

Integrating Celtix with an Intalio|n3 Process

Table of Contents

Overview	1
Installing the Software	2
Celtix.....	2
Intalio n3.....	2
The Intalio n3 Tutorial	2
The Intalio n3 Application	2
The Basic Intalio n3 Application	3
Developing the Basic Application.....	3
Defining the Mappings.....	6
Deploying and Testing the Basic Application.....	8
Reviewing the Message Content.....	9
Exposing the Basic Intalio n3 Application as a Web Service.....	9
Modifying and Redeploying the Basic Intalio n3 Application.....	9
Running the Modified Basic Application.....	11
Writing the Celtix Client to the Basic Application	11
Setting up the Directory Hierarchy.....	11
Compiling and Running the Client Application.....	12
The Extended Intalio n3 Application	12
The Celtix Web Service WSDL File.....	12
Developing the Extended Application.....	12
Making the WSDL File Available to the Intalio n3 Server.....	12
Configure an Intalio n3 WSDL Connector.....	12
Integrating the StockCheckerService into the Business Process.....	13
Writing the Celtix Web Service	15
Setting up the Directory Hierarchy.....	15
Compiling and Running the Server and Test Client Applications.....	15
Testing the Server Application.....	15
Running the Entire Extended Intalio n3 Application	15
Appendix	16
The Basic Application Files.....	16
The file tutorial_revised.xsd.....	16
The file build.xml.....	19
The client mainline.....	19
The Extended Application Files.....	21
The file intalio_integration.wsdl.....	21
The file build.xml.....	25
The test client mainline.....	25
The server mainline.....	26
The StockCheckerImpl code.....	27

Overview

An Intalio|n3 business process may be exposed as a Web service, which allows a client application to initiate a business process by sending a Web service request. Additionally, a task within an Intalio|n3 business process may represent an invocation on a distributed Web service. When the process reaches this task, Intalio|n3 sends a Web services request to the external Web service. This application note will describe how to write two Celtix applications: an application that kicks off an Intalio|n3 process, and an application that provides the business logic representing an Intalio|n3 task.

Installing the Software

You will need to install Celtix, Intalio|n3 Server v3.0.0.1, Intalio|n3 Console v3.0, and Intalio|n3 Designer v3.0. Later versions of the Intalio|n3 components are also suitable. You will use the Apache Derby and Active MQ database management system that is embedded within the Intalio|n3 Server product, so there is no need for an Oracle or SQL Server installation. The example will be developed in Java, so you will also need the JDK v5.0 in order to compile the Celtix application code. Celtix requires this version of the JDK. Intalio|n3 includes its own JRE and is not dependent on an external JRE.

Celtix

Install a binary distribution as described in the product documentation. Use any product version following the milestone 3 release.

Intalio|n3

The The installation procedure for all three Intalio|n3 components is described in the Administrator's Guide. All three applications use a common license file, which you must identify during each installation. During the Intalio|n3 Server installation process you will be asked to select a database management system. Select the radio button corresponding to the database included in the server product.

Prior to installing this software, use the `netstat -a` command line utility to determine which TCP/IP ports are already in use. By default, Intalio|n3 will use ports 8081 and 4321. If either of these ports is already in use, you will need to specify different port numbers in the Server Configuration window during the Intalio|n3 Server installation process (in developing this note, port numbers 8999 and 4321 were used). You must also enter a Shutdown Password into the Server Configuration window. This password is not the same as the password needed to connect the Intalio|n3 Console to the server. On Windows, installation of each product also creates Start menu entries.

After the products are installed, use the Start Server menu entry to run the Intalio|n3 Server. Once the server has initialized, use the Intalio-n3 Console menu entry to start the console. In the console's login window, enter `admin` and `changeit` into the User name and Password text boxes. (Although the User name and Password text boxes contain values, the password entry is invalid. You must enter `changeit`.) If you changed the port used by the server from the default value of 8081, you will also need to edit the value in the Port text box. The console should now be able to connect to the server.

Now use the Intalio-n3 Designer menu entry to start the designer. If you changed the port used by the server, you must reconfigure the designer. On the designer's menu bar select the Edit > Preferences... menu item. Click on the Deployment entry in the Preferences window, and in the Server URL text box edit the URL as required. Click the Test command button to confirm that the designer can contact the server. Finally, click the OK command button to save the edit.

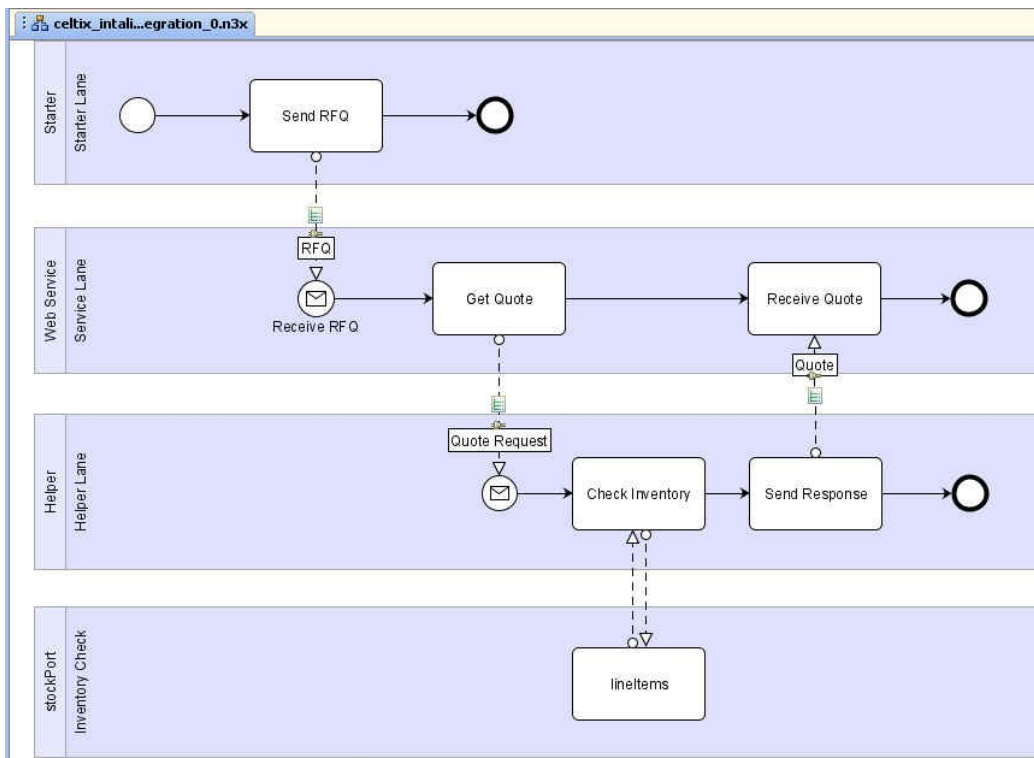
The Intalio|n3 Tutorial

This tutorial illustrates how to use the Intalio|n3 Designer to develop a business process. The process described in the remainder of this note assumes that you are familiar with the concepts described in this tutorial.

The Intalio|n3 Application

You will create a new process diagram that includes four pools. One pool represents the process used to kick-off the process flow; the second and third pools include the tasks and activities that represent the actual business processes; and the fourth pool represents the external Celtix Web service. When you reconfigure this application as a Web service, the first pool will be excluded from the application and the start event in one of the other pools will become the endpoint on which the Celtix client application will invoke to kick off the business flow.

The following diagram shows the completed process diagram. The application created within this note has been derived from the Intalio|n3 tutorial application. The schema file used to describe the message content within the Intalio|n3 process is provided at the end of this document.



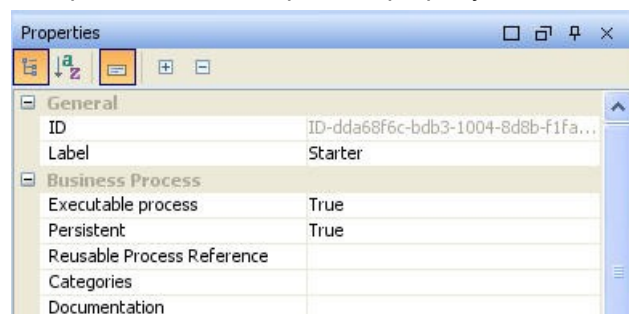
The Basic Intalio|n3 Application

Initially, you develop a basic Intalio|n3 process comprised of the first three process pools and then expose it as a Web service to a Celtix client. Then, you will extend the process diagram, adding the fourth pool, to include a distributed Celtix Web service as the processing logic behind an Intalio|n3 task.

Developing the Basic Application

The following outline presents the steps needed to complete the basic process diagram.

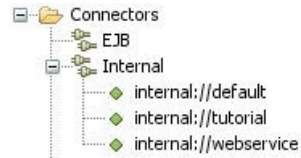
1. Start the Intalio|n3 Designer. Once the designer initializes, highlight the icon representing the Intalio|n3 workspace, right click, and select Create Folder from the popup menu. Create a new folder named celtix.
 - a. Then create the subdirectory celtix/schemas.
 - b. Copy the schema file (see the Appendix at the end of this document) and save as tutorial_revised.xsd in the directory workspace/celtix/schemas.
2. Now select the File > Create Diagram menu item. This creates a diagram with an empty pool containing a single lane.
 - a. Rename the pool Starter. Rename the lane Starter Lane.
 - b. Confirm that the pool's Executable process property is set to True.



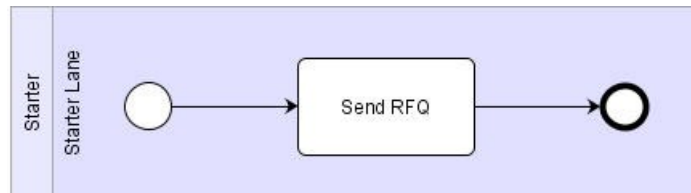
- c. Save the diagram with the name celtix_intalio_integration into the directory workspace/celtix.
3. Before you begin to create the remainder of the diagram, define the internal connector that will be

used in this application.

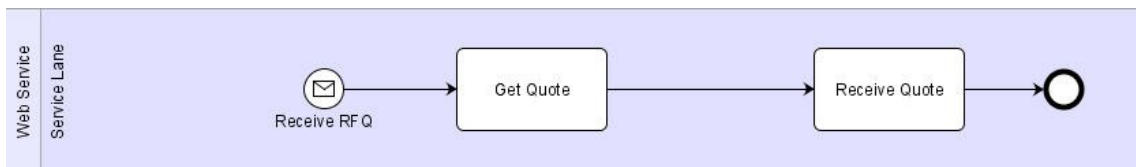
- a. Select the Connector tab.
- b. Highlight the Internal icon, right click, and select Configure from the popup menu.
- c. In the Connector Properties window, click the Add command button. In the Scheme text box, enter internal; in the Channel Name text box, enter webservice.
- d. Click the Save and OK command buttons. This adds the connector internal://webservice to the listing under Internal.



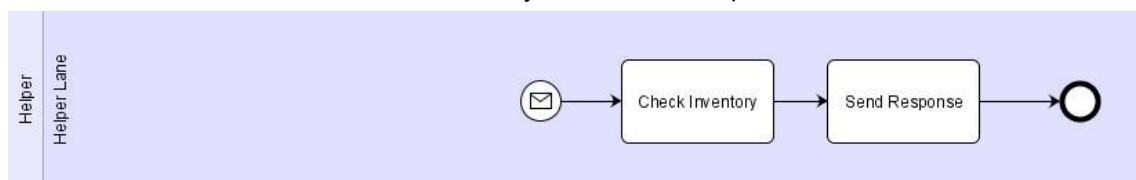
4. Now return to the process diagram and add two additional pools.
 - a. Rename the second pool Web Service; rename its lane Service Lane.
 - b. Rename the third pool Helper; rename its lane Helper Lane.
5. Use an Empty Start Event Shape, a Task Basic Shape, an Empty End Event Shape, and the Sequence Flow Tool to define the activities in the Starter/Starter Lane process.
 - a. Rename the task Send RFQ.



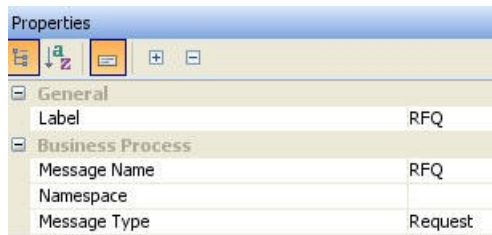
6. Use a Message Start Event Shape, two Task Basic Shapes, an Empty End Event Shape, and the Sequence Flow Tool to define the activities in the Web Service/Service Lane process.
 - a. Rename the start event Receive RFQ.
 - b. Rename the tasks Get Quote and Receive Quote.



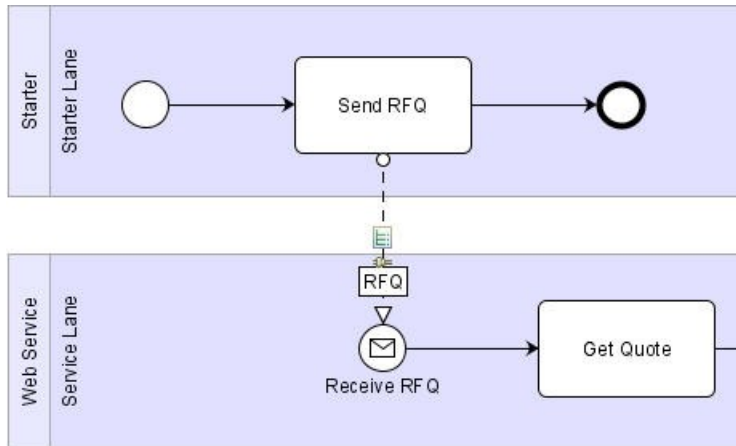
7. Use a Message Start Event Shape, two Task Basic Shapes, an Empty End Event Shape, and the Sequence Flow Tool to define the activities in the Helper/Helper Lane process.
 - a. Rename the start event Receive Request.
 - b. Rename the tasks Check Inventory and Return Response.



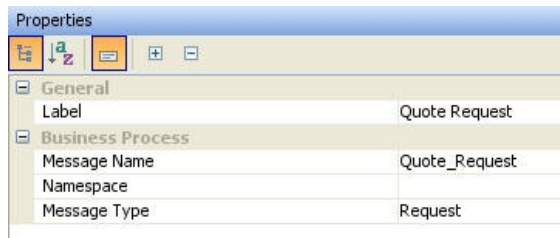
8. Use the Sequence Flow Tool to create a link between the Send RFQ task and the Receive RFQ start event.
 - a. Use RFQ as the label and change the Message Type to Request.



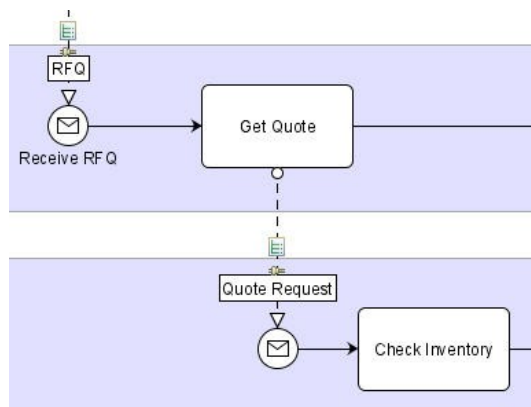
9. Click on the Navigator tab and expand the icon representing the schema tutorial_revised.
 - a. Drag the type intalio:RFQ from the schema listing onto the RFQ link.
10. Click on the Connectors tab and expand the icon representing the internal connectors.
 - a. Drag the internal://webservice connector onto the RFQ link.



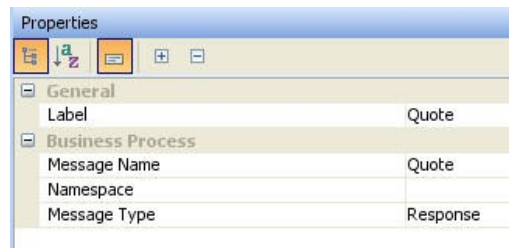
11. Use the Sequence Flow Tool to create a link between the Get Quote task and the Receive Request start event.
 - a. Use Quote Request as the label and change the Message Type to Request.



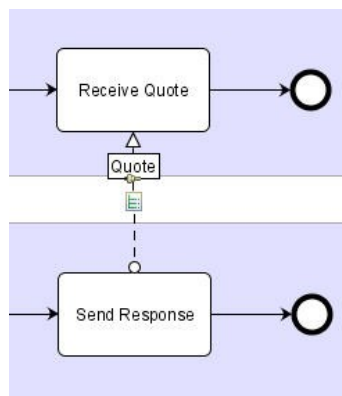
12. Click on the Navigator tab and, if necessary, expand the icon representing the schema tutorial_revised.
 - a. Drag the type intalio:RFQ from the schema listing onto the Quote Request link.
13. Click on the Connectors tab and expand the icon representing the internal connectors.
 - a. Drag the internal://webservice connector onto the Quote Request link.



14. Use the Sequence Flow Tool to create a link between the Return Response task and the Receive Quote task.
 - a. Use Quote as the label and change the Message Type to Response.



15. Click on the Navigator tab and expand the icon representing the schema tutorial_revised.
 - a. Drag the type intalio:Quote from the schema listing onto the Quote link.
16. Click on the Connectors tab and expand the icon representing the internal connectors.
 - a. Drag the internal://webservice connector onto the Quote link.

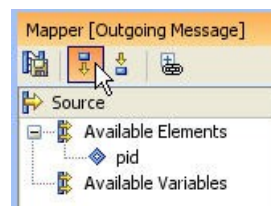


17. Validate and save the diagram.

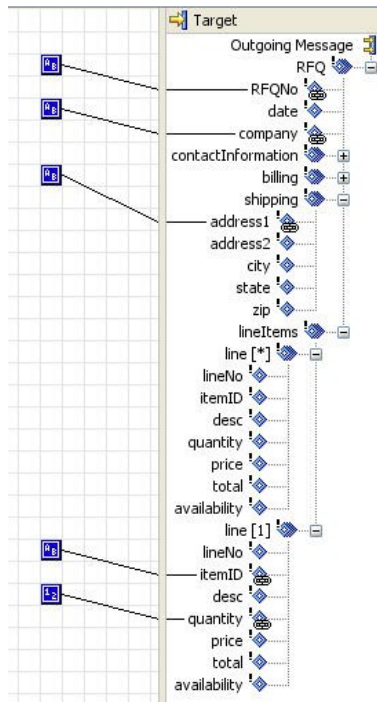
Defining the Mappings

You must now define how data will be mapped across the links. In the RFQ link, you will initialize several fields of the RFQ data type. In the Quote Request link, you will simply pass the RFQ data across the link. And in the Quote link, you will map data from the RFQ data type into the Quote data type; in addition, you will initialize other fields of the Quote data type.

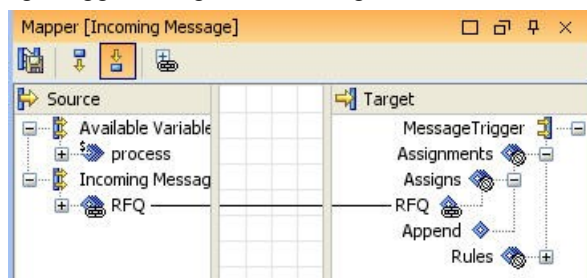
1. Use the Select Tool and highlight the RFQ link. Click on the Mapping tab and then click the Show Outgoing Message Mapping tool bar button.



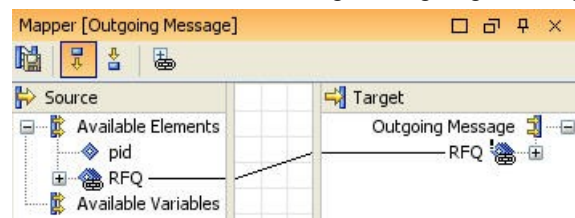
2. In the Target panel, expand the RFQ icon and supply values for the RFQNo, company, shipping/address1, line[*]/itemID, and line[*]/quantity entries. It does not matter what you enter as values.



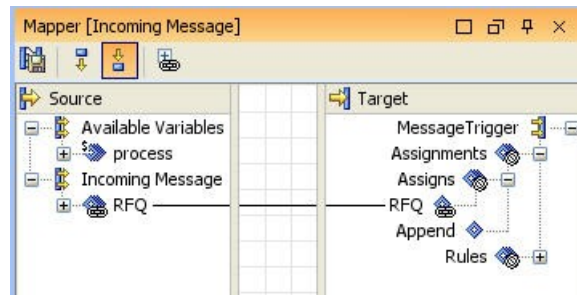
- a. Click the Apply Mapping to Properties tool bar button and then save the diagram.
3. Click on the Show Incoming Message Mapping tool bar button.
 - a. In the Target panel, right click on the Target/MessageTrigger/Assignments/Assigns icon and create a Complex Element named RFQ. Then map from Source/Incoming Message/RFQ to Target/ MessageTrigger/Assignments/Assigns/RFQ.



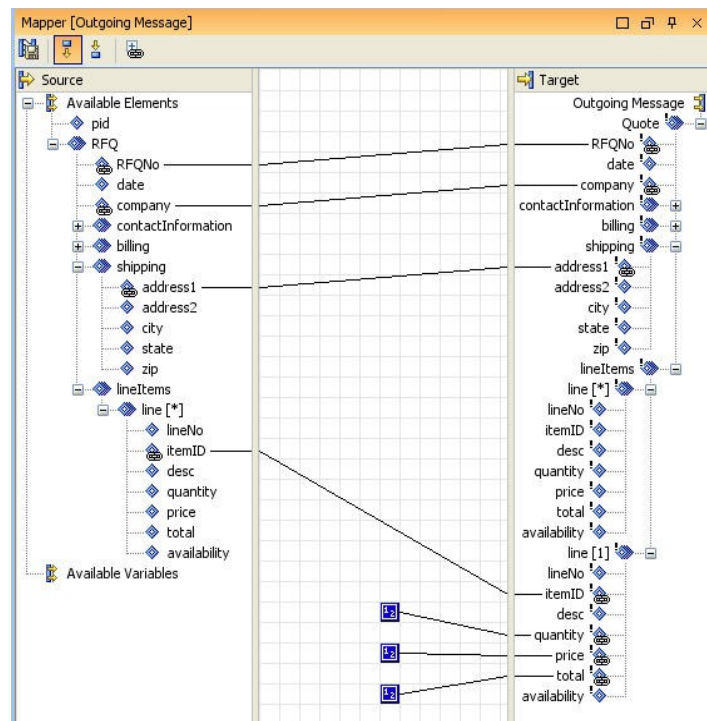
- b. Click the Apply Mapping to Properties tool bar button and then save the diagram.
4. Use the Select Tool and highlight the Quote Request link. Click on the Mapping tab and then click the Show Outgoing Message Mapping tool bar button.
5. Map from Source/Available Elements/RFQ to Target/Outgoing Message/RFQ.



- a. Click the Apply Mapping to Properties tool bar button and then save the diagram.
6. Click on the Show Incoming Message Mapping tool bar button.
 - a. In the Target panel, create a Complex Element named RFQ and map from Source/Incoming Message/RFQ to Target/Assignments/Assigns/RFQ.



- b. Click the Apply Mapping to Properties tool bar button and then save the diagram.
7. Use the Select Tool and highlight the Quote link. Click on the Mapping tab and then click the Show Outgoing Message Mapping tool bar button.
8. Map from the initialized fields in the Source/Available Elements/RFQ data type to the corresponding fields in Target/Outgoing Message/Quote.
 - a. Initialize the line[1]/quantity, line[1]/price, and line[2]/total fields; for example use quantity=4, price=100, and total=400.




- b. Click the Apply Mapping to Properties tool bar button and then save the diagram.

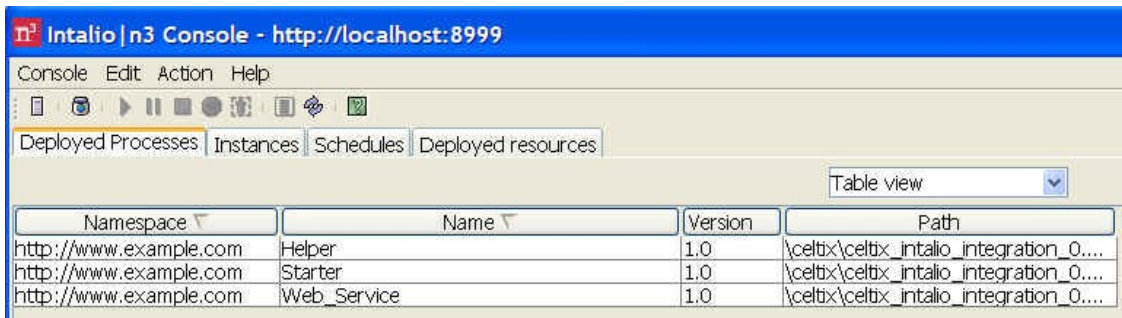
This completes the mapping assignments.


Deploying and Testing the Basic Application

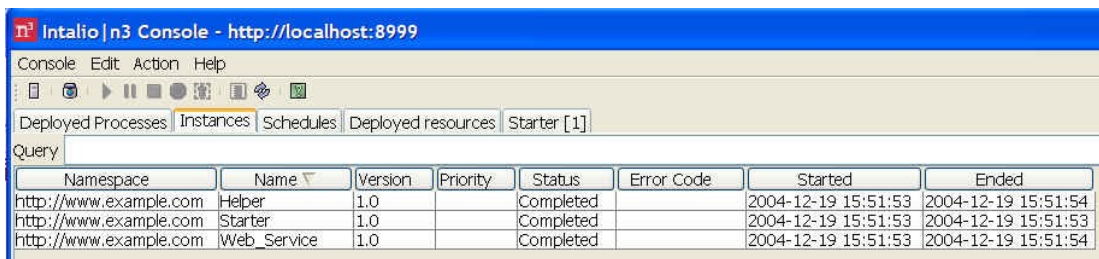
The completed application may be deployed into the Intalio|n3 Server and run using the Intalio|n3 Console. In addition, you will use the console to view the content of the messages exchanged between the process' tasks.

1. Start the Intalio|n3 Server.
2. Start the Intalio|n3 Console.
3. From the Intalio|n3 Designer, deploy the application.
4. On the console, select the Deployed Processes tab and click the Refresh current view  tool bar button.

- a. Confirm that there are three deployed processes.



- b. Select the Instances tab and click the Refresh current view tool bar button. Since the application has not been run, there are no process instances.
5. Select the Deployed Processes tab and highlight the entry corresponding to the Starter process and then click on the Start/resume selected item(s)  tool bar button.
6. Select the Instances tab and click the Refresh current view tool bar button. Note that all three processes have run to completion.



Reviewing the Message Content



You can review the data available to each process. Simply double click on an entry in the listing on the Instance tab. If you view the data available to the Web_Service process, you will observe both the RFQ and Quote. If you view the data available to the Helper process, you will observe only the RFQ. This is correct, as the Helper process initializes the Quote within the output message mapping of the Quote link.

Exposing the Basic Intalio|n3 Application as a Web Service

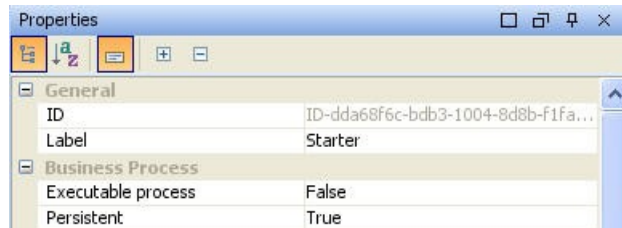
The process Starter/Starter Lane only serves as a point of reference for configuring the messaging protocols for access via a Web service. This process should not be deployed. To expose the application as a Web service, you only need to modify the existing diagram and redeploy the application. Then, rather than using the console to initiate the process flow, you will use a Web service client targeted to the start event in the Web Service/Service Lane process.

Since the processes in this modified application will have the same names as the original application, it is best to remove the original application from the server prior to deploying this modified application.

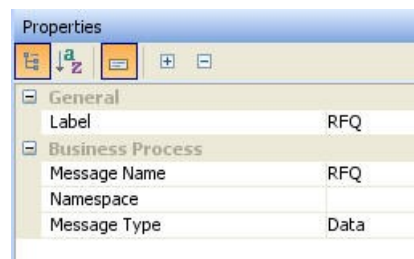
Modifying and Redeploying the Basic Intalio|n3 Application

1. Start the Intalio|n3 Server.
2. Start the Intalio|n3 Console.
 - a. If necessary, close any tabs that summarize the process data.
 - b. Select the Instances tab and click the Refresh current view tool bar button.
 - c. Shift-click to select all listed instances and delete by clicking the Destroy selected item(s)  tool bar button.
 - d. Select the Deployed Processes tab and click the Refresh current view tool bar button .

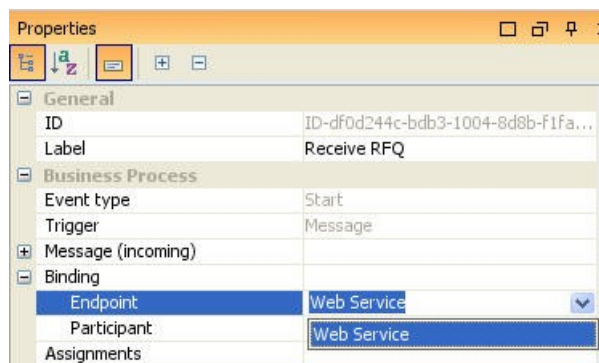
- e. Shift-click to select all listed processes and delete by clicking the Destroy selected item(s) tool bar button.
3. Start the Intalio/n3 Designer.
 - a. Open the celnix_intalio_integration_0.n3x workspace item.
 - b. Select the File > Save As... menu item and save the diagram with the name celnix_intalio_integration_1 (the .n3x suffix is added automatically). You will modify this copy of the application.
 4. Click the Starter/Starter Lane process with the Select Tool.
 - a. Set the pool's Executable process property is set to False.




5. Use the Select Tool and highlight the RFQ link.
 - a. Change the Message Type to Data.



6. Use the Select Tool and highlight the Receive RFQ start event.
 - a. Change the entry associated with the Endpoint/Binding property to Web Service.
 - b. Making this change disassociates the internal://webservice connector from the event.
 - c. The Receive RFQ start event becomes the target endpoint for your client Web service.



7. Save and validate the diagram twice.
 - a. The first time you validate the diagram an error and a warning will be reported.
 - b. The error will state: "The Web service message 'RFQ' will not work if its namespace is not the same as the diagram namespace. So it has been changed from " to the diagram namespace 'http://www.example.com'." Since the validation process has updated the namespace, this error will not appear when you re-validate the diagram.

- c. The warning will state: "The pool 'Starter' was not validated because its Executable property is set to False." This is what your application design intends and it is a correct warning.
 - d. Re-validate the diagram and confirm that the error has been eliminated.
8. Confirm that the server and console are running. If not, restart these Intalio components.
 9. From the Intalio Designer, deploy the application.
 10. On the console, select the Deployed Processes tab and click the Refresh current view  tool bar button.
 - a. Confirm that there are two deployed processes.
 - b. Select the Instances tab and click the Refresh current view tool bar button. Since the application has not been run, there are no process instances.

Running the Modified Basic Application

Since the Starter process is not deployed, you cannot kick off this process flow from the console. You must write a Web service client that sends a request to the endpoint that represents the Receive RFQ start event.

The Intalio Server will publish the WSDL file that describes the Intalio application deployed as Web service. You then use this WSDL file to write a client application.

1. Confirm that the Intalio Server and Console are running.
2. Open a Web browser and enter the following URL.

```
http://<hostName>:port/processes/Web_Service
```

Where `<hostName>` is the name of the computer running the Intalio Server and port is the TCP/IP port used by the server (8081 by default, but you may have changed this assignment when installing the server). `Web_Service` is the context name derived from the name of the pool containing the Web service endpoint, the start event Receive RFQ.

3. The browser displays the WSDL file of the Web service. Save the WSDL file to a local directory. In developing this note, the WSDL file was saved using the name `Web_Service.wsdl`.
4. Use the WSDL file to write a client application, as described in the following section.

The WSDL file includes the content of the `tutorial_revised` schema and describes two port types: `Web_Service` and `Helper`, which correspond to the two processes deployed from the designer. The `Web_Service` port type includes the `Receive_RFQ` operation, which is the operation your client application invokes to kick off the process flow.

At this stage of Celtix's product development, it is easiest if you edit the WSDL file and remove the service and port definitions that are not needed for your application. In a text editor, open the WSDL file and remove the `Helper_Service` and the `Web_Service_HttpsPort` definition from within the `Web_Service_Service` definition.

Writing the Celtix Client to the Basic Application

Now that you have a copy of the WSDL file that represents the Intalio process exposed as a Web service, you can generate starting point code for your Celtix client application. This document describes how to adapt the Celtix product demo directory hierarchy to this task. Alternatively, you could develop your application from within Eclipse.

At this stage of Celtix's product development, the starting point code will not include the client mainline code (or server mainline code or code for an implementation object, if you are trying to develop a server application). You will need to manually add these to a project. For the purposes of this note, the coding for these files is provided in the Appendix.

Setting up the Directory Hierarchy

Under the Celtix installation directory, create the directory `intalio_integration_client` under the `samples` subdirectory. Under the directory `intalio_integration_client`, create the subdirectories `wsdl`

and `src`. Under the directory `src`, create the subdirectory hierarchy `intalio.integration.client`.

Copy the WSDL file `Web_Service.wsdl` into the directory `intalio_integration/wsdl`. Copy the client mainline code into the directory `intalio.integration.client`, and copy the file `build.xml` into the directory `intalio_integration_client`.

Compiling and Running the Client Application

Confirm that the Intalio|n3 Server and Console applications are running.

Since the client mainline coding is complete and the `build.xml` file is specific to this application, you can compile the application as you would any of the other Celtix demos using the command `ant build`. To run the client application, first confirm that the Intalio|n3 Server is running and that your project has been deployed. Then run the client with the command `ant client`. Then use the Intalio|n3 Console to review the message content.

The Extended Intalio|n3 Application

Now that you have an Intalio|n3 process exposed as a Web service, you will extend the process diagram and integrate a Celtix Web service as the processing logic behind the Check Inventory task. This is very easy to do from the Intalio Designer once you have the WSDL file that describes the Web service. The types used within the basic Intalio|n3 process are defined in a schema file. In writing your WSDL file, you will use this same schema file as the content within the `<types>` section. This will make it easier to map data between the Intalio|n3 process and the Celtix Web service.

The Celtix Web Service WSDL File

Using the previously defined schema, complete the WSDL file. This file defines the `stockChecker` portType, which implements the `lineItems` operation. This operation accepts a collection of desired items, checks current inventory against the items listed in the request, and returns a collection in which multiple entries have been updated. The Celtix Web service will provide an implementation of this portType. A copy of the WSDL file is included in the Appendix to this document.

Note that the entire schema has been incorporated into the WSDL file, even though only the `lineItems` operation only used the `LineType` complex type.

In this document, this WSDL file has been named `intalio_integration.wsdl`.

Developing the Extended Application

There are three steps to extending the application.

1. Make the Celtix Web service WSDL file available to the Intalio|n3 Server.
2. Configure an Intalio|n3 WSDL connector.
3. Integrate the Celtix Web service into the Intalio|n3 process. This involves creating the fourth process pool, which represents the Celtix Web service, and mapping the data flow between the Check Inventory task and the external Web service.

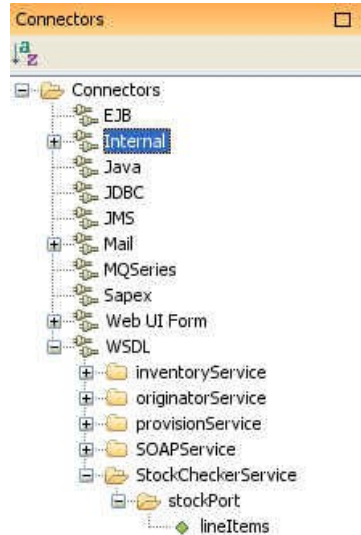
Note that all three of these steps can be completed using only the WSDL file that describes the target Web service. The target application is not needed until the you actually run the process flow.

Making the WSDL File Available to the Intalio|n3 Server

Place a copy of the Celtix Web service WSDL file into the `<Intalio|n3 server installation directory>\wsdl` directory. You will use this copy of the WSDL file to configure the Intalio|n3 WSDL connector.

Configure an Intalio|n3 WSDL Connector

Start the Intalio|n3 Designer and open your project. Click on the connectors tab, which displays the panel used to configure connectors. Highlight the WSDL entry, right click, and select Configure from the popup menu. Click the Import command button and browse to the file `intalio_integration.wsdl` that you just copied into the directory `<Intalio|n3 server installation directory>\wsdl`. Click the Open command button; this places an entry corresponding to this WSDL file in the Referenced WSDL services list box. Finally, click the OK command button. Note that the name of the entry corresponding to the Celtix Web service (StockCheckerService) corresponds to the service name assigned in the WSDL file.

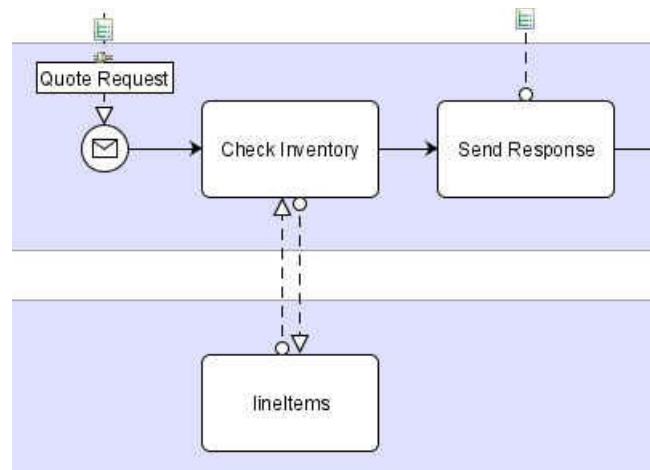


Integrating the StockCheckerService into the Business Process

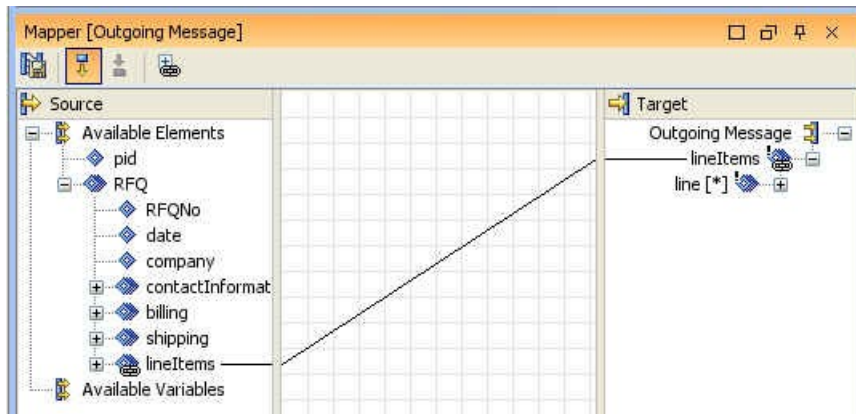
The StockCheckerService provides the processing logic behind the Check Inventory task. It accepts a collection of line items from the request and returns another collection of line items in which the availability, quantity and total fields have been updated.

Since the StockCheckerService provides processing logic to the Helper process, you want to create the pool and lane that encapsulate this service adjacent to the Helper process's pool and lane.

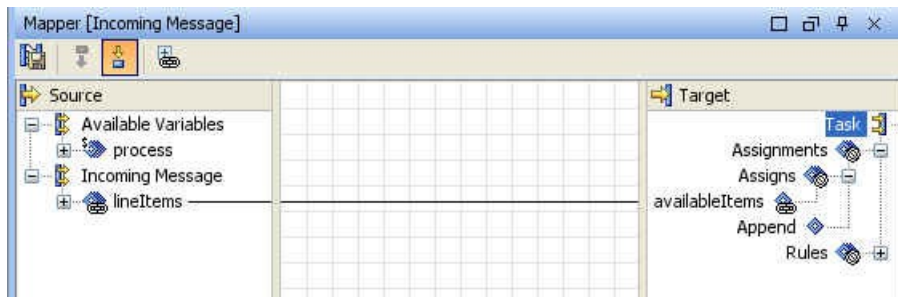
1. In the designer, click on the Connectors tab and then drag the StockCheckerService icon onto the Helper/Helper Lane process.
2. Change the name of the lane to Inventory Check.
3. Now drag the icon representing the `lineItems` operation onto the lane Inventory Check. This creates a new task named `lineItems`.
4. Use the Sequence Flow Connector tool to create a link from the Helper/Helper Lane/Check Inventory task to the `stockPort/Inventory Check/lineItems` task.
5. Create a return link from the `stockPort/Inventory Check/lineItems` task to the Helper/Helper Lane/Check Inventory task.



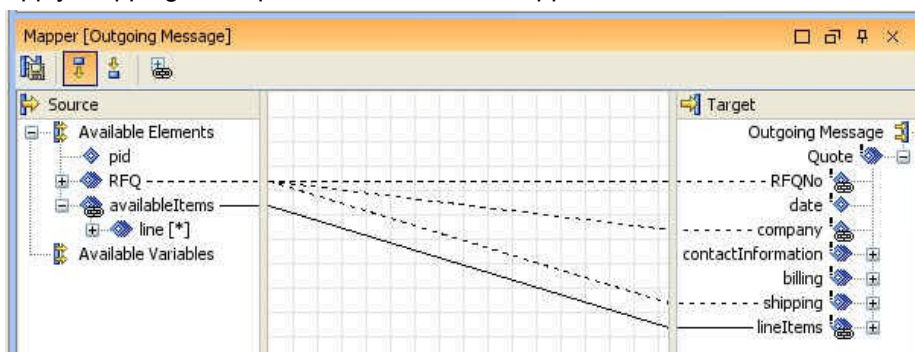
- Click on the Check Inventory to lineItems link. In the Mapper view, connect the Source/Available Elements/RFQ/lineItems element to the Target/Outgoing Message/lineItems element, and then click on the Apply Mapping to Properties button in the Mapper's title bar.



- Click on the lineItems to Check Inventory link. In the Mapper view's target panel, right click on the Task>Assignments>Assigns element and select Create Element>Create Complex Element from the popup menu to open the Schema Object Property dialog box. Enter availableItems in the Element name field and click the OK command button.
- Connect the Source/Available Variables/Incoming Message/lineItems element to the Task/Assignments/Assigns/availableItems element, and then click on the Apply Mapping to Properties button in the Mapper's title bar.



- Select the Quote link and open the associated Mapper view. Currently, elements from the RFQ as well as static content are mapped to the outgoing Quote message.
- Delete the static mappings and the itemID mapping and create a new mapping from Source/Available Elements/availableItems to Target/Outgoing Message/Quote/lineItems. Then click on the Apply Mapping to Properties button in the Mapper's title bar.



Through these steps, the Check Inventory task has been configured to invoke on the Celtix Web service, sending a collection of desired line items. The Celtix Web service will check current inventory levels and update the number available and the total cost for each line items. The Web service then returns the modified collection to the Check Inventory task. You then temporarily store the modified collection in a process variable and as part of the following task you replace the original line items collection in the RFQ with the modified collection returned from the Web service.

Writing the Celtix Web Service

With the WSDL file that represents the Celtix Web service, you can generate starting point code for your Celtix server application. This document describes how to adapt the Celtix product demo directory hierarchy to this task. Alternatively, you could develop your application from within Eclipse.

In this section, you will create both the server process and a client application that may be used to test the server's processing logic.

At this stage of Celtix's product development, the starting point code will not include the server mainline code or code for an implementation object. You will need to manually add these to a project. For the purposes of this note, the coding for these files is provided in the Appendix.

Setting up the Directory Hierarchy

Under the Celtix installation directory, create the directory `intalio_integration_server` under the `samples` subdirectory. Under the directory `intalio_integration_server`, create the subdirectories `wSDL` and `src`. Under the directory `src`, create the subdirectory hierarchies `intalio.integration.server` and `intalio.integration.client`.

Copy the WSDL file `intalio_integration.wSDL` into the directory `intalio_integration_server/wSDL`. Copy the client mainline code into the directory `intalio.integration.client`, copy the server mainline and implementation object code into the directory `intalio.integration.server`, and copy the file `build.xml` into the directory `intalio_integration_server`.

Compiling and Running the Server and Test Client Applications

Since the coding is complete and the `build.xml` file is specific to this application, you can compile the application as you would any of the other Celtix demos using the command `ant build`.

Testing the Server Application

To run the server application, use the command `ant server`. Then run the client with the command `ant client`. The command windows will document what is happening: the client sends a collection of line items to the server; for the first line item, the server replaces the availability message, the number of items available, and the total cost; and the client then displays the information contained in the returned collection of line items.

Running the Entire Extended Intalio|n3 Application

You now have all of the pieces needed to run the entire extended Intalio|n3 application: the Intalio|n3 extended process; the Celtix Web service; and the Celtix client that initiates the Intalio|n3 process.

1. Confirm that the Celtix Web service is running. If not, issue the command `ant server` from the directory `intalio_integration_server`.
2. Confirm that the Intalio|n3 Server and Console components are running.
3. Run the Celtix client application by issuing the command `ant client` from the directory `intalio_integration_client`.

Once the process flow completes, use the Intalio|n3 console to view the messages available to the Helper process. You should be able to confirm that the values of the availability, number, and total fields in the first line item have been changed.

Now view the messages available to the `Web_Service` process. Note that the values of the availability, number, and total fields for the first line item in the Quote data type have been set to the revised values.

Appendix

The Basic Application Files

The file `tutorial_revised.xsd`

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.intalio.com/"
            xmlns:intalio="http://www.intalio.com/">

  <!-- Request for Quote -->
  <xsd:element name="RFQ">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="intalio:RFQNo"/>
        <xsd:element ref="intalio:date"/>
        <xsd:element ref="intalio:company"/>
        <xsd:element ref="intalio:contactInformation"/>
        <xsd:element ref="intalio:billing"/>
        <xsd:element ref="intalio:shipping"/>
        <xsd:element ref="intalio:lineItems"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <!-- Quote Response -->
  <xsd:element name="Quote">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="intalio:RFQNo"/>
        <xsd:element ref="intalio:date"/>
        <xsd:element ref="intalio:company"/>
        <xsd:element ref="intalio:contactInformation"/>
        <xsd:element ref="intalio:billing"/>
        <xsd:element ref="intalio:shipping"/>
        <xsd:element ref="intalio:lineItems"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <!-- Quote Response -->
  <xsd:element name="NA">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="intalio:RFQNo"/>
        <xsd:element ref="intalio:date"/>
        <xsd:element ref="intalio:company"/>
        <xsd:element ref="intalio:contactInformation"/>
        <xsd:element ref="intalio:billing"/>
        <xsd:element ref="intalio:shipping"/>
        <xsd:element ref="intalio:lineItems"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <!-- Purchase Order -->
  <xsd:element name="PO">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="intalio:PONo"/>
        <xsd:element ref="intalio:date"/>
        <xsd:element ref="intalio:company"/>
        <xsd:element ref="intalio:contactInformation"/>
        <xsd:element ref="intalio:billing"/>
        <xsd:element ref="intalio:shipping"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

```

```

    <xsd:element ref="intalio:lineItems"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:element>

<!-- Sales Order -->
<xsd:element name="SO">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="intalio:SONo"/>
      <xsd:element ref="intalio:date"/>
      <xsd:element ref="intalio:company"/>
      <xsd:element ref="intalio:contactInformation"/>
      <xsd:element ref="intalio:billing"/>
      <xsd:element ref="intalio:shipping"/>
      <xsd:element ref="intalio:lines"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<!-- Pack List -->

<!-- Bill of Lading -->

<!-- Invoice -->
<xsd:element name="Invoice">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="intalio:InvoiceNo"/>
      <xsd:element ref="intalio:date"/>
      <xsd:element ref="intalio:SONo"/>
      <xsd:element ref="intalio:PONo"/>
      <xsd:element ref="intalio:company"/>
      <xsd:element ref="intalio:contactInformation"/>
      <xsd:element ref="intalio:billing"/>
      <xsd:element ref="intalio:invoiceLines"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<!-- ===== -->
<!-- REFERENCES -->
<!-- ===== -->

<xsd:element name="contactInformation" type="intalio:contactType"/>
<xsd:element name="billing" type="intalio:billingType"/>
<xsd:element name="shipping" type="intalio:shippingType"/>
<xsd:element name="lineItems" type="intalio:lineItemType"/>
<xsd:element name="lines" type="intalio:linesType"/>
<xsd:element name="delivery" type="intalio:delType"/>
<xsd:element name="invoiceLines" type="intalio:invoiceType"/>

<!-- Billing Address -->
<xsd:complexType name="billingType">
  <xsd:sequence>
    <xsd:element ref="intalio:address1" />
    <xsd:element ref="intalio:address2" />
    <xsd:element ref="intalio:city" />
    <xsd:element ref="intalio:state" />
    <xsd:element ref="intalio:zip" />
  </xsd:sequence>
</xsd:complexType>

<!-- Shipping Address -->
<xsd:complexType name="shippingType">
  <xsd:sequence>
    <xsd:element ref="intalio:address1"/>
    <xsd:element ref="intalio:address2"/>
    <xsd:element ref="intalio:city"/>
    <xsd:element ref="intalio:state"/>
    <xsd:element ref="intalio:zip"/>
  </xsd:sequence>
</xsd:complexType>

```

```

</xsd:sequence>
</xsd:complexType>

<!-- Contact Details -->
<xsd:complexType name="contactType">
  <xsd:sequence>
    <xsd:element ref="intalio:firstName"/>
    <xsd:element ref="intalio:lastName" />
    <xsd:element ref="intalio:fax" />
    <xsd:element ref="intalio:phone" />
    <xsd:element ref="intalio:email" />
  </xsd:sequence>
</xsd:complexType>

<!-- PO and RFQ Lines -->
<xsd:complexType name="lineItemType">
  <xsd:sequence>
    <xsd:element ref="intalio:line" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="line" type="intalio:lineType"/>

<!-- PO And RFQ Line Item Details -->
<xsd:complexType name="lineType">
  <xsd:sequence>
    <xsd:element ref="intalio:lineNo" />
    <xsd:element ref="intalio:itemID" />
    <xsd:element ref="intalio:desc"/>
    <xsd:element ref="intalio:quantity"/>
    <xsd:element ref="intalio:price" />
    <xsd:element ref="intalio:total"/>
    <xsd:element ref="intalio:availability"/>
  </xsd:sequence>
</xsd:complexType>

<!-- SO Lines -->
<xsd:complexType name="linesType">
  <xsd:sequence>
    <xsd:element name="line" type="intalio:SOLineType" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<!-- SO Line Item Details -->
<xsd:complexType name="SOLineType">
  <xsd:sequence>
    <xsd:element ref="intalio:lineNo" />
    <xsd:element ref="intalio:itemID" />
    <xsd:element ref="intalio:desc"/>
    <xsd:element ref="intalio:quantity"/>
    <xsd:element ref="intalio:price" />
    <xsd:element ref="intalio:total"/>
    <xsd:element ref="intalio:delivery"/>
  </xsd:sequence>
</xsd:complexType>

<!-- SO Delivery -->
<xsd:complexType name="delType">
  <xsd:sequence>
    <xsd:element name="del" type="xsd:int"/>
    <xsd:element name="delDate" type="xsd:date"/>
    <xsd:element name="qty" type="xsd:int"/>
    <xsd:element name="requestDate" type="xsd:date"/>
    <xsd:element name="promiseDate" type="xsd:date"/>
  </xsd:sequence>
</xsd:complexType>

<!-- Invoice Lines -->
<xsd:complexType name="invoiceType">
  <xsd:sequence>
    <xsd:element ref="intalio:lineNo"/>

```

```

    <xsd:element ref="intalio:itemID"/>
    <xsd:element ref="intalio:desc"/>
    <xsd:element ref="intalio:quantity"/>
    <xsd:element ref="intalio:price"/>
    <xsd:element ref="intalio:total"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="RFQNo" type="xsd:string"/>
<xsd:element name="PONo" type="xsd:string"/>
<xsd:element name="SONo" type="xsd:string"/>
<xsd:element name="InvoiceNo" type="xsd:string"/>
<xsd:element name="date" type="xsd:date"/>
<xsd:element name="company" type="xsd:string"/>
<xsd:element name="firstName" type="xsd:string"/>
<xsd:element name="lastName" type="xsd:string"/>
<xsd:element name="address1" type="xsd:string"/>
<xsd:element name="address2" type="xsd:string"/>
<xsd:element name="desc" type="xsd:string"/>
<xsd:element name="city" type="xsd:string"/>
<xsd:element name="state" type="xsd:string"/>
<xsd:element name="zip" type="xsd:string"/>
<xsd:element name="fax" type="xsd:string"/>
<xsd:element name="phone" type="xsd:string"/>
<xsd:element name="email" type="xsd:string"/>
<xsd:element name="lineNo" type="xsd:int"/>
<xsd:element name="itemID" type="xsd:string"/>
<xsd:element name="quantity" type="xsd:decimal"/>
<xsd:element name="price" type="xsd:decimal"/>
<xsd:element name="total" type="xsd:decimal"/>
<xsd:element name="availability" type="xsd:string"/>
</xsd:schema>

```

The file build.xml

```

<?xml version="1.0"?>
<project name="intalio integration demo" default="build" basedir=".">

  <import file="../common_build.xml"/>

  <target name="client" description="run demo client">
    <property name="param" value=""/>
    <celtixrun classname="intalio.integration.client.Client"
      param1="\${basedir}/wsdl/Web_Service.wsdl"
      param2="\${op}" param3="\${param}"/>
  </target>

  <target name="generate.code">
    <echo level="info" message="Generating code using wsdl2java..."/>
    <wsdl2java file="Web_Service.wsdl"/>
  </target>

</project>

```

The client mainline

```

package intalio.integration.client;

import java.io.File;
import java.net.URL;
import java.util.ArrayList;
import java.math.BigDecimal;
import java.util.GregorianCalendar;
import javax.xml.datatype.DatatypeFactory;
import javax.xml.namespace.QName;
import javax.xml.datatype.XMLGregorianCalendar;

import com.example.WebService;
import com.example.WebServiceService;

```

```

import com.intalio.RFQ;
import com.intalio.ContactType;
import com.intalio.BillingType;
import com.intalio.ShippingType;
import com.intalio.LineItemType;
import com.intalio.LineType;

public final class Client {

    private static final QName SERVICE_NAME
        = new QName("http://www.example.com", "Web_Service_Service");

    private Client() {
    }

    public static void main(String args[]) throws Exception {

        if (args.length == 0) {
            System.out.println("please specify wsdl");
            System.exit(1);
        }

        URL wsdlURL;
        File wsdlFile = new File(args[0]);
        if (wsdlFile.exists()) {
            wsdlURL = wsdlFile.toURL();
        } else {
            wsdlURL = new URL(args[0]);
        }

        // create a proxy for the target service
        WebServiceService s = new WebServiceService(wsdlURL, SERVICE_NAME);
        WebService proxy = s.getWebServiceHttpPort();

        // create an instance of the RFQ
        RFQ rfq = new RFQ();

        // initialize the RFQ
        rfq.setRFQNo("1001");

        javax.xml.datatype.DatatypeFactory datatypeFactory =
            javax.xml.datatype.DatatypeFactory.newInstance();
        XMLGregorianCalendar c =
            datatypeFactory.newXMLGregorianCalendar(new GregorianCalendar());
        rfq.setDate(c);

        rfq.setCompany("ObjectWeb");

        ContactType contact = new ContactType();
        contact.setFirstName("Harry");
        contact.setLastName("Potter");
        contact.setFax("781-902-8001");
        contact.setPhone("781-902-8000");
        contact.setEmail("harry.potter@objectweb.org");
        rfq.setContactInformation(contact);

        BillingType billing = new BillingType();
        billing.setAddress1("INRIA - ZIRST");
        billing.setAddress2("655 avenue de l'Europe");
        billing.setCity("Montbonnot");
        billing.setState("FRANCE");
        billing.setZip("38334 SAINT-ISMIER Cedex");
        rfq.setBilling(billing);

        ShippingType shipping = new ShippingType();
        shipping.setAddress1("INRIA - ZIRST");
        shipping.setAddress2("655 avenue de l'Europe");
        shipping.setCity("Montbonnot");
        shipping.setState("FRANCE");
        shipping.setZip("38334 SAINT-ISMIER Cedex");
        rfq.setShipping(shipping);
    }
}

```

```

// populate two line items
LineItemType lineItems = new LineItemType();
LineType lt = new LineType();
lt.setLineNo(1);
lt.setItemID("1001");
lt.setDesc("CPU chip");
lt.setQuantity(new BigDecimal(5));
lt.setPrice(new BigDecimal(100));
lt.setTotal(new BigDecimal(500));
lt.setAvailability("available");
lineItems.getLine().add(lt);
lt.setLineNo(2);
lt.setItemID("2001");
lt.setDesc("Memory chip");
lt.setQuantity(new BigDecimal(15));
lt.setPrice(new BigDecimal(20));
lt.setTotal(new BigDecimal(300));
lt.setAvailability("available");
lineItems.getLine().add(lt);
rfq.setLineItems(lineItems);

// invoke the receiveRFQ operation
// this is a oneway operation that kicks off the
// Intalio\|n3 process flow
System.out.println("Invoking receiveRFQ operation");
proxy.receiveRFQ(rfq);
System.out.println("\tOperation complete");

System.exit(0);
}
}

```

The Extended Application Files

The file `intalio_integration.wsdl`

```

<wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://www.intalio.com/intalio"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsd1="http://www.intalio.com/"
targetNamespace="http://www.intalio.com/" name="intalio_integration.wsdl">
  <wsdl:types>
    <schema targetNamespace="http://www.intalio.com/"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
elementFormDefault="qualified">
      <!-- Request for Quote -->
      <element name="RFQ">
        <complexType>
          <sequence>
            <element ref="xsd1:RFQNo"/>
            <element ref="xsd1:date"/>
            <element ref="xsd1:company"/>
            <element ref="xsd1:contactInformation"/>
            <element ref="xsd1:billing"/>
            <element ref="xsd1:shipping"/>
            <element ref="xsd1:lineItems"/>
          </sequence>
        </complexType>
      </element>
      <!-- Quote Response -->
      <element name="Quote">
        <complexType>
          <sequence>

```

```

        <element ref="xsd1:RFQNo"/>
        <element ref="xsd1:date"/>
        <element ref="xsd1:company"/>
        <element ref="xsd1:contactInformation"/>
        <element ref="xsd1:billing"/>
        <element ref="xsd1:shipping"/>
        <element ref="xsd1:lineItems"/>
    </sequence>
</complexType>
</element>
<!-- Quote Response -->
<element name="NA">
    <complexType>
        <sequence>
            <element ref="xsd1:RFQNo"/>
            <element ref="xsd1:date"/>
            <element ref="xsd1:company"/>
            <element ref="xsd1:contactInformation"/>
            <element ref="xsd1:billing"/>
            <element ref="xsd1:shipping"/>
            <element ref="xsd1:lineItems"/>
        </sequence>
    </complexType>
</element>
<!-- Purchase Order -->
<element name="PO">
    <complexType>
        <sequence>
            <element ref="xsd1:PONo"/>
            <element ref="xsd1:date"/>
            <element ref="xsd1:company"/>
            <element ref="xsd1:contactInformation"/>
            <element ref="xsd1:billing"/>
            <element ref="xsd1:shipping"/>
            <element ref="xsd1:lineItems"/>
        </sequence>
    </complexType>
</element>
<!-- Sales Order -->
<element name="SO">
    <complexType>
        <sequence>
            <element ref="xsd1:SONo"/>
            <element ref="xsd1:date"/>
            <element ref="xsd1:company"/>
            <element ref="xsd1:contactInformation"/>
            <element ref="xsd1:billing"/>
            <element ref="xsd1:shipping"/>
            <element ref="xsd1:lines"/>
        </sequence>
    </complexType>
</element>
<!-- Invoice -->
<element name="Invoice">
    <complexType>
        <sequence>
            <element ref="xsd1:InvoiceNo"/>
            <element ref="xsd1:date"/>
            <element ref="xsd1:SONo"/>
            <element ref="xsd1:PONo"/>
            <element ref="xsd1:company"/>
            <element ref="xsd1:contactInformation"/>
            <element ref="xsd1:billing"/>
            <element ref="xsd1:invoiceLines"/>
        </sequence>
    </complexType>
</element>
<!-- ===== -->
<!-- REFERENCES -->
<!-- ===== -->
<element name="contactInformation" type="xsd1:contactType"/>

```



```

<element name="billing" type="xsd1:billingType"/>
<element name="shipping" type="xsd1:shippingType"/>
<element name="lineItems" type="xsd1:lineItemType"/>
<element name="lines" type="xsd1:linesType"/>
<element name="delivery" type="xsd1:delType"/>
<element name="invoiceLines" type="xsd1:invoiceType"/>
<!-- Billing Address -->
<complexType name="billingType">
  <sequence>
    <element ref="xsd1:address1"/>
    <element ref="xsd1:address2"/>
    <element ref="xsd1:city"/>
    <element ref="xsd1:state"/>
    <element ref="xsd1:zip"/>
  </sequence>
</complexType>
<!-- Shipping Address -->
<complexType name="shippingType">
  <sequence>
    <element ref="xsd1:address1"/>
    <element ref="xsd1:address2"/>
    <element ref="xsd1:city"/>
    <element ref="xsd1:state"/>
    <element ref="xsd1:zip"/>
  </sequence>
</complexType>
<!-- Contact Details -->
<complexType name="contactType">
  <sequence>
    <element ref="xsd1:firstName"/>
    <element ref="xsd1:lastName"/>
    <element ref="xsd1:fax"/>
    <element ref="xsd1:phone"/>
    <element ref="xsd1:email"/>
  </sequence>
</complexType>
<!-- PO and RFQ Lines -->
<complexType name="lineItemType">
  <sequence>
    <element maxOccurs="unbounded" name="line"
      type="xsd1:lineType"/>
  </sequence>
</complexType>
<!-- PO And RFQ Line Item Details -->
<complexType name="lineType">
  <sequence>
    <element ref="xsd1:lineNo"/>
    <element ref="xsd1:itemID"/>
    <element ref="xsd1:desc"/>
    <element ref="xsd1:quantity"/>
    <element ref="xsd1:price"/>
    <element ref="xsd1:total"/>
    <element ref="xsd1:availability"/>
  </sequence>
</complexType>
<!-- SO Lines -->
<complexType name="linesType">
  <sequence>
    <element maxOccurs="unbounded" name="line"
      type="xsd1:SOLineType"/>
  </sequence>
</complexType>
<!-- SO Line Item Details -->
<complexType name="SOLineType">
  <sequence>
    <element ref="xsd1:lineNo"/>
    <element ref="xsd1:itemID"/>
    <element ref="xsd1:desc"/>
    <element ref="xsd1:quantity"/>
    <element ref="xsd1:price"/>
    <element ref="xsd1:total"/>
  </sequence>

```

```

        <element ref="xsd1:delivery"/>
    </sequence>
</complexType>
<!-- SO Delivery -->
<complexType name="delType">
    <sequence>
        <element name="del" type="int"/>
        <element name="delDate" type="date"/>
        <element name="qty" type="int"/>
        <element name="requestDate" type="date"/>
        <element name="promiseDate" type="date"/>
    </sequence>
</complexType>
<!-- Invoice Lines -->
<complexType name="invoiceType">
    <sequence>
        <element ref="xsd1:lineNo"/>
        <element ref="xsd1:itemID"/>
        <element ref="xsd1:desc"/>
        <element ref="xsd1:quantity"/>
        <element ref="xsd1:price"/>
        <element ref="xsd1:total"/>
    </sequence>
</complexType>
<element name="RFQNo" type="string"/>
<element name="PONo" type="string"/>
<element name="SONo" type="string"/>
<element name="InvoiceNo" type="string"/>
<element name="date" type="date"/>
<element name="company" type="string"/>
<element name="firstName" type="string"/>
<element name="lastName" type="string"/>
<element name="address1" type="string"/>
<element name="address2" type="string"/>
<element name="desc" type="string"/>
<element name="city" type="string"/>
<element name="state" type="string"/>
<element name="zip" type="string"/>
<element name="fax" type="string"/>
<element name="phone" type="string"/>
<element name="email" type="string"/>
<element name="lineNo" type="int"/>
<element name="itemID" type="string"/>
<element name="quantity" type="decimal"/>
<element name="price" type="decimal"/>
<element name="total" type="decimal"/>
<element name="availability" type="string"/>
</schema>
</wsdl:types>
<message name="lineItemsRequest">
    <part name="requestedItems" element="xsd1:lineItems"/>
</message>
<message name="lineItemsResponse">
    <part name="inStockItems" element="xsd1:lineItems"/>
</message>
<portType name="stockChecker">
    <operation name="lineItems">
        <input name="lineItemsRequest"
            message="xsd1:lineItemsRequest"/>
        <output name="lineItemsResponse"
            message="xsd1:lineItemsResponse"/>
    </operation>
</portType>
<binding name="stockCheckerSOAPBinding" type="xsd1:stockChecker">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="lineItems">
        <soap:operation style="document"/>
        <input>
            <soap:body use="literal"/>
        </input>
    </operation>

```

```

        <output>
            <soap:body use="literal"/>
        </output>
    </operation>
</binding>
<service name="StockCheckerService">
    <port name="stockPort" binding="xsd:stockCheckerSOAPBinding">
        <soap:address
            location="http://localhost:8888/StockCheckerService"/>
    </port>
</service>
</wsdl:definitions>

```

The file build.xml

```

<?xml version="1.0"?>
<project name="intalio integration demo" default="build" basedir=".">

    <import file="../common_build.xml"/>

    <target name="client" description="run demo client">
        <property name="param" value=""/>
        <celtixrun classname="intalio.integration.client.Client"
            param1="${basedir}/wsdl/intalio_integration.wsdl"
            param2="{op}"
            param3="{param}"/>
    </target>

    <target name="server" description="run demo server">
        <celtixrun classname="intalio.integration.server.Server"
            param1="${basedir}/wsdl/intalio_integration.wsdl"/>
    </target>

    <target name="generate.code">
        <echo level="info" message="Generating code using wsdl2java..."/>
        <wsdl2java file="intalio_integration.wsdl"/>
    </target>

</project>

```

The test client mainline

```

package intalio.integration.client;

import java.io.File;
import java.net.URL;
import java.util.ArrayList;
import java.math.BigDecimal;
import java.util.GregorianCalendar;
import javax.xml.datatype.DatatypeFactory;
import javax.xml.namespace.QName;
import javax.xml.datatype.XMLGregorianCalendar;

import com.example.WebService;
import com.example.WebServiceService;
import com.intalio.RFQ;
import com.intalio.ContactType;
import com.intalio.BillingType;
import com.intalio.ShippingType;
import com.intalio.LineItemType;
import com.intalio.LineType;

public final class Client {

    private static final QName SERVICE_NAME
        = new QName("http://www.example.com", "Web_Service_Service");

    private Client() {
    }

```

```

public static void main(String args[]) throws Exception {

    if (args.length == 0) {
        System.out.println("please specify wsdl");
        System.exit(1);
    }

    URL wsdlURL;
    File wsdlFile = new File(args[0]);
    if (wsdlFile.exists()) {
        wsdlURL = wsdlFile.toURL();
    } else {
        wsdlURL = new URL(args[0]);
    }

    // create a proxy for the target service
    WebServiceService s = new WebServiceService(wsdlURL, SERVICE_NAME);
    WebService proxy = s.getWebServiceHttpPort();

    // create an instance of the RFQ
    RFQ rfq = new RFQ();

    // initialize the RFQ
    rfq.setRFQNo("1001");

    javax.xml.datatype.DatatypeFactory datatypeFactory =
        javax.xml.datatype.DatatypeFactory.newInstance();
    XMLGregorianCalendar c =
        datatypeFactory.newXMLGregorianCalendar(new GregorianCalendar());
    rfq.setDate(c);

    rfq.setCompany("ObjectWeb");

    ContactType contact = new ContactType();
    contact.setFirstName("Harry");
    contact.setLastName("Potter");
    contact.setFax("781-902-8001");
    contact.setPhone("781-902-8000");
    contact.setEmail("harry.potter@objectweb.org");
    rfq.setContactInformation(contact);

    BillingType billing = new BillingType();
    billing.setAddress1("INRIA - ZIRST");
    billing.setAddress2("655 avenue de l'Europe");
    billing.setCity("Montbonnot");
    billing.setState("FRANCE");
    billing.setZip("38334 SAINT-ISMIER Cedex");
    rfq.setBilling(billing);

    ShippingType shipping = new ShippingType();
    shipping.setAddress1("INRIA - ZIRST");
    shipping.setAddress2("655 avenue de l'Europe");
    shipping.setCity("Montbonnot");
    shipping.setState("FRANCE");
    shipping.setZip("38334 SAINT-ISMIER Cedex");
    rfq.setShipping(shipping);

    // populate two line items
    LineItemType lineItems = new LineItemType();
    LineType lt = new LineType();
    lt.setLineNo(1);
    lt.setItemID("1001");
    lt.setDesc("CPU chip");
    lt.setQuantity(new BigDecimal(5));
    lt.setPrice(new BigDecimal(100));
    lt.setTotal(new BigDecimal(500));
    lt.setAvailability("available");
    lineItems.getLine().add(lt);

    LineType lt2 = new LineType();
    lt2.setLineNo(2);

```

```

        lt2.setItemID("2001");
        lt2.setDesc("Memory chip");
        lt2.setQuantity(new BigDecimal(15));
        lt2.setPrice(new BigDecimal(20));
        lt2.setTotal(new BigDecimal(300));
        lt2.setAvailability("available");
        lineItems.getLine().add(lt2);
        rfq.setLineItems(lineItems);

        // invoke the receiverRFQ operation
        // this is a oneway operation that kicks off the
        // Intalio\|n3 process flow
        System.out.println("Invoking receiverRFQ operation");
        proxy.receiverRFQ(rfq);
        System.out.println("\tOperation complete");

        System.exit(0);
    }
}

```

The server mainline

```

package intalio.integration.server;

import javax.xml.ws.Endpoint;

public class Server {

    protected Server() throws Exception {
        System.out.println("Starting Server");

        Object implementor = new StockCheckerImpl();
        String address = "http://localhost:8888/StockCheckerService";
        Endpoint.publish(address, implementor);
    }

    public static void main(String args[]) throws Exception {
        new Server();
        System.out.println("Server ready...");

        Thread.sleep(5 * 60 * 1000);
        System.out.println("Server exiting");
        System.exit(0);
    }
}

```

The StockCheckerImpl code

```

package intalio.integration.server;

import java.math.BigDecimal;
import java.util.logging.Logger;

import java.util.ArrayList;
import java.util.List;
import javax.xml.ws.Holder;

import com.intalio.LineType;
import com.intalio.StockChecker;

@javax.jws.WebService(name = "StockChecker", serviceName = "StockCheckerService",
    targetNamespace = "http://www.intalio.com/",
    wsdlLocation = "file:./wsdl/intalio_integration.wsdl")
public class StockCheckerImpl implements StockChecker {

    private static final Logger LOG =

```

```
        Logger.getLogger(StockCheckerImpl.class.getPackage().getName());

    public void lineItems(Holder<List<LineType>> line) {
        List<LineType> lines = line.value;

        if (!lines.isEmpty()) {
            System.out.println("Checking Inventory");
            System.out.println("\tDesired items: ");
            System.out.println("\t\tID: " + lines.get(0).getItemID());
            System.out.println("\t\tDesc.: " + lines.get(0).getDesc());
            System.out.println("\t\tQuantity: " + lines.get(0).getQuantity());
            lines.get(0).setAvailability("not available");
            lines.get(0).setQuantity(new BigDecimal(0));
            lines.get(0).setTotal(new BigDecimal(0));
        } else {
            System.out.println("No content sent in message");
        }
    }
}
```