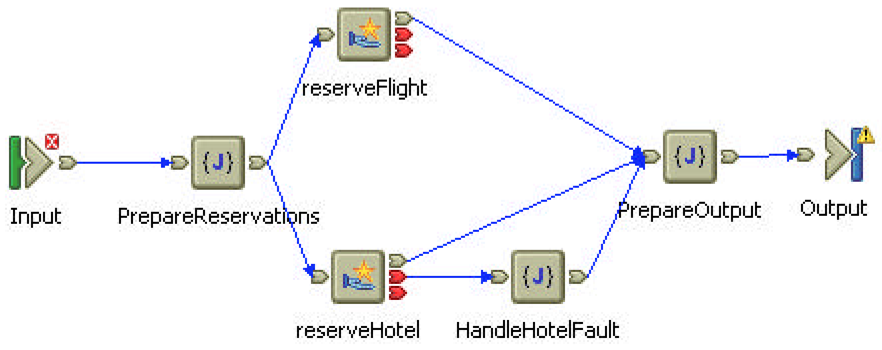


__ b. Hover your mouse over the Fault Terminals on the reserveHotel service. The Message and Variable will be displayed.



Fault
Message: HotelReservationException
Variable: hotelReservationException

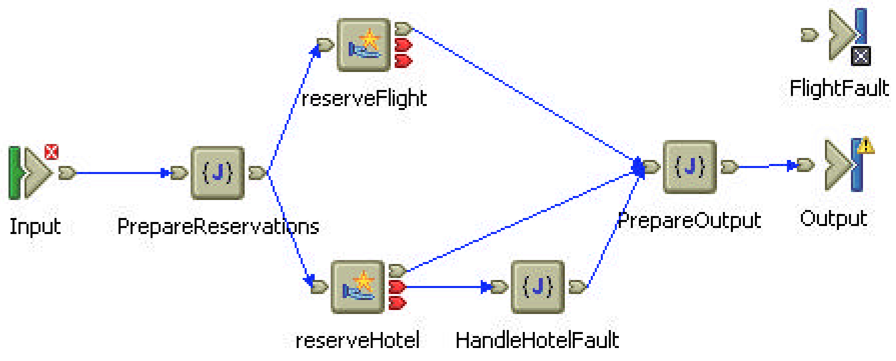
__ c. Create a Control Link between the **HotelReservationException** Fault Terminal and the input terminal of the **HandleHotelFault**. Create a Control Link from the output terminal of the **HandleHotelFault** Java snippet to the input terminal of the **PrepareOutput** Java Snippet.



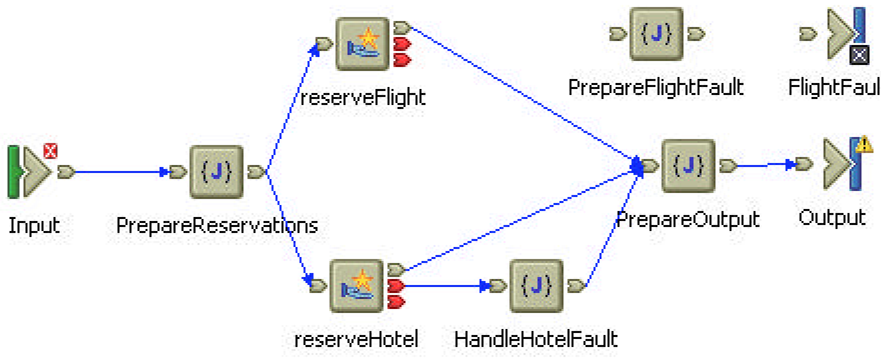
If the HotelReservationException occurs, execution will occur to the HandleHotelFault Java snippet and then onto PrepareOutput with out failing the Business Process. The HandleHotelFault can be replaced at a later time with a service which may notify the employee that a hotel cannot be reserved, You can also record the result in a message which can then be sent when the Output is reached as part of a summary of the travel reservation. With this Fault, it is never directly exposed outside of the Business Process.

__ d. Another way in which a Fault can be handled in a Business Process is to catch the Fault and pass the failure onto another service outside the Business Process for proper handling. For a failure in the reserveFlight Service, you may want to perform some other type of process, looking at other airlines, destination cities, or forms of transportation. In this case, the execution should be halted and the Business Process should be marked as failed. The failure can then be passed onto the outside service.

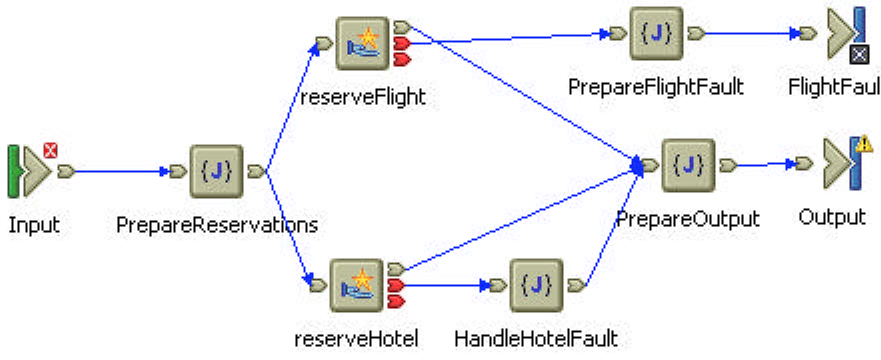
Select Fault Terminal from the pallete and click in the Business Process Editor. Change the name to **FlightFault**.



__ e. To pass the failure out of the Business Process, you will need to create a message with the appropriate error information which can be sent to the FlightFault. Unless the type of the FlightReservationException message matches the type of the message to be sent to the FlightFault, you will have to place the contents of the FlightReservationException message into the FlightFaultmessage. To move these contents, you can use a Java snippet to prepare the FlightFault message. Create a Java snippet and name it **PrepareFlightFault**.

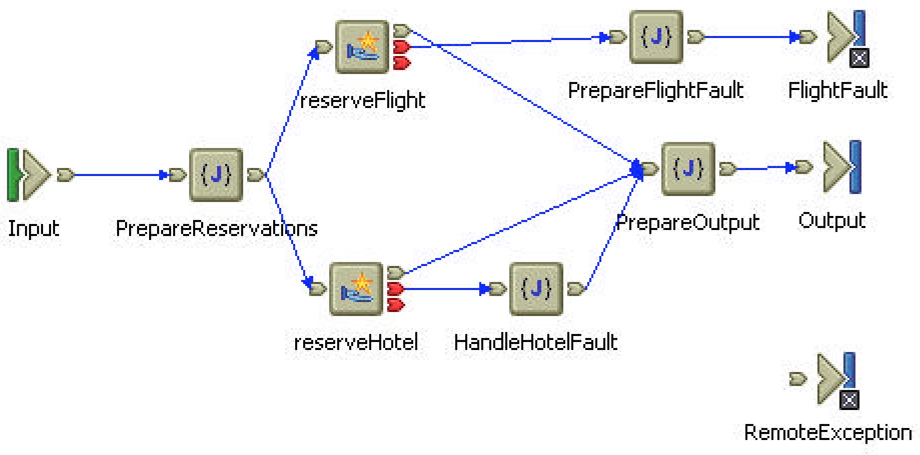


__ f. Using a Control Link, connect the Fault Terminal **FlightReservationException** to the input terminal of the **PrepareFlightFault** Java snippet (hover your mouse over the fault terminals to select the correct terminal). Connect the output terminal of the **PrepareFlightFault** Java snippet to the input terminal of the **FlightFault**.

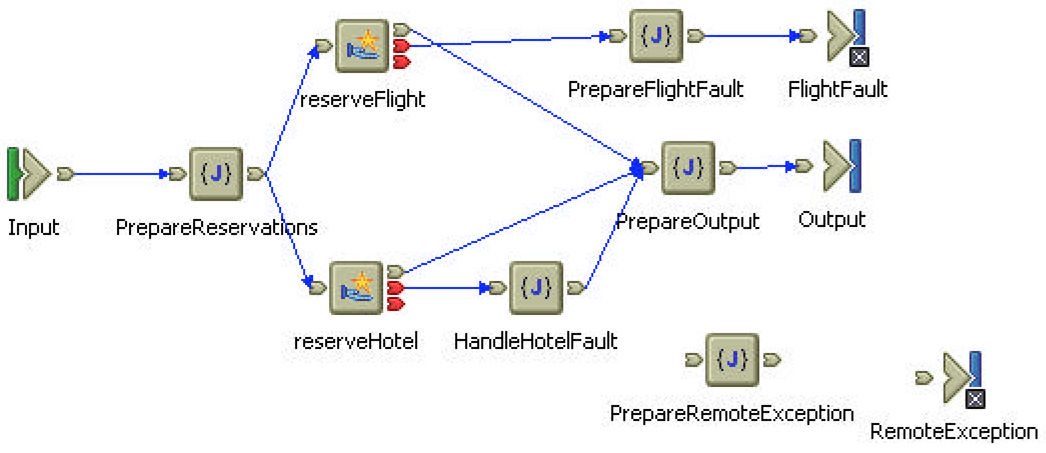


__ g. The remaining Fault Terminals on the reserveFlight and reserveHotel are for javax.rmi.RemoteExceptions that have been defined as Fault messages when the service was created for these Enterprise beans. In most cases you will not “wire” these Fault Terminals to another activity to contiue execution of your Business Process. The exceptions which the Fault Terminals represent are an indication of something much more seriously wrong (network, configuration, or access problems) with the implementation of the Business Process. These Fault Terminals should be wired directly to Fault Activities with only Java Snippets inbetween, used to prepare the message which will be sent to the Fault service.

Select **Throw Fault** from the pallete and click within the Business Process editor. Change the name to **RemoteException**.

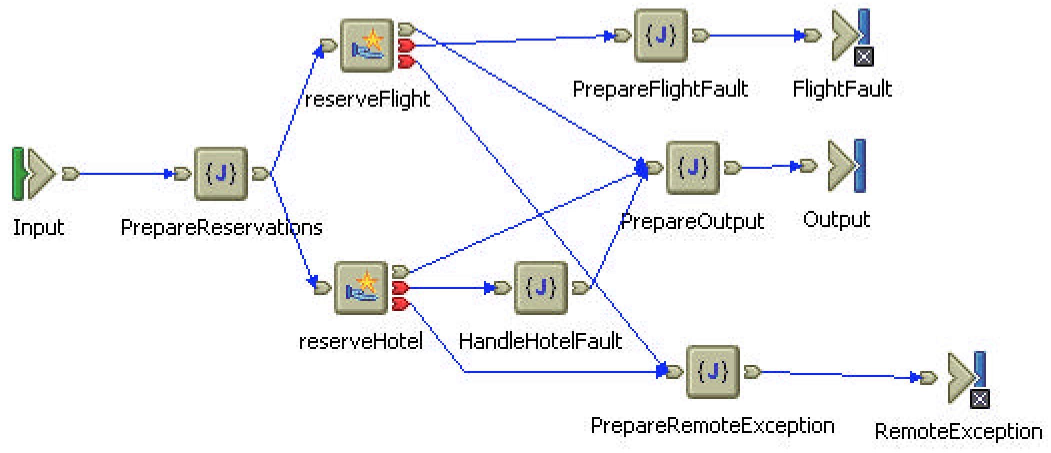


__ h. Select **Java Snippet** from the pallete and click within the Business Process editor. Rename the snippet, **PrepareRemoteException**.



__ i. Connect the **RemoteException** Fault Terminals on the **reserveFlight** and **reserveHotel** service to the input terminal of the **PrepareRemoteException** snippet.

__ j. Connect the ouput terminal of the **PrepareRemoteException** snippet to the input terminal of the **RemoteException** activity.

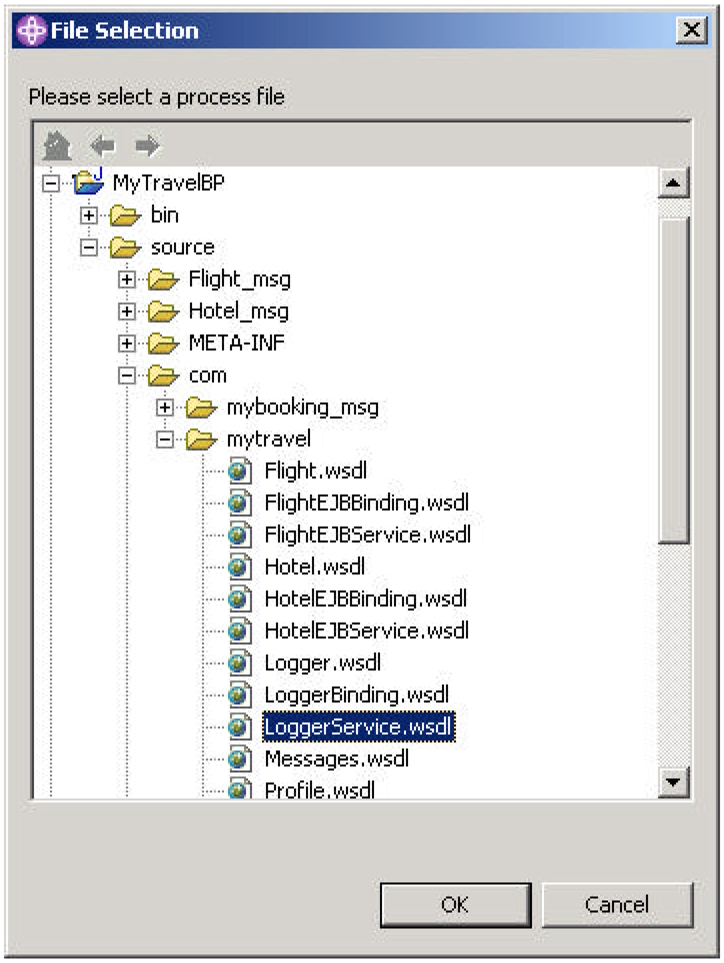


__8. The mybooking Business Process is a macroflow as you have the possibility of parallel processing when a flight and a hotel are reserved. For macroflows you have a separate operation which is called when the Business Process is completed. The operation is available on a separate service. You will need to specify the service and operation for your Output.

__ a. From the pallete, select **Selection**.

__ b. Right-click Output and select **Properties**. Select **Implemenation**.

__ c. The service is described in the LoggerService.wsdl file. The service basically records the results of the reservation process to a repository where it can be viewed. For the File, select Browse.... Expand **MyTravelBP/source/com/mytravel**. Select **LoggerService.wsdl**. Select **OK**.

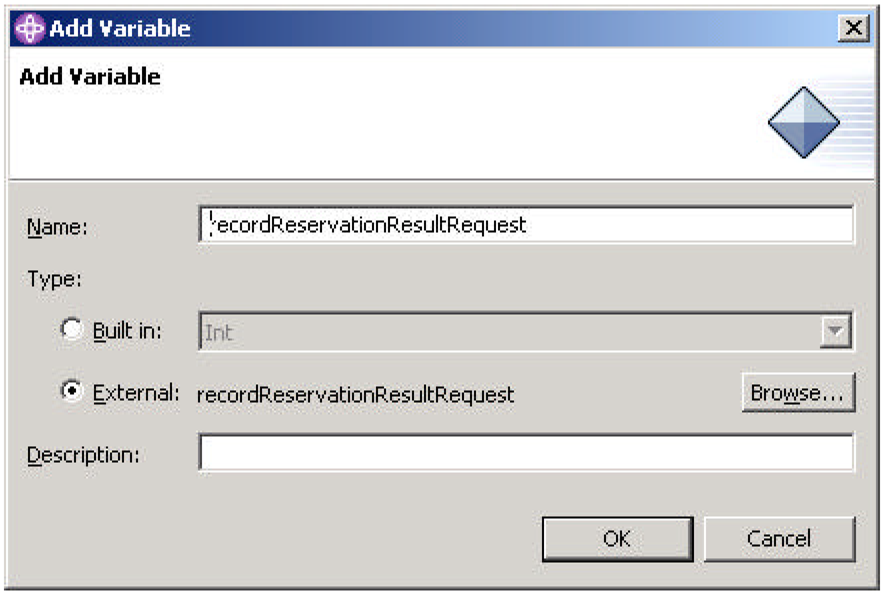

File Selection

Please select a process file

MyTravelBP
bin
source
Flight_msg
Hotel_msg
META-INF
com
mybooking_msg
mytravel
Flight.wsdl
FlightEJBBinding.wsdl
FlightEJBService.wsdl
Hotel.wsdl
HotelEJBBinding.wsdl
HotelEJBService.wsdl
Logger.wsdl
LoggerBinding.wsdl
LoggerService.wsdl
Messages.wsdl
Profile.wsdl

__ d. In the Properties for Output window, select **recordReservationResult(...,...)**.

__ e. You will need to create a Variable to be used as a message to the service indicating the outcome of the reservation. Select **New Variable**. Insure the External radio button is selected and set to **recordReservationResultRequest**.


Add Variable

Add Variable

Name: recordReservationResultRequest

Type:

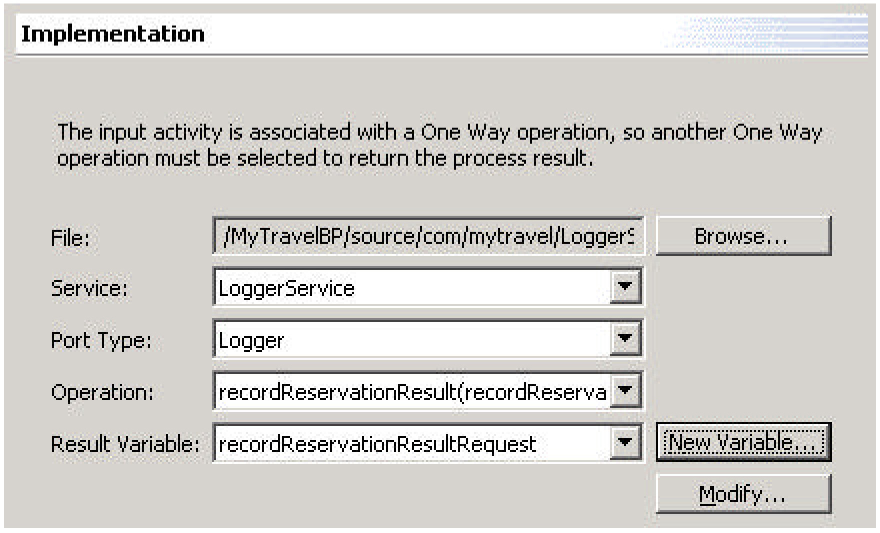
( ) Built in: Int

(●) External: recordReservationResultRequest

Description:

Select **OK**.

__ f. The implementation details for the Output are complete.


Implementation

The input activity is associated with a One Way operation, so another One Way operation must be selected to return the process result.

File: /MyTravelBP/source/com/mytravel/LoggerS

Service: LoggerService

Port Type: Logger

Operation: recordReservationResult(recordReserva

Result Variable: recordReservationResultRequest

Select **OK**.

__9. For the Fault Activities, you will also need to specify the services to which the failure messages are sent. The services are described in the LoggerService.wsdl file. The Logger services record the failure fresult within a repository where it can be viewed.

__ a. Right-click **FlightFault** and select **Properties**. Select **Implementation**.

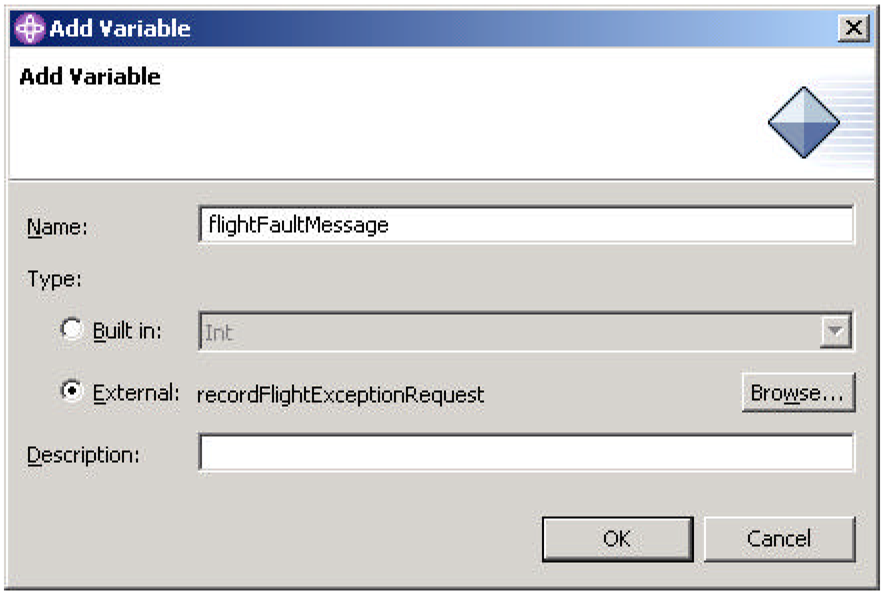
__ b. For the File, select **Browse...**. Select **LoggerService.wsdl** again and click **OK**.

__ c. In the Properties for FlightFault window, ensure the Operation is set to **recordFlightException(...,...)**.

__ d. A Variable will need to be created for the message that will be sent to the service. Select **New Variable...**.

__ e. For the Name, change it to **flightFaultMessage**.

__ f. Ensure the External radio button is selected and set to **recordFlightExceptionRequest**.


Add Variable

Add Variable

Name: flightFaultMessage

Type:

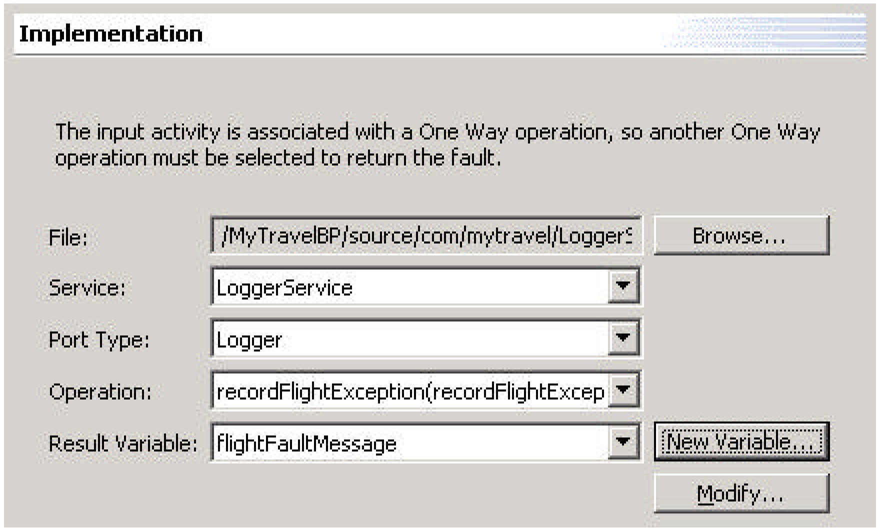
( ) Built in: Int

(●) External: recordFlightExceptionRequest

Description:

Select **OK**.

__ g. The implementation for the Fault service is complete.



Implementation

The input activity is associated with a One Way operation, so another One Way operation must be selected to return the fault.

File: /MyTravelBP/source/com/mytravel/LoggerS

Service: LoggerService

Port Type: Logger

Operation: recordFlightException(recordFlightExcep

Result Variable: flightFaultMessage

__ h. Select **OK**.

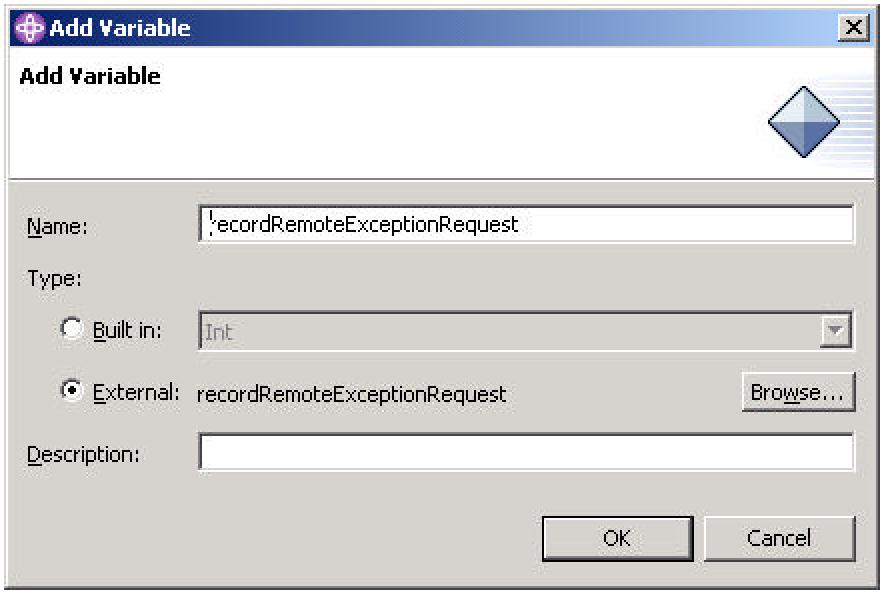
__ i. Repeat these steps for the RemoteException Fault Activity. Right-click RemoteException and select **Properties**. Select **Implementation**.

__ j. For the File, select **Browse...**. Select **LoggerService.wsdl** and select **OK**.

__ k. In the Properties for RemoteException window, for the Operation, select **recordRemoteException(...,...)**.

__ l. You must specify a message to be sent to the service and the recordRemoteException operation. Select **New Variable...**.

__ m. Ensure the External Radio button is selected and set to **recordRemoteExceptionRequest**.


Add Variable

Add Variable

Name: recordRemoteExceptionRequest

Type:

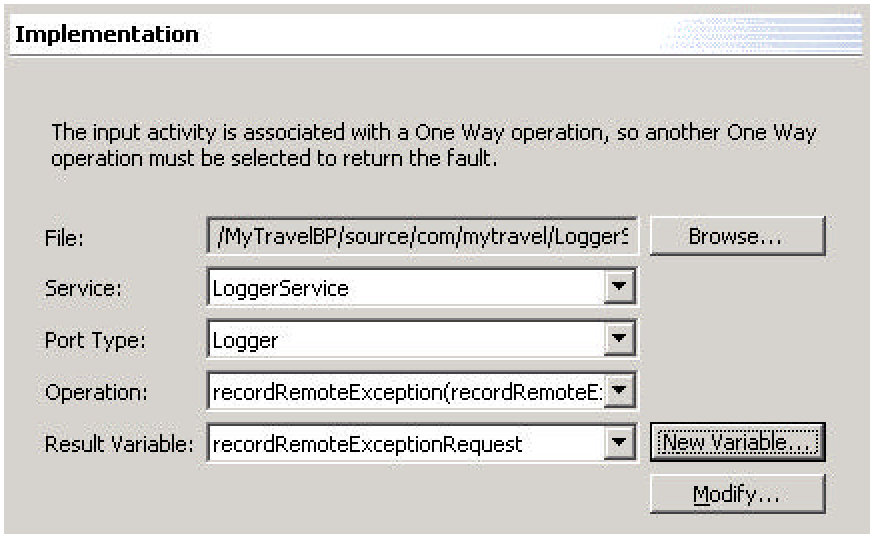
Built in: Int

External: recordRemoteExceptionRequest

Description:

__ n. Select **OK**.

__ o. The implementaton for the Fault service is complete.


Implementation

The input activity is associated with a One Way operation, so another One Way operation must be selected to return the fault.

File: /MyTravelBP/source/com/mytravel/LoggerS

Service: LoggerService

Port Type: Logger

Operation: recordRemoteException(recordRemoteE:

Result Variable: recordRemoteExceptionRequest

__ p. Select **OK**.

__10. The Final step in defining your Business Process is to add the support for preparing the different messages which are sent to the different activities.

__ a. Select the Java snippet **PrepareReservations** and select the **Show Java** button.

__ b. The PrepareReservations snippet takes the input message which has the reservation information and breaks it into a flight reservation message and a hotel reservation message. Add the following code between the user code comments. (To save time copy the code from PrepareReservations.txt).

```
InputReservationMessage in = getInput();
ReserveFlightRequestMessage flightIn = getReserveFlightRequest();
ReserveHotelRequestMessage hotelIn = getReserveHotelRequest();

flightIn.setAirline(in.getFlightAirline());
flightIn.setSource(in.getFlightSource());
flightIn.setTarget(in.getFlightTarget());
flightIn.setDepartureDate(in.getFlightDepartureDate());
flightIn.setReturnDate(in.getFlightReturnDate());

hotelIn.setName(in.getHotelName());
hotelIn.setCity(in.getHotelCity());
hotelIn.setFrom(in.getHotelFrom());
hotelIn.setTo(in.getHotelTo());

setReserveFlightRequest(flightIn);
setReserveHotelRequest(hotelIn);


System.out.println("Java Snippet:  PrepareReservations");
```

__ c. Select the Java snippet **HandleHotelFault**. This snippet can be used to prepare information indicating the Hotel reservation failed. For your Business Process however, you will just add a System.out.println to indicate this snippet was executed. This will be used to verify the exectution when the business process is tested. Add the following code between the user code comment(To save time copy the code from HandleHotelFault.txt).

```
System.out.println("Java Snippet:  HandleHotelFault");
```

__ d. Select the Java snippet PrepareFlightFault. This snippet will prepare the message to be sent to the FlightFault service. Add the following code between the user code comment (To save time copy the code from PrepareFlightFault.txt).

```
FlightReservationExceptionMessage flightOut =
                                 getFlightReservationException();
RecordFlightExceptionRequestMessage faultIn =
                                         getFlightFaultMessage();
faultIn.setMsg(flightOut.getDetail().getMessage());

setFlightFaultMessage(faultIn);


System.out.println("Java Snippet:  PrepareFlightFault");
```

__ e. Selet the Java snippet **PrepareOutput**. In this snippet you can build the message which will be sent to the Output service to indicate the result of the reservation. Add the following code between the user code comments (To save time, copy the code from PrepareOutput.txt).

```
RecordReservationResultRequestMessage out =
getRecordReservationResultRequest();

out.setMsg("Reservation Complete");
setRecordReservationResultRequest(out);

System.out.println("Java Snippet:  PrepareOutput");
```

__ f. Save the process by pressing **Ctrl-S**.

Part Four: Prepare the Business Process for WebSphere

__1. Generate Deploy Code.

__ a. From the **Package Explore** view of the **Business Integration** perspective, right-click on the **mybooking.process** file. From the pop-up menu, select **Enterprise Services** > **Generate Deploy Code...**.

__ b. On the **Deployment** panel, select **EJB** from the **Inbound binding type:** drop-down list. Click the **Next** button to proceed.

__ c. On the **Inbound Service Files** panel, click the **Next** button to proceed.

__ d. On the **EJB Port** panel, select **Finish.**

__2. Your application is complete and can be tested within the Test Environment of WebSphere Application Server v5 Enterprise.

Part Five: Installing the Solution (Optional)

__1. Start WebSphere Studio Application Developer Integration Edition.

__ a. Select **Start > Programs > IBM WebSphere Studio** and select Application Developer Integration Edition. In the dialog box, enter **c:\labfiles50\bpfaultlab\workspaceSolution**. It will start Application Developer Integration Edition with an empty workspace. An empty workspace will leave your existing workspace untouched and help avoid name conflicts between what you may already have in your workspace and what you will be creating in this lab.

__2. Once Application Developer Integration Edition starts, import in the solution.

__ a. Select **File > Import...**.

__ b. Select **EAR file** and **Next**.

__ c. Select **Browse...** and navigate to **c:\labfiles50\bpfaultslab\solution\MyTravelBP.ear** and select **OK**.

__ d. For the Project name, enter **MyBooking**.

__ e. Select **Finish**.

__3. Repeat the above steps for importing in the solution MyTravel.ear.

__4. At this point, you can view the different Services and the Business Process.

What you did in this exercise

In this exercise you built a simple Business Process composed of two individual Services that both raise faults. After setting control links to define the sequence of execution within the Business Process you configured Variables which will hold and receive the messages sent and received to and from the individual Services. With that process defined, you connected the fault terminal of the reserveHotel service to a Java snippet activity, thus handling the fault. The FlightReservationFault is not handled but raised to the next level and can be exposed outside the Business Process, so in case the fault occurs (there are no seats available to do the flight reservation) the whole MyBooking process is terminated abnormally. Once the Business Process was completely defined, you deployed the Business Process with an EJB binding in preparation of running on WebSphere Application Server v5 Enterprise.