

Invoke Web services with WebSphere MQ and WebSphere Enterprise Service Bus

Build a solution with WebSphere Integration Developer

Skill Level: Advanced

[Philip Norton \(nortonp@uk.ibm.com\)](mailto:nortonp@uk.ibm.com)

Software Engineer, WebSphere ESB development team
IBM

16 Jan 2007

Learn how to invoke a Web service with an IBM® WebSphere® MQ client, using IBM WebSphere Enterprise Service Bus (ESB) and IBM WebSphere Integration Developer. You'll create an MQ Java™ client, write a custom WebSphere MQ data binding and a custom function selector for WebSphere ESB, and configure WebSphere ESB to receive messages from an MQ Queue.

Section 1. Before you start

About this tutorial

The WebSphere Enterprise Service Bus (ESB) enables the connection of applications and services that in many cases have been developed at different times, using different programming languages, interfaces and standards. This tutorial demonstrates how to use an existing WebSphere MQ system to invoke services hosted on WebSphere ESB.

Objectives

You will build an end-to-end solution using the visual tooling supplied by WebSphere

Integration Developer.

Prerequisites

This tutorial assumes that WebSphere Integration Developer is installed with the WebSphere ESB Integrated Test Environment and that the servers default host is configured to port 9080. WebSphere MQ V5.3 FixPack12 or above is required. WebSphere ESB version 6.0.2 or above is required to complete this tutorial.

System requirements

To run the examples in this tutorial, you will require a machine with at least 500MB of memory.

Section 2. Create an MQ client

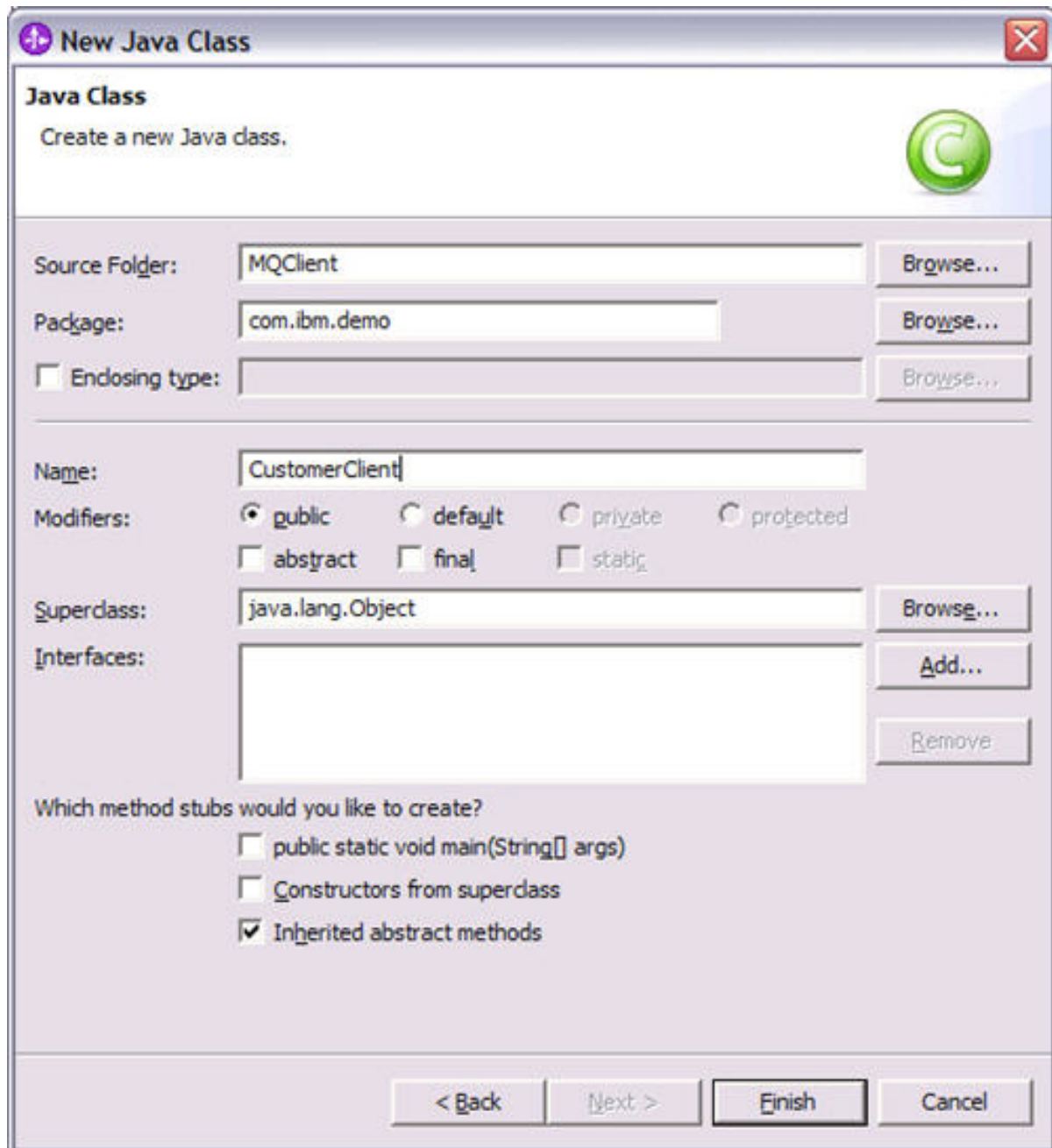
First we must create a WebSphere MQ client that will send a message to a queue. In this tutorial we use Java but this client can usually be written in any of the supported programming languages. We will use a custom data type in the message to show how existing systems can be made to communicate with WebSphere ESB. The Web service we intend to use will accept a customer and return a confirmation that the customer details have been stored.

Start by opening WebSphere Integration Developer and close the welcome pane to enter the Business Integration perspective. If you happen to close this perspective, you can re-open it by selecting **Window > Open Perspective > Other > Business Integration**. For the first task we need to use the Physical Resources view. Select **Window > Show View > Physical Resources**:

1. Right click in the Physical Resources view and select **New > Other...**
2. Choose a project type of **Java > Java Project**. This Opens the **New Java Project** Dialog.
3. Enter `MQClient` as the name of the project.
4. Click **Finish**. If asked to switch to the Java Perspective click **No**.
5. Right click on the MQClient project and click **properties**

6. Select **Java Build Properties** and click the **Libraries** tab
7. If the WESB library is not in your list of libraries, click **Add Library** and select **WebSphere ESB Server v6.0**. Click **Next**
8. Tick **Configure WebSphere ESB Server classpath** click **Finish** then click **OK**
9. Expand the **MQClient** Java project. Right Click and select **New > Other...**
10. Choose **Java > Class** and click **Next**.
11. Enter the package as `com.ibm.demo` and the class name as `CustomerClient` as shown in [Figure 1](#).
12. Click **Finish**.

Figure 1. Create CustomerClient class



9. CustomerClient.java will open. Replace the contents with that shown in [Listing 1](#).

Listing 1. Code for CustomerClient.java

```
package com.ibm.demo;  
  
import com.ibm.mq.MQC;
```

```

import com.ibm.mq.MQEnvironment;
import com.ibm.mq.MQException;
import com.ibm.mq.MQGetMessageOptions;
import com.ibm.mq.MQMessage;
import com.ibm.mq.MQPutMessageOptions;
import com.ibm.mq.MQQueue;
import com.ibm.mq.MQQueueManager;

public class CustomerClient {
    Arguments args; //Command line arguments
    private MQQueueManager qmgr; //Connection to Queue Manager
    /*
     * Constructor
     */
    public CustomerClient(String[] arguments) {
        args = new Arguments(arguments);
    }
    /*
     * Create a new Queue Manager connection if one doesn't exist
     */
    private MQQueueManager getQueueManager() throws MQException {
        MQEnvironment.properties.put(MQC.TRANSPORT_PROPERTY, args.transport);
        MQEnvironment.hostname = args.hostname;
        MQEnvironment.port = args.port;
        MQEnvironment.channel = args.channel;
        if (qmgr == null || !qmgr.isConnected()) qmgr = new MQQueueManager(args.queueManager);
        return qmgr;
    }
    /*
     * Send an MQ message
     */
    public void sendMessage(MQMessage message) throws MQException {
        MQQueue mqQueue = getQueueManager().accessQueue
            (args.requestQueue, MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT);
        MQPutMessageOptions pmo = new MQPutMessageOptions();
        mqQueue.put(message, pmo);
        mqQueue.close();
        disconnect();
    }
    /*
     * Receive an MQ message
     */
    public MQMessage receiveMessage(int wait) throws MQException {
        MQQueue mqQueue = getQueueManager().accessQueue
            (args.responseQueue, MQC.MQOO_INPUT_AS_Q_DEF | MQC.MQOO_OUTPUT);
        MQGetMessageOptions gmo = new MQGetMessageOptions();
        gmo.options = MQC.MQGM_WAIT;
        gmo.waitInterval = wait;
        MQMessage message = new MQMessage();
        mqQueue.get(message, gmo);
        mqQueue.close();
        disconnect();
        return message;
    }
    /*
     * Disconnect from the Queue Manager
     */
    private void disconnect() {
        try {
            if (qmgr != null && qmgr.isConnected()) qmgr.disconnect();
        } catch (MQException e) {
            qmgr = null;
        }
    }
}
/*
 * Class used to gather arguments from the command line in the form of
 * -name value -name value ...
 */
private class Arguments {

```

```

int index = 0;
private String hostname;
private String channel;
private int port;
private String queueManager;
private String requestQueue;
private String responseQueue;
private String transport;

public Arguments(String[] args) {
    process(args);
}

private void process(String[] args) {
    while (index < args.length) {
        String arg = args[index++];
        String value = args[index++];
        if (value.startsWith("-")) {
            value = null;
            index--;
        }
        if (value != null) {
            if (arg.startsWith("-hostname")) this.hostname = value;
            else if (arg.startsWith("-channel")) this.channel = value;
            else if (arg.startsWith("-port")) this.port = Integer.parseInt(value);
            else if (arg.startsWith("-queueManager")) this.queueManager = value;
            else if (arg.startsWith("-requestQueue")) this.requestQueue = value;
            else if (arg.startsWith("-responseQueue")) this.responseQueue = value;
            else if (arg.startsWith("-transport")) {
                if (value.equals("bindings")) this.transport = MQC.TRANSPORT_MQSERIES_BINDINGS;
                else this.transport = MQC.TRANSPORT_MQSERIES_CLIENT;
            }
        }
    }
}

public static void main(String[] args) {
    CustomerClient client = new CustomerClient(args);
    try {
        MQMessage request = new MQMessage();
        //Build a customer
        request.writeInt(1234); //CustomerId MQINT
        request.writeString("Fred "); //First Name MQCHAR12
        request.writeString("Flintstone "); //Last Name MQCHAR12
        request.writeInt(25); //Age MQINT
        request.writeString("Bedrock "); //Address MQCHAR12

        //Send the message
        client.sendMessage(request);

        //Wait for response
        MQMessage response = client.receiveMessage(5000);
        System.out.println("Response from Web Service");
        System.out.println("Customer Id "+response.readInt());
        boolean success = false;
        if (response.readInt() == 1) success = true;
        System.out.println("Success "+success);

    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

There are now compile errors because the MQ Java classes are not on the

classpath.

10. Right click on the MQClient project and select **Properties**.
11. Select **Java Build Path** and select the **Libraries** tab.
12. Click **Add External JARs...**
13. Locate your WebSphere MQ Java/lib directory and select **com.ibm.mq.jar**.
14. Click **Open**.
15. Click **OK**.

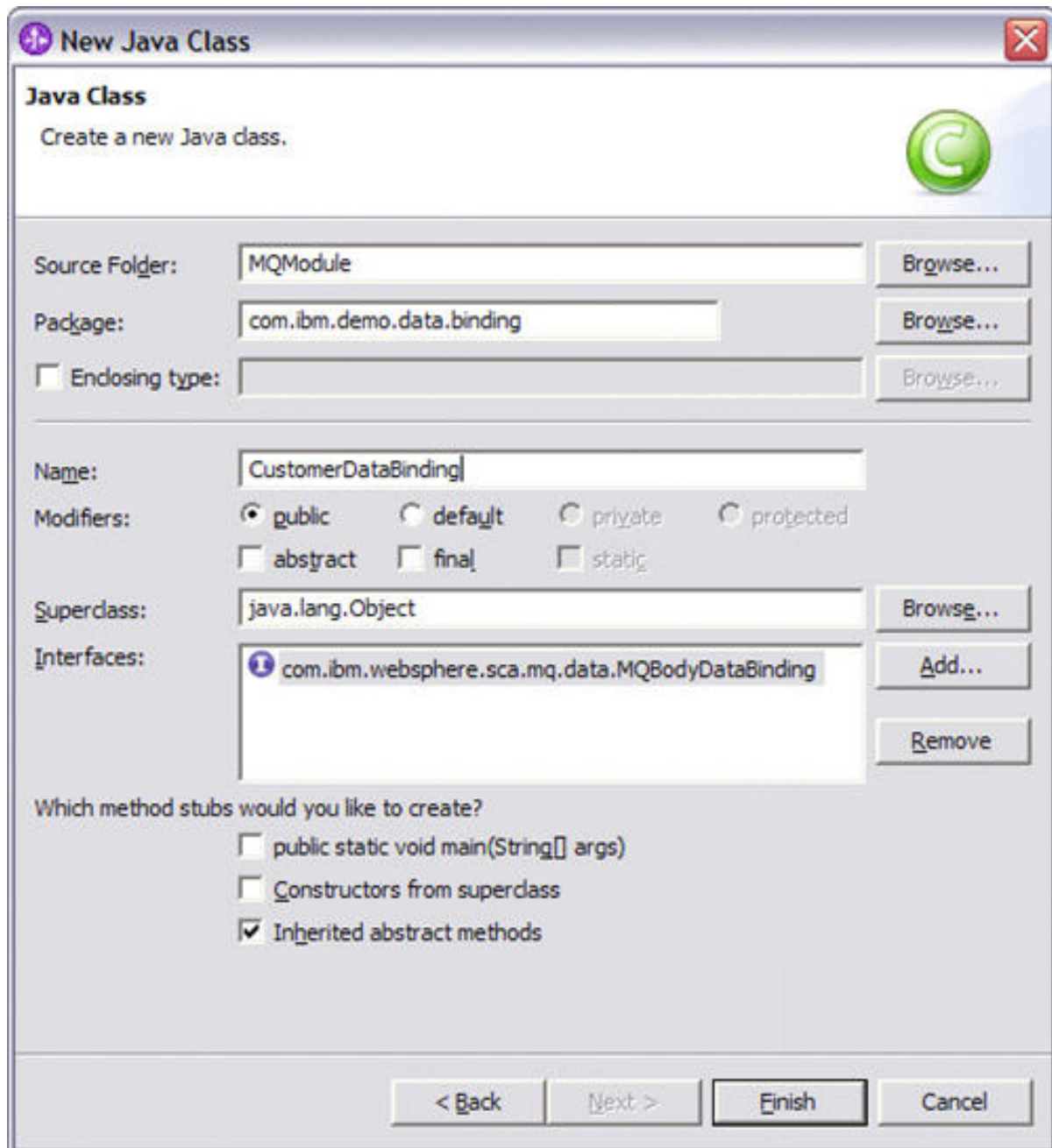
The creation of the MQ client is now complete.

Section 3. Write custom MQ data binding

How does WebSphere ESB know how to understand our customer message? At the moment it doesn't, so instead, we will create a custom data binding. This describes to WebSphere ESB how to build a Business object from the contents of the MQ message. Let's do the following:

1. In the Physical Resources view, right click on the MQModule and select **New > Other...**
2. Choose **Java > Class** and click **Next**.
3. Enter the package as `com.ibm.demo.data.binding` the class name as `CustomerDataBinding` and add the interface **MQBodyDataBinding** as shown in [Figure 2](#).
4. Click **Finish**.

Figure 2. Customer data binding



CustomerDataBinding.java is created with some empty methods. Here is a summary of what the methods are used for.

read is used to read the contents of the message and build a DataObject.

write is used to write the contents of the message from a DataObject.

setBusinessException Unused.

isBusinessException Unused.

setFormat Setter for the message Format.

getFormat Getter for the message Format.

getDataObject Called when a DataObject is required from the data binding.

setDataObject Called when a new DataObject is passed to the data binding.

5. Replace the contents of CustomerDataBinding.java with the implementation in [Listing 2](#) and save.

Listing 2. Code for CustomerDataBinding.java

```
package com.ibm.demo.data.binding;

import java.io.IOException;
import java.util.List;
import com.ibm.mq.data.MQDataInputStream;
import com.ibm.mq.data.MQDataOutputStream;
import com.ibm.websphere.sca.mq.data.MQBodyDataBinding;
import com.ibm.websphere.sca.mq.structures.MQMD;
import com.ibm.websphere.sca.sdo.DataFactory;
import commonj.connector.runtime.DataBindingException;
import commonj.sdo.DataObject;

public class CustomerDataBinding implements MQBodyDataBinding {
    //This is the namespace of the expected request type.
    private static String REQUEST_TYPE_NAMESPACE = "http://MQModule";
    //This is the request type expected
    private static String REQUEST_TYPE_NAME = "Customer";
    //The QName built from the namespace and type
    private static String REQUEST_TYPE_QNAME =
        "{" + REQUEST_TYPE_NAMESPACE + "}" + REQUEST_TYPE_NAME;
    //This is the namespace of the expected response type
    private static String RESPONSE_TYPE_NAMESPACE = "http://MQModule";
    //This is the response type expected
    private static String RESPONSE_TYPE_NAME = "Confirmation";
    //The QName built from the namespace and type
    private static String RESPONSE_TYPE_QNAME =
        "{" + RESPONSE_TYPE_NAMESPACE + "}" + RESPONSE_TYPE_NAME;
    //Format
    public String format;
    //DataObject that is built from the request message
    //It is also used to build the response messages
    public DataObject dataObject;

    public void read(MQMD mqmd, List headers,
        MQDataInputStream input) throws IOException {
        if (dataObject == null) {
            //If the Data Object we received is null create a new request
            dataObject = DataFactory.INSTANCE.create
                (REQUEST_TYPE_NAMESPACE, REQUEST_TYPE_NAME);
        }
        if (dataObject.getType().getURI().equals
            (REQUEST_TYPE_NAMESPACE) &&
            dataObject.getType().getName().equals(REQUEST_TYPE_NAME)) {
            //If the Data Object is a request build a Customer data object
            //by reading the known structure from the input stream
        }
    }
}
```

```

dataObject.setInt("customerId", input.readInt());
dataObject.setString("firstName", input.readMQCHAR12().trim());
dataObject.setString("lastName", input.readMQCHAR12().trim());
dataObject.setInt("age", input.readInt());
dataObject.setString("address", input.readMQCHAR12().trim());
} else if (dataObject.getType().getURI().equals(RESPONSE_TYPE_NAMESPACE)
&& dataObject.getType().getName().equals(RESPONSE_TYPE_NAME)) {
//If the Data Object is a response build a Confirmation data object
//by reading the known structure from the input stream
dataObject = DataFactory.INSTANCE.create
(RESPONSE_TYPE_NAMESPACE, RESPONSE_TYPE_NAME);
dataObject.setInt("customerId", input.readInt());
if (input.readInt() == 1) {
dataObject.setBoolean("success", true);
} else {
dataObject.setBoolean("success", false);
}
}
}

public void write(MQMD mqmd, List headers,
MQDataOutputStream output) throws IOException {
if (dataObject == null) {
//If the Data Object is null create a new response
dataObject = DataFactory.INSTANCE.create
(RESPONSE_TYPE_NAMESPACE, RESPONSE_TYPE_NAME);
}
if (dataObject.getType().getURI().equals
(REQUEST_TYPE_NAMESPACE) &&
dataObject.getType().getName().equals(REQUEST_TYPE_NAME)) {
//If the Data Object is a request write the Customer
//to the message output stream
output.writeInt(dataObject.getInt("customerId"));
output.writeMQCHAR12(dataObject.getString("firstName"));
output.writeMQCHAR12(dataObject.getString("lastName"));
output.writeInt(dataObject.getInt("age"));
output.writeMQCHAR12(dataObject.getString("address"));
} else if (dataObject.getType().getURI().equals(RESPONSE_TYPE_NAMESPACE)
&& dataObject.getType().getName().equals(RESPONSE_TYPE_NAME)) {
//If the Data Object is a response write the confirmation
//to the message output stream
output.writeInt(dataObject.getInt("customerId"));
if (dataObject.getBoolean("success") == true) {
output.writeInt(1);
} else {
output.writeInt(0);
}
}
}

public void setBusinessException(boolean arg0) {
}

public boolean isBusinessException() {
return false;
}

public void setFormat(String arg0) {
format = arg0;
}

public String getFormat() {
return format;
}

public DataObject getDataObject() throws DataBindingException {
return dataObject;
}

```

```
public void setDataObject(DataObject dObj) throws DataBindingException {
    if (dObj == null) dataObject = null;
    else if ((dObj.getType().getURI().equals(REQUEST_TYPE_NAMESPACE) &&
        dObj.getType().getName().equals(REQUEST_TYPE_NAME))
        || (dObj.getType().getURI().equals(RESPONSE_TYPE_NAMESPACE) &&
        dObj.getType().getName().equals(RESPONSE_TYPE_NAME))) {
        // Check the DataObject is the correct type
        dataObject = dObj;
    } else {
        throw new DataBindingException();
    }
}
```

The data binding checks whether it is handling a request or response message. This allows it to be used on both an MQ export and an MQ import. The custom data binding is now complete.

Section 4. Build a module using MQ bindings

The next step is to create an interface that accepts a customer and returns a confirmation

1. Return to the Business Integration view.
2. Select **File > New > Project...**
3. Select **Mediation Module** and click **Next**.
4. Enter the name of the module as `MQModule` and uncheck the **Create mediation flow component check box** as show in [Figure 3](#).

Figure 3. New mediation module

New Mediation Module

Mediation Module

Create a new mediation module. A mediation module is a project that is used for development, version management, organizing resources, and deploying to the ESB runtime environment.

Module Name:

Module Location:

Use default

Directory:

Target Runtime:

Create mediation flow component

Mediation modules can be deployed and run on WebSphere Enterprise Service Bus or WebSphere Process Server. They contain flows, which link together operations for modifying and routing messages between service consumers and service providers.

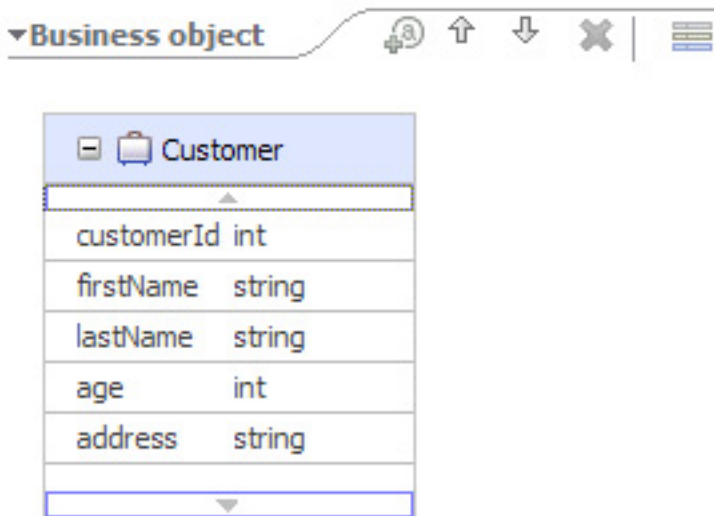
< Back Next > Finish Cancel

Now let's create the business objects that describe the customer and the confirmation used in the MQ client.

5. Click **Finish**. A new module is created.
6. Right click on **Data Types**, in MQModule and select **New > Business Object**.
7. Enter the name of the Business Object as `Customer`.

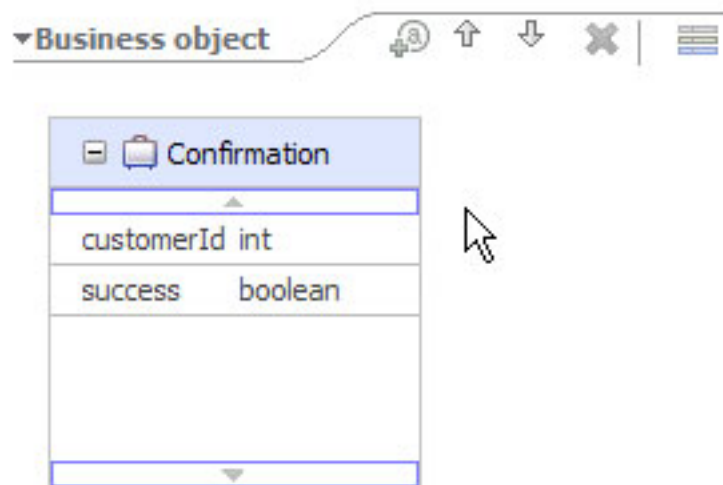
8. Click **Finish**. The Business Object Editor will open.
9. Add the required attributes to the Customer, as shown in [Figure 4](#).

Figure 4. Customer business object



10. Create another business object called `Confirmation` with the required attributes, as shown in [Figure 5](#).

Figure 5. Confirmation business object

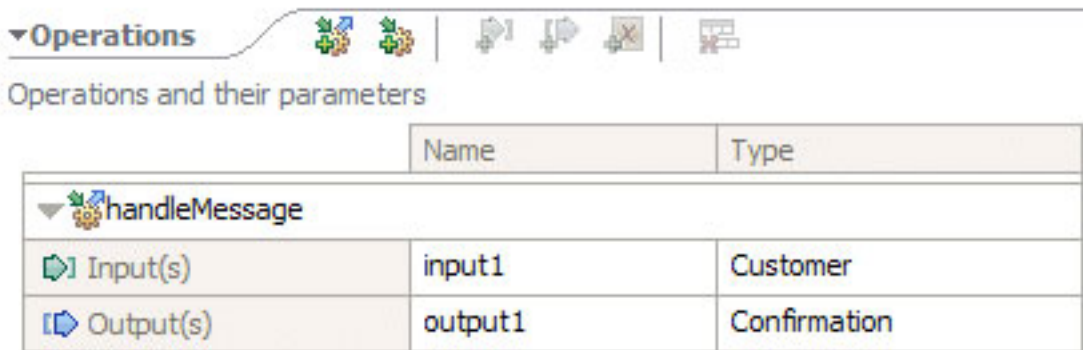


Now we will create the interface that accepts a customer and returns a confirmation:

11. In the Business Integration view, right click on **Interfaces** in MQModule.

12. Select **New > Interface**.
13. Name the interface `CustomerInterface`.
14. Click **Finish**. The interface editor will open.
15. Add a two way operation called `handleMessage`.
16. Set the input to be of type `Customer` and output to be of type `Confirmation`, as shown in [Figure 6](#).

Figure 6. Customer interface



Next we will create the MQ Module. A Java component is used to simulate a Web service in this scenario.

17. Open the assembly editor for MQModule.
18. Drag the **CustomerInterface** onto the editor and select **Export with no binding** and click **Ok**.
19. Now add a Java component to the editor and wire it to the export, as show in [Figure 7](#).

Figure 7. MQModule



20. Right click on the Java component and select **Generate Implementation**.

21. Select the default package and click **OK**.
22. Replace the implementation with that in [Listing 3](#).

Listing 3. Code for Java component

```
package sca.component.java.impl;

import java.util.HashMap;
import java.util.Map;
import commonj.sdo.DataObject;
import com.ibm.websphere.bo.BOFactory;
import com.ibm.websphere.sca.ServiceManager;

public class Component1Impl {

    static Map customers = new HashMap();

    public Component1Impl() {
        super();
    }

    private Object getMyService() {
        return (Object) ServiceManager.INSTANCE.locateService("self");
    }

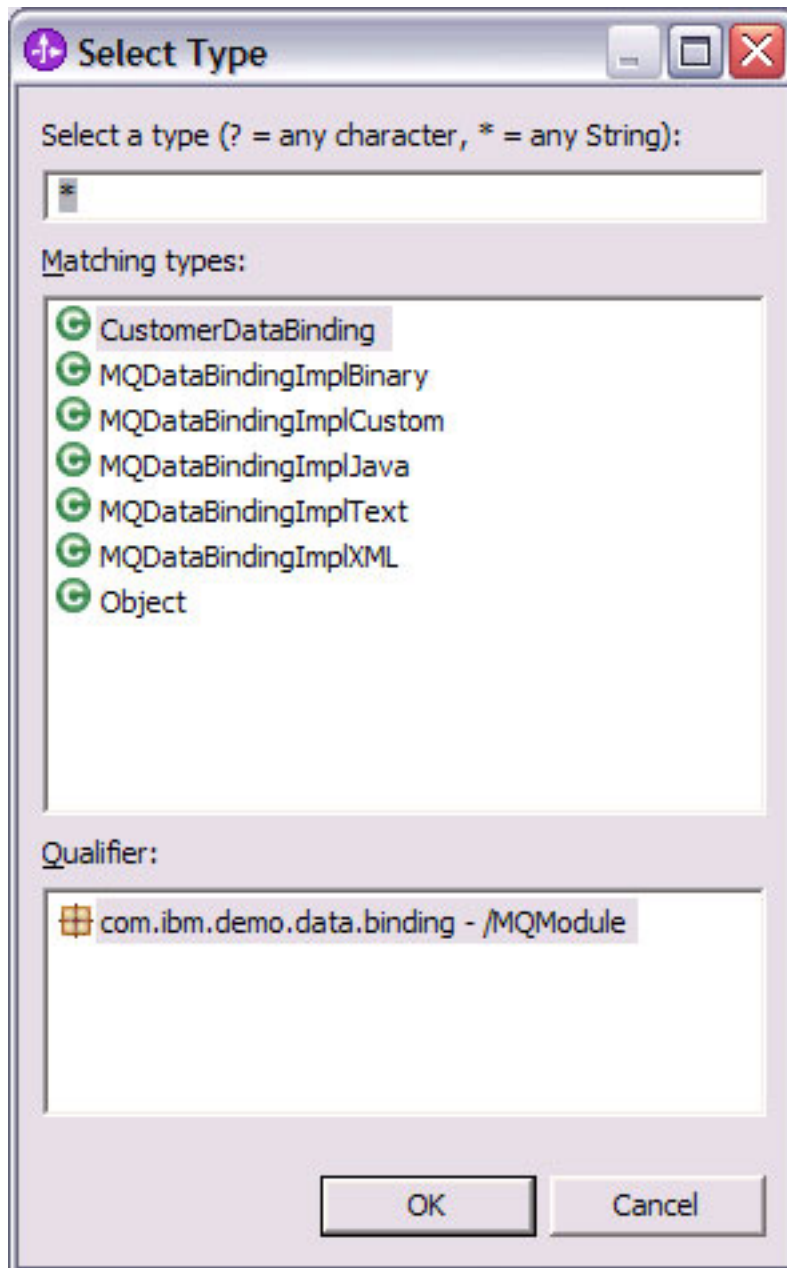
    public DataObject handleMessage(DataObject input) {
        boolean success = false;
        //Get the customer id
        int id = input.getInt("customerId");
        //Create the customer from the request
        Customer customer = new Customer();
        customer.fName = input.getString("firstName");
        customer.lName = input.getString("lastName");
        customer.age = input.getInt("age");
        customer.address = input.getString("address");
        //Check whether the customer is already stored
        if(customers.get(new Integer(id)) == null)
        {
            //Store the customer if they don't currently exist
            customers.put(new Integer(id), customer);
            success = true;
        }
        //Build the confirmation response
        ServiceManager serviceManager = new ServiceManager();
        BOFactory bofactory =
            (BOFactory) serviceManager.locateService("com/ibm/websphere/bo/BOFactory");
        DataObject response =
            bofactory.createElement("http://MQModule/CustomerInterface", "handleMessageResponse");
        DataObject confirmation = response.createDataObject("output1");
        confirmation.setInt("customerId", id);
        confirmation.setBoolean("success", success);
        return confirmation;
    }

    private class Customer {
        String fName;
        String lName;
        int age;
        String address;
    }
}
```

Next we need to generate the MQ binding on the export. As this tutorial is focused on configuring WebSphere ESB, we will assume that an MQ Queue Manager **QM** is available on the machine and two queues **REQUESTQ** and **RESPONSEQ** have been created. To use a different configuration simply fill in your MQ configuration when creating the data binding.

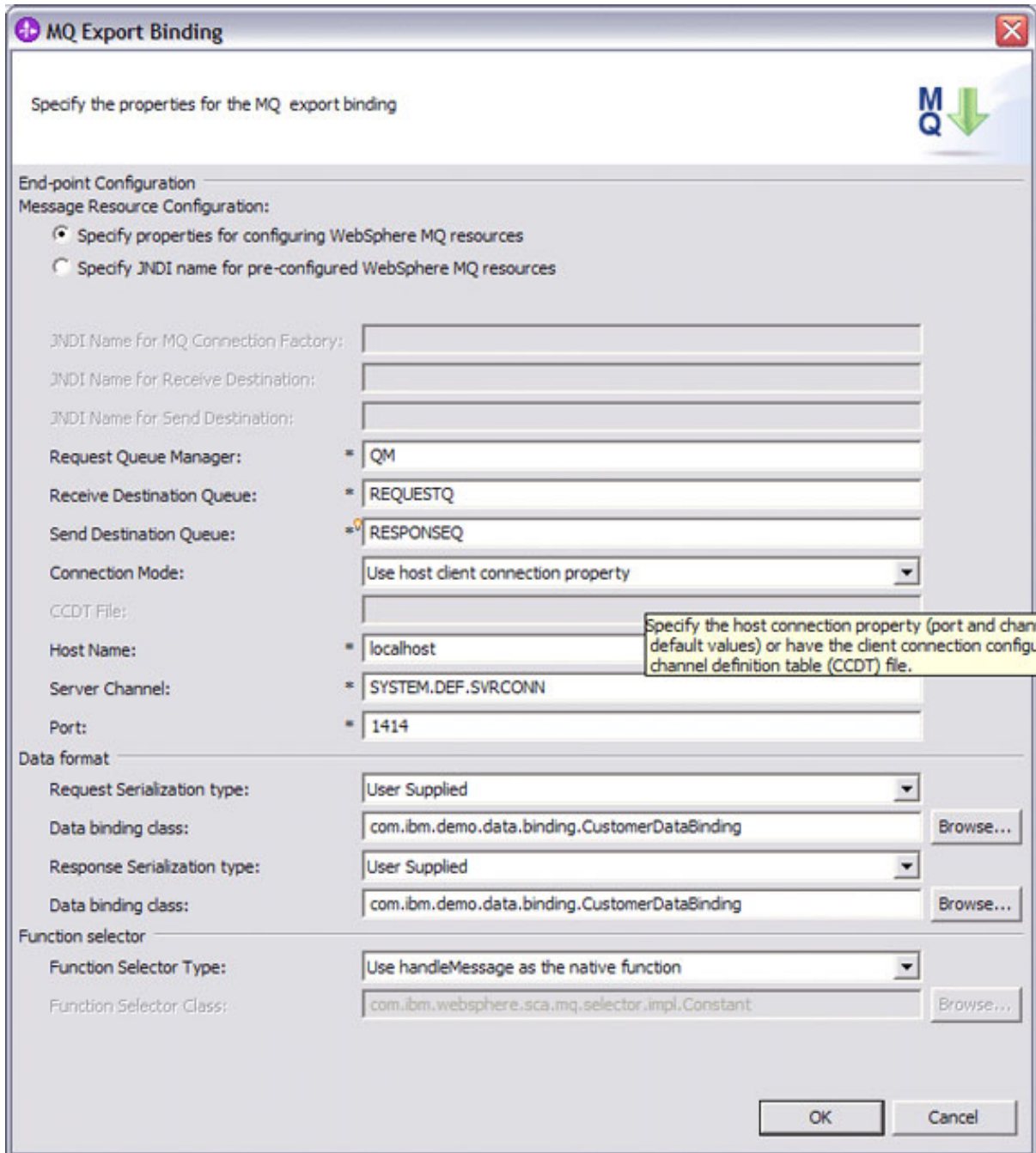
23. Right click on **Export1** and select **Generate Binding... -> Messaging Binding... -> MQ Binding**.
24. The MQ Export Binding property panel will be opened.
25. Fill in the End-point Configuration text boxes with your MQ properties as shown in [Figure 9](#).
26. In the *Data Format* section under *Request Serialization type* select **User supplied** and click **Browse**.
27. In the Select Type dialog type * and select our **CustomerDataBinding** as shown in Figure 8.

Figure 8. Select data binding



28. Select the same data binding for *Response Serialization Type* the panel should now look like [Figure 9](#).

Figure 9. MQ export binding



29. Click **OK** and save. The module is now complete.

Section 5. Test the MQ Binding

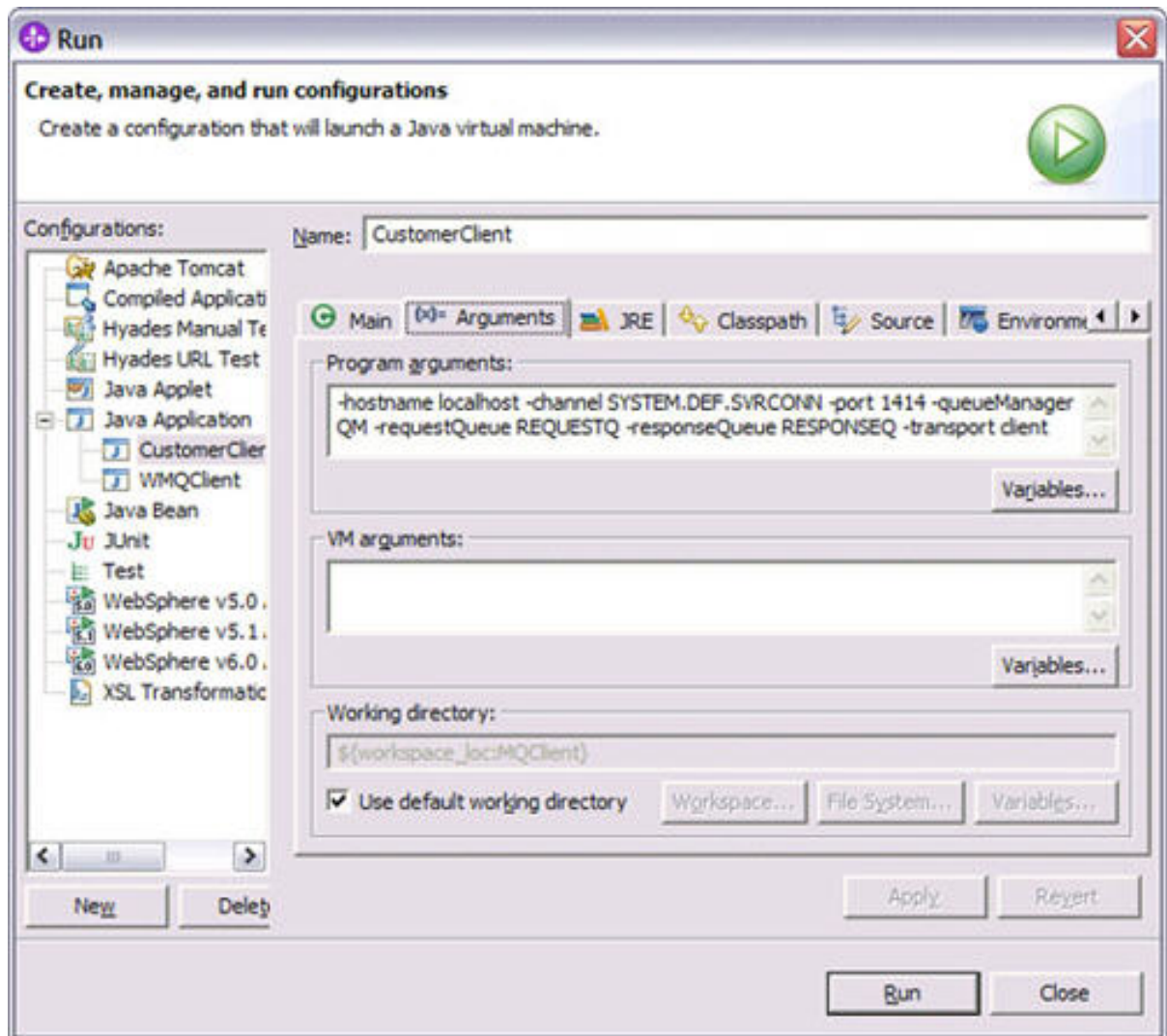
Before we run the client, we need to deploy the module.

1. Open the server view, right click on your server and select **Start**.
2. Once the server has started, right click on it again and select **Add and Remove projects...**
3. Select the **MQModule**, click **Add** and then **OK**.
4. Wait until the project has been deployed and successfully started.

Now let's try sending an MQ Message to a queue and see if we get a reply.

5. Right click on **CustomerClient.java** and select **Run > Run...**
6. Click on the Arguments tab.
7. In the Program Arguments text box enter `-hostname localhost
-channel SYSTEM.DEF.SVRCONN -port 1414 -queueManager
QM -requestQueue REQUESTQ -responseQueue RESPONSEQ
-transport client` as shown in [Figure 10](#).
8. Click **Run**.

Figure 10. Run MQ client



In the console you should see the output from the client confirming the customer has been successfully added. Try running the client again to add the same customer, you should see a response but this time the success field will be false. You can edit the MQClient to add customers with different ID's.

Section 6. Conclusion

In this tutorial we have used WebSphere Integration Developer and WebSphere ESB to develop and test:

1. An MQ client that can send MQ messages containing a custom data type

2. A Java Service implemented using a Java component
3. A custom data binding allowing the format of the message to be translated into the format required by the service

This tutorial has shown how to invoke a Web service using WebSphere MQ. However, the same custom data binding could be used to expose an MQ application as a Web service. By connecting an Export, with a Web service Binding, to an MQ Import, using our custom data binding, MQ applications could be invoked with a Web service call. Hopefully this article has given you a basic understanding of how to use these two products, to integrate WebSphere MQ and WebSphere MQ applications with Web services.

Downloads

Description	Name	Size	Download method
Project Interchange for mediation module	MQModule.zip	7KB	HTTP
MQ Client	MQClient.zip	3KB	HTTP

[Information about download methods](#)

Resources

Learn

- Read the tutorial "[Invoking a Web service with a JMS client](#)" (developerWorks, April 2006).
- [WebSphere ESB product information](#) offers product descriptions, product news, training information, support information, and more.
- [WebSphere ESB Information Center](#) is an Eclipse-based Web portal to all WebSphere ESB documentation, with conceptual, task, and reference information on installing, configuring, and using WebSphere ESB.
- [WebSphere ESB support](#) has a searchable database of support problems and their solutions, plus downloads, fixes, problem tracking, and more.
- [WebSphere Integration Developer product page](#) offers a single Eclipse-based Web portal to all WebSphere Integration Developer documentation, with conceptual, task, and reference information on installing, configuring, and using your WebSphere Integration Developer environment.
- [WebSphere Integration Developer information center](#) provides product descriptions, product news, training information, support information, and more.
- [WebSphere Integration Developer support](#) has a searchable database of support problems and their solutions, plus downloads, fixes, problem tracking, and more.
- [Getting started with WebSphere Enterprise Service Bus and WebSphere Integration Developer](#) introduces developers to the IBM WebSphere Enterprise Service Bus server and its accompanying tooling, WebSphere Integration Developer.
- [Developing custom mediations for WebSphere Enterprise Service Bus](#) introduces the use of custom mediations using the WebSphere Integration Developer V6 environment for WebSphere Enterprise Service Bus V6.
- [Hello World: WebSphere Enterprise Service Bus: Develop message flow for protocol transformation](#), an intro tutorial, covers the basic concepts and includes a hands-on exercise that shows you how to build a mediation module for protocol transformation.
- [WebSphere MQ V6.0 certification preparation tutorials](#), although designed to help you prepare for the IBM certification test 996, offer a comprehensive look at WebSphere MQ. Starting with an overview and finishing with advanced WebSphere MQ API (MQI) topics, this series gets you thoroughly acquainted with WebSphere MQ.

Get products and technologies

- Get the [IBM WebSphere MQ trial download](#).

About the author

Philip Norton



Philip Norton is a Software Engineer at the IBM Hursley Lab, England. He works on the WebSphere ESB Development team. His expertise includes Java and JMS for WebSphere MQ. He is a certified Java Programmer and he has a degree in Computer Science from Canterbury University.