



---

**Q-Pulse Web Service APIs  
Developing a BizTalk Solution**

# Contents

---

Introduction.....	3
Assumptions .....	3
Overview of the example.....	3
Initial Solution setup .....	4
Define Schemas and Maps and Build Schemas Project.....	5
Creating the initial Orchestration.....	9
Adding the Q-Pulse Web Service References and associated ports .....	11
Implementing Authentication with the QPulse Web Service .....	12
Retrieving the list of Occurrence IDs.....	14
Processing the Received List .....	15
Fetching the New Occurrences.....	16
Sending each fetched Occurrence out to an external system .....	17
Final Steps .....	18
Deploying the Application.....	18
Running the Application .....	19
Undeploying the Application .....	19
Limitations .....	19
Future Recommendations.....	20
Useful Links .....	20

## Introduction

The purpose of this tutorial is to demonstrate how BizTalk can be used to retrieve data from QPulse Web Services and send it to another system. This will require implementing a BizTalk Orchestration which polls the QPulse Web Service periodically for data, perform any data transformations required and send the data to some external system or location.

The example was developed on a system with the following software versions:

- Microsoft Server 2003 SQL Server 2003
- Visual Studio 2005
- BizTalk Server 2006 R2

## Assumptions

This tutorial assumes the reader is experienced in the use of Visual Studio 2005 as well as the use and administration of BizTalk Server 2006.

## Overview of the example

In this example scenario, the required system should poll the QPulse Web Services to find Occurrences with Status 'New' (on an on-going basis), fetch the Occurrences that haven't been seen by the system before and send them to some external system or location.

A descriptive overview of the BizTalk orchestration process would be:

- Receive a message to start the system.
- Send a confirmation message out to the filesystem to indicate the system has been successfully started.
- Authenticate with the Core Web Service to obtain an authentication token.
- Send a request to the Occurrence Web Service to obtain a list of the Occurrence IDs which have Status 'New'.
- Receive the response.
- Compare the list of IDs received to a list held in memory and work out which Occurrences haven't been processed before.
- For each Occurrence ID not processed before, fetch the full Occurrence from the Web Service and write out the Occurrence to a new file.
- Wait for a pre-determined amount of time and then repeat steps 3 -6.

## Initial Solution setup

1. Create a new Solution:
2. Go to File -> New -> Project -> Visual C# -> BizTalk Projects -> Empty BizTalk Server Project and name it QPulseBizTalkSolution
3. Create a new project for the schemas and maps:
4. In Solution Explorer, right-click on the Solution and go to Add -> New Project -> Visual C# -> BizTalk Projects -> Empty BizTalk Server Project and name it QPulseBizTalkSchemas
5. Create a new project for the orchestration:
6. In Solution Explorer, right-click on the Solution and go to Add -> New Project -> Visual C# -> BizTalk Projects -> Empty BizTalk Server Project and name it QPulseBizTalkOrchestrations
7. Right-click on the project QPulseBizTalkSolution and 'Remove'.

## Define Schemas and Maps and Build Schemas Project

1. We need to define the format of the incoming message to start the system. Right-click on QPulseBizTalkSchemas and Add-> New Item -> Schema Files -> Schema and call it Start.xsd and copy and paste the following xml:

```
<?xml version="1.0" encoding="utf-16"?>

<xs:schema xmlns:b="http://schemas.microsoft.com/BizTalk/2003"
xmlns="http://QPulseBizTalkSchemas.Start"
targetNamespace="http://QPulseBizTalkSchemas.Start"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="Orchestration">

    <xs:annotation>

      <xs:appinfo>

        <b:properties>

          <b:property distinguished="true" xpath="/*[local-
name()='Orchestration' and namespace-
uri()='http://QPulseBizTalkSchemas.Start']/*[local-name()='Start' and
namespace-uri()='']" />

        </b:properties>

      </xs:appinfo>

    </xs:annotation>

    <xs:complexType>

      <xs:sequence>

        <xs:element name="Start" type="xs:boolean" />

      </xs:sequence>

    </xs:complexType>

  </xs:element>

</xs:schema>
```

2. We need to define the format of the outgoing message to confirm that the system has started. Similarly create another schema file called StartConfirmation.xsd and paste the following xml:

```
<?xml version="1.0" encoding="utf-16"?>

<xs:schema xmlns:b="http://schemas.microsoft.com/BizTalk/2003"
xmlns="http://QPulseBizTalkSchemas.StartConfirmation"
targetNamespace="http://QPulseBizTalkSchemas.StartConfirmation"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="Orchestration">

    <xs:annotation>

      <xs:appinfo>

        <b:properties>

          <b:property distinguished="true" xpath="/*[local-
name()='Orchestration' and namespace-
uri()='http://QPulseBizTalkSchemas.StartConfirmation']/*[local-
name()='Start' and namespace-uri()='']" />

        </b:properties>

      </xs:appinfo>

    </xs:annotation>

    <xs:complexType>

      <xs:sequence>

        <xs:element name="Start" type="xs:boolean" />

      </xs:sequence>

    </xs:complexType>

  </xs:element>

</xs:schema>
```

3. We need to define the format of the Occurrences written out to the filesystem. Create another schema file called OccurrenceToExternalSystem.xsd and paste the following xml:

```
<?xml version="1.0" encoding="utf-16"?>

<xs:schema xmlns:b="http://schemas.microsoft.com/BizTalk/2003"
xmlns="http://QPulseBizTalkSchemas.OccurrenceToExternalSystem"
targetNamespace="http://QPulseBizTalkSchemas.OccurrenceToExternalSystem"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="Occurrence">

    <xs:annotation>

      <xs:appinfo>

        <b:properties>

          <b:property distinguished="true" xpath="/*[local-
name()='Occurrence' and namespace-
uri()='http://QPulseBizTalkSchemas.OccurrenceToExternalSystem']/*[local-
name()='ID' and namespace-uri()='']" />

          <b:property distinguished="true" xpath="/*[local-
name()='Occurrence' and namespace-
uri()='http://QPulseBizTalkSchemas.OccurrenceToExternalSystem']/*[local-
name()='Title' and namespace-uri()='']" />

        </b:properties>

      </xs:appinfo>

    </xs:annotation>

    <xs:complexType>

      <xs:sequence>

        <xs:element name="ID" type="xs:string" />

        <xs:element name="Title" type="xs:string" />

      </xs:sequence>

    </xs:complexType>

  </xs:element>

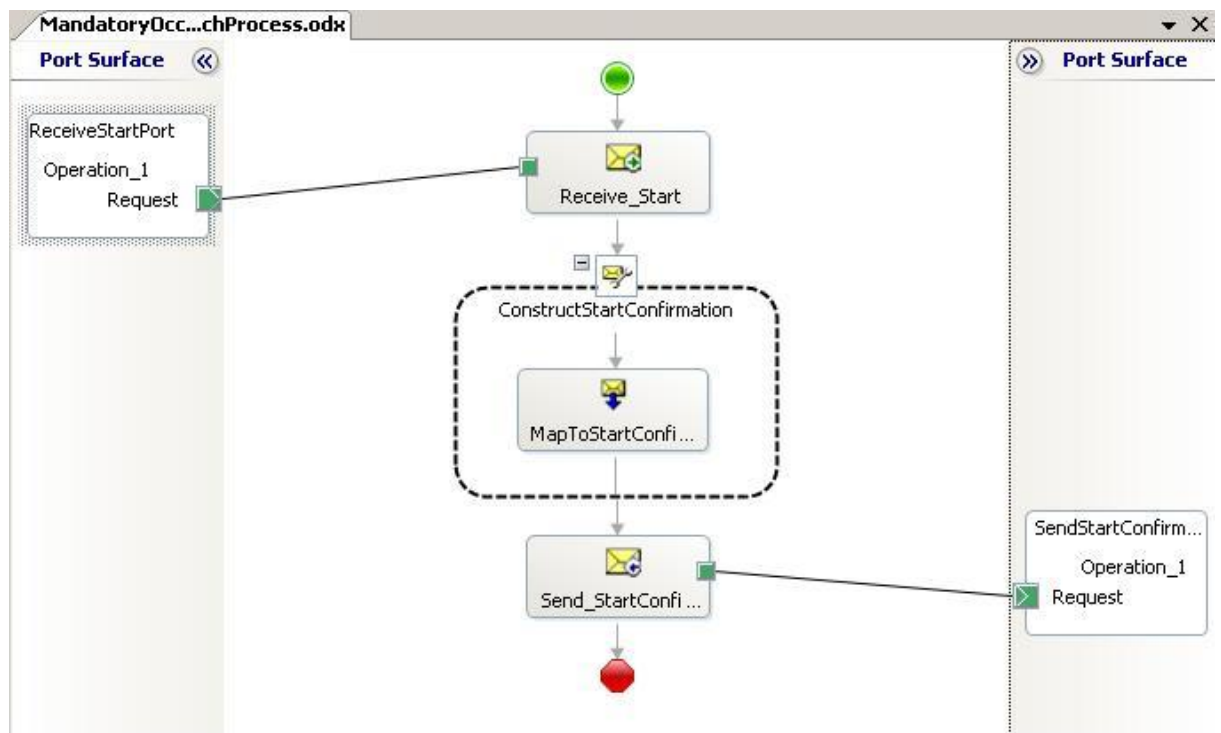
</xs:schema>
```

4. We need to create a Map to map the values of the Start message to the values of the StartConfirmation message. Right-click on QPulseBizTalkSchemas and Add -> New Item -> Map Files -> Map and name it MapToStartConfirmation.btm
5. In the Map editor, set the source schema to Start.xsd and the destination schema to StartConfirmation.xsd and drag and line from the Start element of the source to the Start element of the destination.
6. We need to define a pipeline to write out XML to a flat file for use when writing out the StartConfirmation message. Right-click on QPulseBizTalkSchemas and Add -> New Item -> Pipeline Files -> Send Pipeline and name it SendStartConfirmationPipeline.btp
7. From the toolbox, drag a 'Flat file assembler' into the 'Drop Here!' under the Assemble section and set the Document Schema property to QPulseBizTalkSchemas.Start.
8. We need to create a key for signing assemblies so that they can be installed in the Global Assembly Cache. Create a key by opening a Visual Studio 2005 Command prompt and change the current directory to that of the Solution folder. Generate a key pair with the command 'sn -k QPulseBizTalkKey.snk'.
9. Right-click on QPulseBizTalkSchemas in Solution Explorer and choose properties. Under Common Properties -> Assembly, set the Assembly Key File to that of the key generated in the previous step. In the Configuration Properties, set Deployment -> Application Name to QPulseBizTalkApplication
10. Now right-click on QPulseBizTalkSchemas and choose Build. The build should be successful.

## Creating the initial Orchestration

1. We need to add a reference to the schemas project in the orchestration project. Right-click on QPulseBizTalkOrchestrations in Solution Explorer and go to Add Reference, click on the Projects tab and add QPulseBizTalkSchemas. After doing so, click on the new reference under the Reference section and set the Copy Local property to False.
2. An Orchestration is required to hold all the components required and define the process. In the QPulseBizTalkOrchestrations project right click and go to Add -> New Item -> Orchestration Files -> BizTalk Orchestration, rename it to NewOccurrenceFetchProcess.odx.
3. Drag a 'Receive' component from the Toolbox on to the diagram and rename it 'Receive\_Start' in the Properties pane and set Activate to True.
4. Similarly, drag a 'Send' component from the Toolbox on to the diagram (below the Receive component) and rename it to 'Send\_StartConfirmation'.
5. In the 'Orchestration View' tab, right-click messages and choose 'New Message', rename it to 'StartInstance' and set the Message Type to Schemas -> Select from referenced assembly -> QPulseBizTalkSchemas -> QPulseBizTalkSchemas -> Start
6. Similarly, add another message called 'StartConfirmationInstance' with type 'StartConfirmation'.
7. Click on the 'Receive\_Start' component and set the Message property to 'StartInstance' in the Properties pane.
8. Similarly, click on the 'Send\_StartConfirmation' component and set the Message property also to 'StartInstance'.
9. Drag a 'Transform' component onto the diagram between the receive and send components. Click on the outer Construct box and rename it to 'ConstructStartConfirmation'. Set the Messages Constructed property to StartConfirmationInstance.
10. Click on the inner Transform box and rename it to 'MapToStartConfirmation'. In the Properties pane, click on Input Messages to show the Transform Configuration dialog. Choose 'Existing Map' and in the pull-down menu, choose 'Select from referenced assembly' -> QPulseBizTalkSchemas -> QPulseBizTalkSchemas -> MapToStartConfirmation. Still in the Transform Configuration dialog, in the Source Transform section, set the Variable Name to 'StartInstance' (it should pop up) and in the Destination section, set the Variable Name to 'StartConfirmationInstance'.
11. Drag a Port component from the Toolbox on to the left-hand side of the diagram which shows the Port Configuration Wizard, set the name to ReceiveStartPort, click Next, leave the settings at 'Create a new Port Type', set the name to ReceiveStartType, leave Communication Pattern at One Way, and Access Restriction at Internal, click Next, leave the two pull down boxes at 'I'll always be receiving.....' and 'Specify Now'. Set the URI to 'C:\Projects\FileDrop\Start\\*.xml' and the Transport to 'File', leave the Receive Pipeline setting at 'Microsoft.BizTalk.DefaultPipeline.XMLReceive'.
12. Connect the port to the Receive component by clicking and dragging the green arrow on the port to the green receptacle on the Receive component.

13. Drag a Port component from the Toolbox on to the right-hand side of the diagram. The setting should be the same as the previous port except for the name which should be SendStartConfirmationPort and the type name which should be SendStartConfirmationType. The pull down boxes should be changed to 'I'll always be sending....' And 'Specify now'. The URI should be set to: 'C:\Projects\FileDrop\StartConfirmation\%MessageID%.xml', set the Transport to 'FILE' and the Pipeline should be set to 'Microsoft.BizTalk.DefaultPipeline.PassThruTransmit'.
14. Connect the port to the Send component.



At this point, the project should be in a working state. It should now be able to receive a message in XML (which starts the orchestration process) and sends out a confirmation message, also in XML.

Similar to the QPulseBizTalkSchemas project, the key and Application name have to be set.

15. Right-click on QPulseBizTalkOrchestrations in Solution Explorer and choose properties. Under Common Properties -> Assembly, set the Assembly Key File to that of the key generated previously. In the Configuration Properties, set Deployment -> Application Name to 'QPulseBizTalkApplication'
16. Now right-click on QPulseBizTalkOrchestrations and choose Build. The build should be successful.

In order to Deploy and test the Application at this point, please refer to the Deployment and Run sections.

## Adding the Q-Pulse Web Service References and associated ports

1. In Solution Explorer, right-click on the QPulseBizTalkOrchestrations project and click on Add Web Reference. Enter the URL to the service (of the form `http://machineName/QPulseWebServices/services/core.svc`) in the text box and click Go. If successful, this should display the Web Service's web page. In the 'Web reference name' text box, enter 'CoreService' and click the Add Reference button.
2. Similarly, add a Web Reference to <http://machineName/QPulseWebServices/services/occurrence.svc> and name it 'OccurrenceService'.
3. Drag a Port component from the Toolbox on to the left-hand side of the diagram which shows the Port Configuration Wizard, set the name to CoreServicePort, click Next. Click on 'Use an existing Port Type', choose Web Port Types -> QPulseBizTalkOrchestrations.Core\_.Core, leave the Port binding at 'Specify now' and proceed to completion.
4. Similarly, drag another port on to the diagram, name it OccurrenceServicePort and set the type to QPulseBizTalkOrchestrations.Occurrence\_.Occurrence.

## Implementing Authentication with the QPulse Web Service

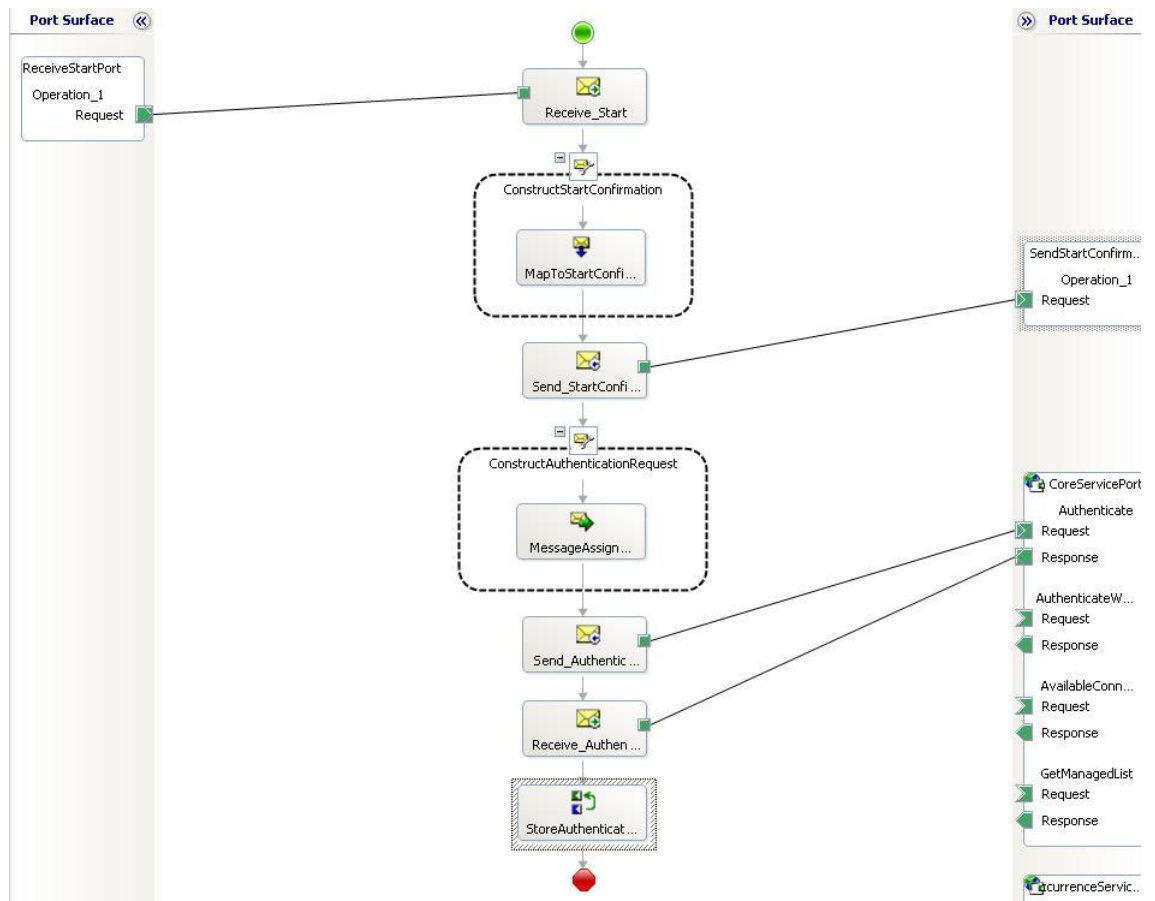
In order to get the list of Occurrence IDs, we need to pass in a valid authentication token so we must first authenticate with QPulse via the Core Web Service.

1. Create a new Message in the Orchestration View window called `AuthenticationRequestInstance` and set the message type to Web Message Types -> `QPulseBizTalkOrchestrations.CoreService.Core_.Authenticate_request`.
2. Similarly, create another message called `AuthenticationResponseInstance` with type Web Message Types -> `QPulseBizTalkOrchestrations.CoreService.Core_.Authenticate_response`.
3. Drag a Send component from the Toolbox to the bottom most point on the diagram and name it `Send_AuthenticationRequest` and set the Message property to `AuthenticationRequestInstance`. Connect it to the Request part of the Authenticate operation.
4. Drag a MessageAssignment component on to the diagram, just above the `Send_Login` component. Click on the outer box and rename it to `ConstructAuthenticationRequest` and set the Messages Constructed property to `AuthenticationRequestInstance`. In the inner box, rename it to `MessageAssignment_AuthenticationRequest`. In the Expression property, enter the following into the editor, replacing the values as required:  

```
AuthenticationRequestInstance.database = "Bacchus";
AuthenticationRequestInstance.password = "password";
AuthenticationRequestInstance.userName = "admin";
```
5. Drag a receive port from the Toolbox on to the diagram, underneath `Send_AuthenticationRequest` and rename it to `Receive_AuthenticationResponse` and set the Message property to `AuthenticationResponseInstance`. Connect it to the Response part of the Authenticate operation.
6. In the Orchestration View window, create a new variable called `AuthenticationToken` and set the type to `System.String`.
7. Drag an Expression component from the Toolbox on to the diagram beneath the `Receive_AuthenticationResponse` component, name it `StoreAuthenticationToken` and in the Expression Editor, enter the following code to assign the retrieved value:  

```
AuthenticationToken =
AuthenticationResponseInstance.AuthenticateResult;
```

The diagram should now look similar to this:



## Retrieving the list of Occurrence IDs

To get the list of Occurrence IDs, we need to call the `GetOccurrenceIDs` method of the Occurrence Web Service which takes in an authentication token and an `OccurrenceQuery` object. This would be possible to achieve with the Send and Receive BizTalk Orchestration components (in a similar fashion to authentication), however, we will choose to do it in custom code in order to construct the `OccurrenceQuery` object and manipulate the received data more easily.

1. Right-click on the Solution in Solution Explorer and choose to Add a new Project of type Visual C# -> Windows -> Class Library and name it `QPulseBizTalkCode`.
2. Sign the Assembly with the key generated previously (via the Signing tab in Project properties).
3. Add a Web Reference to the project with the URI `'http://machineName/QPulseWebServices/services/occurrence.svc'` and name `'OccurrenceService'`.
4. Open a command prompt (Run -> `cmd.exe`) and run the `Wsdll` utility against the Occurrence Web Service to generate a client-side proxy in order to communicate with the Web Service. The command should be similar to `"C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin\wsdl.exe" /n:QPulseBizTalkCode /out:OccurrenceProxy.cs http://machineName/QpulseWebServices/services/Occurrence.svc?wsdl`
5. Back in Visual Studio, right-click on the `QPulseBizTalkCode` project and choose Add -> Existing Item, navigate to the generated proxy file and add it to the project.
6. Create a new Class called `Util.cs` to the project.
7. Insert this method in `Util.cs` which communicates with Occurrence Web Service and returns an `ArrayList` of the IDs returned:

```
public static ArrayList GetOccurrenceIDs(string token)
{
    // initialise the proxy
    Occurrence client = new Occurrence();

    //construct the query object
    OccurrenceQuery query = new OccurrenceQuery();
    query.Status = OccurrenceStatusQueryCriteria.New;

    // communicate with Web Service
    int [] arrayOfIDs = client.GetOccurrenceIDs(token, query);

    //convert the array to an ArrayList
    ArrayList list = new ArrayList();
    for (int i = 0; i < list.Count; i++)
    {
        list.Add(arrayOfIDs[i]);
    }

    return list;
}
```

8. Open up NewOccurrenceFetchProcess.odx and in the Orchestration View window, add a new variable called FetchedIDs with type System.Collections.ArrayList.
9. Similarly add Variables called IDsWithProcess and IDsProcessed, also with type System.Collections.ArrayList.
10. Drag an Expression component from the Toolbox on to the diagram beneath StoreAuthentication token. Rename it to FetchOccurrenceIDs and in the Expression Editor, enter the following:

```
FetchedIDs =
QPulseBizTalkCode.Util.GetOccurrenceIDs (AuthenticationToken)
;
```

## Processing the Received List

1. In Util.cs add another method in order to compare the list of processed Occurrence IDs with the new list retrieved from the WebService:

```
public static ArrayList GetIDsNotProcessed(
    ArrayList occurrenceIDsProcessed,
    ArrayList fetchedOccurrenceIDs)
{
    ArrayList iDsToProcess = new ArrayList();

    for (int i = 0; i < fetchedOccurrenceIDs.Count; i++)
    {
        if (
            !occurrenceIDsProcessed
            .Contains(fetchedOccurrenceIDs[i]))
        {
            iDsToProcess.Add(fetchedOccurrenceIDs[i]);
        }
    }
    return iDsToProcess;
}
```

2. Drag another Expression component on to the diagram beneath FetchOccurrenceIDs. Rename it to ProcessFetchedIDs and in the Expression Editor, enter:
 

```
iDsToProcess =
QPulseBizTalkCode.Util.GetIDsNotProcessed(iDsProcessed,
FetchedIDs);
```

## Fetching the New Occurrences

1. Drag a Loop component from the Toolbox on to the diagram beneath ProcessFetchedIDs and rename it to Loop\_FetchOccurrencesNotSeen. In the Expression Editor, enter:  
`IDsToProcess.Count > 0`
2. In the Orchestration View Window, create two new messages –  
GetOccurrenceRequestInstance with type Web Message Type ->  
QPulseBizTalkOrchestrations.OccurrenceService.Occurrence\_.GetOccurrence\_request  
and GetOccurrenceResponseInstance with type Web Message Type ->  
QPulseBizTalkOrchestrations.OccurrenceService.Occurrence\_.GetOccurrence\_request.
3. Drag a Message Assignment component into the newly created loop, rename the outer box to ConstructGetOccurrenceRequest and set the Message Constructed property to GetOccurrenceRequestInstance. Click on the inner box and rename it to MessageAssignment\_GetOccurrenceRequest
4. Drag a Send component on to the diagram beneath ConstructGetOccurrenceRequest, rename it to Send\_GetOccurrenceRequest and set the Message property to GetOccurrenceRequestInstance. Connect it to the Request section of the GetOccurrence operation on the OccurrenceServicePort.
5. Drag a Receive component on to the diagram beneath Send\_GetOccurrenceRequest, rename it to Receive\_GetOccurrenceResponse and set the Message property to GetOccurrenceResponseInstance. Set the Activate property to True. Connect it to the Response section of the GetOccurrence operation on the OccurrenceServicePort.

## Sending each fetched Occurrence out to an external system

1. In the Orchestration View window, create a new Message called ExternalSystemInstance with type Schemas -> Select from Referenced assemblies -> QPulseBizTalkSchemas -> QPulseBizTalkSchemas -> OccurrenceToExternalSystem.xsd.
2. Drag a Port component on to the right-hand side of the diagram. Name it ToExternalSystemPort, choose 'Create a new Port Type' and set the name to ToExternalSystemType, leave the Communication Pattern at On-Way and the Access Restrictions at Internal. Change the 'Port direction of communication' to 'I'll always be sending....' and choose 'Specify now', set the URI to 'C:\Projects\FileDrop\ToExternalSystem \%\MessageID%.xml', set the Transport to 'FILE' and the Pipeline should be set to 'Microsoft.BizTalk.DefaultPipeline.PassThruTransmit'.
3. In QPulseBizTalkOrchestrations, create a new map MapToExternalSystem.btm, click Open Source and QPulseBizTalkSchemas -> Schemas -> QPulseBizTalkSchemas.OccurrenceService.Reference. Then choose Occurrence as the Root Node. For the destination schema, choose QPulseBizTalkSchemas -> Schemas -> QPulseBizTalkSchemas.ToExternalSystem. Map the ID and Title fields of the source schema to the destination schema.
4. Drag a 'Transform' component onto the diagram directly below Receive\_GetOccurrenceResponse. Click on the outer Construct box and rename it to 'ConstructToExternalSystem'. Set the Messages Constructed property to ExternalSystemInstance.
5. Click on the inner Transform box and rename it to 'MapToExternalSystem'. In the Properties pane, click on Input Messages to show the Transform Configuration dialog. Choose 'New Map' and name it QPulseBizTalkOrchestrations.MapToExternalSystem. Still in the Transform Configuration dialog, in the Source Transform section, set the Variable Name to 'GetOccurrenceResponseInstance' (it should pop up) and in the Destination section, set the Variable Name to 'ExternalSystemInstance'.
6. Drag a Send component on to the diagram directly beneath Receive\_GetOccurrenceResponse, rename it to Send\_ToExternalSystem and set the Message property to ExternalSystemInstance. Connect it to the Request section of the ToExternalSystemPort.

## Final Steps

1. Drag an Expression component from the Toolbox on to the diagram below Send\_ToExternalSystem. Rename it to UpdateLists and in the Expression Editor, enter:  
`IDsProcessed.Add((System.Int32)IDsToProcess[0]);`  
`IDsToProcess.RemoveAt(0);`
2. Below UpdateLists but outside Loop\_FetchOccurrencesNotSeen, drag a Delay component on to the diagram, rename it Delay\_PollingTime and in the Expression Editor, enter:  
`new System.TimeSpan(1, 0, 0);`  
 This will cause the loop to wait an hour before proceeding with execution.
3. Drag a Loop component on to the diagram below Send\_StartConfirmation and rename it to Loop\_PollForNewOccurrences. In the Expression Editor, enter 'true' to cause the loop to execute indefinitely.
4. Drag all components from Construct\_AuthenticationRequest to Delay\_PollingTime into Loop\_PollForNewOccurrences, making sure that the original order is maintained.

## Deploying the Application

1. If the application has been previously deployed, please see the Undeploying section to properly remove the previous deployment before proceeding.
2. Make sure each project has the signing key set and the Application Name is set to 'QPulseBizTalkApplication'.
3. Right-click on the Solution in Solution Explorer and go to Properties -> Configuration Properties -> Configuration and tick the deploy tick-boxes for both projects.
4. Go to Build -> Deploy Solution which should succeed.
5. Drag and drop the built QPulseBizTalk assembly into the Global Assembly Cache ('C:\windows\assembly').
6. Go to Start -> Microsoft BizTalk Server 2006 -> BizTalk Server Administration in order to display the BizTalk Administration console.
7. In the tree view, go to BizTalk Server 2006 Administration -> BizTalk Group -> Applications, right-click on QPulseBizTalkApplication and go to Configure which shows the Configure Application dialog.
8. Click on MandatoryOccurrenceFetchProcess and set the host to BizTalkServerApplication
9. In the BizTalk Server Administration Console, expand the tree and click on Host Instances under Platform Settings. Right-click on BizTalkServerApplication and choose Restart.

## Running the Application

If all steps have been followed accurately, the application should start when it receives an XML file with content:

```
<ns0:Orchestration xmlns:ns0="http://QPulseBizTalkSchemas.Start">
  <Start>true</Start>
</ns0:Orchestration>
```

1. Create a new text file called StartOrchestration.xml and copy and paste the XML above.
2. Copy and paste the file into 'C:\Projects\FileDrop\Start' (or whichever location you chose when defining the ports).

## Undeploying the Application

1. Go to Start -> Microsoft BizTalk Server 2006 -> BizTalk Server Administration in order to display the BizTalk Administration console.
2. In the tree view, go to BizTalk Server 2006 Administration -> BizTalk Group -> Applications, right-click on QPulseBizTalkApplication and click on Stop. Choose Full Stop and click the Stop button. Then right-click on it again and choose Delete.
3. Right-click on QPulseBizTalkApplication and choose Delete.
4. Expand the tree and click on Host Instances under Platform Settings. Right-click on BizTalkServerApplication and choose Restart.
5. In Windows Explorer, navigate to 'C:\Windows\Assembly', find QPulseBizTalkSchemas and QPulseBizTalkOrchestrations and uninstall them both.

## Limitations

As this is an example application, there are some limitations that are worth being aware. If this example was to be used in a real-world deployment scenario, the limitations would be considered for improvement.

- The List of Occurrence IDs are stored in memory, it may be worth persisting them to a database if the list grew large.
- If any Web Service call fails, the Orchestration will stop and require staring again. It would be advisable to handle the error more elegantly and allow some form of recovery
- If the fetching of Occurrence failed, (due to no longer existing in the database) an exception would occur and the orchestration instance would fail.
- The `GetIDsNotProcessed` method could be improved by using a more efficient list comparison algorithm.

## Future Recommendations

The following are recommendations that could be desirable if improving the existing solution:

- Allow the polling time interval delay to be passed in with the initial start message. This would require a simple change in schemas and an additional Expression component to store the value for later use.

## Useful Links

BizTalk 2006 MSDN tutorials

<http://www.microsoft.com/downloads/details.aspx?FamilyID=6A5F6EF4-AEB8-4D8D-A521-37333A875CE4&displaylang=en>

BizTalk Server 2006 Best Practices Analyzer <http://go.microsoft.com/fwlink/?LinkId=83317>

BizTalk Server Developer Center <http://msdn.microsoft.com/en-us/biztalk/default.aspx>



**Gael Ltd.**

Orion House,  
S. E. Technology Park,  
East Kilbride,  
Scotland. G75 0RD

**t:** +44 1355 593400

**f:** +44 1355 579191

**e:** [info@gaelquality.com](mailto:info@gaelquality.com)

**w:** [www.gaelquality.com](http://www.gaelquality.com)

Q-Pulse is a registered trademark of Gael Products Ltd. All rights reserved worldwide.  
Copyright © 2011 Gael Products Ltd. Gael Quality is a trading division of Gael Ltd.