Building Data Integrator Real-time Jobs and Calling Web Services

Applies to:
SAP BusinessObjects Data Integrator 11.7.0.0
For more information, visit the Business Objects homepage.

Summary
This white paper shows how to construct a real-time job, expose it via a web service, and invoke a web service using Data Integrator (DI). It follows on from a previous white paper – Manipulating XML using Data Integrator.

The supplemental files referred to in this article are available for download on the SAP Community Network.

Author: Michael Eaton
Company: SAP
Created on: 15 March 2009
Table of Contents

Introduction .........................................................................................................................................................3
Ensuring the Correct Configuration ....................................................................................................................3
Sample Use Case ...............................................................................................................................................7
Building the Real-time Job .................................................................................................................................7
Testing the Real-time Job ....................................................................................................................................17
Configuring the Real-time Service ....................................................................................................................19
Exposing the Real-time Job as a Web Service ................................................................................................23
Importing the Web Service ...............................................................................................................................25
Building a Batch Job to Call a Web Service .....................................................................................................28
Executing the Batch Job ...................................................................................................................................34
Related Content ................................................................................................................................................36
Copyright ...........................................................................................................................................................37
Introduction

This white paper shows how to construct a real-time job, expose it via a web service, and invoke a web service using Data Integrator (DI). It follows on from a previous white paper – *Manipulating XML using Data Integrator*. This is not a DI tutorial and knowledge of building DI batch jobs is required. However, I have assumed that the audience has no real-time knowledge.

In the following DI examples, all the sample files have been written to local folder `c:\documents`. If you are going to follow the examples, place the sample files into a location where your job server can read them, and use that location when building the examples. You must have completed the previous white paper (mentioned above) if you plan to build the examples.

Ensuring the Correct Configuration

To ensure real-time jobs and adapters work correctly, the following minor configuration tasks must be completed:

1. The repository must be associated with a web administrator.
2. A job server must be present with adapter communication enabled.
3. An access server must be running and associated with a web administrator.

To associate a repository with a web administrator, use the Management > Repositories node in the web administrator as shown in Figure 1:

![Figure 1 - Associating a Repository with a Web Administrator](image-url)
Figure 2 shows the Job Server Properties dialog, the checkbox in the top pane must be checked to support adapter communication.
The screenshot below shows the server manager, the third section down allows an access server to be configured.

![Server Manager Screenshot]

_Figure 3 - Server manager_
If you performed a typical installation, a WebService adapter will already be configured. Figure 4 shows the Adapter Instances portion of the administrator. If the adapter is not configured, add one using the adapter configuration tab.

![Figure 4 - Adapter Instances](image)

The screenshot in Figure 5 shows the status page of the web administrator with everything setup and configured correctly.

![Figure 5 - Fully Configured Web Administrator](image)
Sample Use Case

To demonstrate the real-time and web service features of DI, the use case we'll use is a very simple address lookup service. A consumer will create a request containing a postcode, and the service will return all matching addresses for that postcode. The sample data from the previous white paper will be used.

Building the Real-time Job

At runtime, real-time jobs accept an XML message as their input and return an XML message as their output. The structure of these messages is defined using an XML schema (or DTD), so the first thing we must do is to import source and target XML schemas. Two sample schemas are included in the supplemental zip file, postcodeLookupRequest.xsd and postcodeLookupReply.xsd.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xs:element name="postcodeLookupRequest">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="postcode">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:maxLength value="7"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xs:element name="postcodeLookupReply">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="postcode">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:maxLength value="7"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```
We import these XML definitions by creating a new XML Schema object in DI. In the following examples I import the definition from a local drive, whilst this is fine for the examples, in reality the location of the definition would be published within the domain where it would be used, and referenced by a URL or UNC.

1. From the main menu, select Project, New, XML Schema… as shown in Figure 6.

When importing the schema (see Figure 7) we must specify a name for it within DI (I use the schema name), the location of the schema, and the root element name.
2. Click **OK** to import the request schema.

**Figure 7 - Importing The Request Schema**

DI imports the definition and it can be viewed in the local object library under the formats tab, as shown in **Figure 8**.

**Figure 8 - Formats Tab**
3. Double-click the schema to display the NRDM interpretation.

<table>
<thead>
<tr>
<th>Schema:</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>postcodeLookupRequest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>postcode</td>
<td></td>
<td>varchar(7)</td>
</tr>
</tbody>
</table>

*Figure 9 - Request NRDM*

4. Repeat the above process to import the reply schema.

*Figure 10 - Importing the Reply Schema*

<table>
<thead>
<tr>
<th>Schema:</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>postcodeLookupReply</td>
<td></td>
</tr>
<tr>
<td>postcode</td>
<td>varchar(7)</td>
</tr>
<tr>
<td>postcodeLookupReply_nt_1</td>
<td>varchar(50)</td>
</tr>
</tbody>
</table>

*Figure 11 - Reply NRDM*

Once we have imported the request and reply schemas when can build the real-time job.
5. Create a new real-time job (right click in a project and select **New Real-time Job**) and call it “RT_Postcode_Lookup”.

A real-time job contains placeholders that denote the code that will be executed in real-time. Any code between these placeholders will be executed for each XML message received. See Figure 12 for the empty real-time job.

![Figure 12 - New real-time Job](image)

6. Add a new dataflow (**DF_Postcode_Lookup**) and connect it up to the placeholders as shown in Figure 13.

![Figure 13 – New Dataflow In Real-Time Job](image)
7. Open the dataflow and drag the request schema from the object library onto the dataflow. Select Make XML Message Source from the popup menu as shown in Figure 14.

![Figure 14 - Add Message Source To Dataflow](image)

8. Drag the reply schema onto the dataflow and make that an XML message target.

![Figure 15 - Add Message Target to Dataflow](image)

9. Add demo_postcode and demo_address tables (from the previous white paper) to the dataflow as sources.

![Figure 16 - Add Table Sources to Dataflow](image)
10. Add a query (get_addresses) and connect all the sources to it.

![Diagram](image)

*Figure 17 - Adding get_addresses Query*

Our real-time dataflow must get all addresses for the postcode that is passed in the request message.

11. Open the query and specify the where clause shown in Figure 18.

12. Drag the ADDRESS column from the DEMO_ADDRESS table to the output schema.

![Table](image)

*Figure 18 - Get_Addresses Query Detail*
13. Add another query to the dataflow and connect the request and the output from the get_addresses query to it (as well as the reply message) as shown in Figure 19.

![Figure 19 - Adding the build_reply query](image)

14. Open the `build_reply` query and complete the mapping as follows:

i. The `from` clause for the `build_reply` node (in the output schema) should only contain `postcodeLookupRequest`. You should remove the `get_addresses` source. (Figure 20)

ii. Map the `postcode` output column to `postcodeLookupRequest.postcode` (Figure 21)

iii. The `from` clause for the `postcodeLookupReply_nt_1` node should contain the `get_addresses` source (remember to make current first) (Figure 22)

iv. Map the address output column to `get_addresses.ADDRESS` (Figure 23)

![Figure 20 – build_reply Node](image)
Figure 21 - Postcode Node Mapping

Figure 22 - postcodeLookupReply.nt_1 from clause
The reason this query is mapped like this is because a postcode might not generate any address records, that is why the output postcode is mapped from the request message. A where clause is not required in this NRDM as we have already filtered the data in the get_addresses query. The reason this dataflow is broken down into two queries is purely for clarity.

The real-time job is now complete.
Testing the Real-time Job

The real-time job is complete, but it has not been tested yet. DI allows us to functionally test a real-time job by running it as a batch job. We must specify an XML file to be used as the request (which must exist) and the name of a target XML file that will be created as the response. A sample request file, postcodeLookupRequest.xml is included in the ZIP file.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample request to test DI real-time job, created by Michael Eaton. -->
<postcodeLookupRequest>
  <postcode>DE150LQ</postcode>
</postcodeLookupRequest>
```

1. Open the editor for the request message (postcodeLookupRequest) and specify this file as the XML test file: as shown in Figure 24.

![Figure 24 - Specifying the Request Test File](image)
2. Open the editor for the response message (postcodeLookupReply) and provide a filename for the XML test file; this file does not have to exist as DI will create it. See Figure 25.

![Figure 25 - Response Test File Name](image)

When you execute the job, it should produce an XML file similar to the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- BusinessObjects Data Integrator generated XML -->
<postcodeLookupReply xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <postcode>DE150LQ</postcode>
  <address>Eaton Hall, Eaton Estate</address>
  <address>The Gardeners Cottage, Eaton Estate</address>
</postcodeLookupReply>
```

Try changing the postcode in the request XML file and re-running the job. If the postcode lookup does not find any matching addresses, it simply returns the request postcode, as shown below.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- BusinessObjects Data Integrator generated XML -->
<postcodeLookupReply xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <postcode>zz1</postcode>
</postcodeLookupReply>
```
Configuring the Real-time Service

Now that the real-time job has been tested, we can now execute it. This is done by configuring a real-time service from the web administrator. Open the administrator and navigate to the Real-Time Services node as shown in Figure 26.

1. Click the Real-Time Services Configuration tab and then click the Add button as shown in Figures 27 and 28.
2. Click the **Browse Jobs** button and select the real-time job we created earlier, giving the service a suitable name (Figure 29).
3. Click the **Apply** button to create the real-time service.

![Figure 30 - Provider Configuration](image)

It is possible to configure multiple providers for each service. The default is one provider, which will process all messages for the service in series. If more than one provider is configured each provider will process messages in parallel with other providers, allowing for an increase in throughput.

4. Click the **Real-Time Services** node to see the new service (Figure 31).

![Figure 31 - New Real-Time Service](image)
5. Select the service and click the **Start** button.

![Real-Time Service Started](image)

**Figure 32 - Real-Time Service Started**

It is possible to configure multiple real-time services that use the same real-time job. An example configuration might be two real-time services, where each one uses a different system configuration. One service might be called *Lookup_UK* and the other *Lookup_NI*. Each service would use separate database (controlled via the separate system configurations) for their lookup data.

Now that the real-time service is started, it is available to process messages (to perform postcode lookups) from various types of clients. For example, messages might be sent from a DI JMS (Java Message Service) adapter instance, a custom client written in Java, C++, etc. using the DI message client library (a very simple API), or via DI's web service (covered in the following sections).
Exposing the Real-time Job as a Web Service

DI allows real-time services to be exposed via DI’s web service so that web service clients can invoke them. Each real-time service that is published via this mechanism appears as an operation in DI’s WSDL.

In a typical single machine installation the WSDL URL would be http://localhost:28080/diAdmin/servlet/webservices?wsdlxml.

1. To publish the postcode lookup real-time service, click the Web Services node in the web administrator (Figure 33).

![Figure 33 - Web Services](image1)

2. Click the Web Services Configuration tab (Figure 34).

![Figure 34 - Web Services Configuration](image2)

3. Click the Add Real-time Service button. The PostcodeLookup real-time service appears in the list of real-time services.
4. Select it and click the **Add** button as shown in Figure 35.

![Figure 35 - Add Real-Time Service](image)

The real-time service is now published with the operation name *PostcodeLookup* (as shown in Figure 36). It may take a few moments to appear in the list.

![Figure 36 - Published Operations](image)

Our real-time service can now be invoked by web service clients. Web service clients can be written in a variety of languages and execute on various platforms, this means that the postcode lookup functionality can be consumed by a wide range of processes.
Importing the Web Service

The previous sections covered writing and publishing the postcode lookup functionality as a web service operation. As well as publishing web services, DI is capable of consuming web services. The next sections describe how to invoke the postcode lookup service, although a web service written in any language running on any platform can be invoked in exactly the same way.

Before we can import the metadata for a web service, a web service adapter must be configured (as covered in a previous section). Web service metadata is imported via a datastore, so the first step is to create a new datastore.

Figure 37 - Create New Datastore
1. The Datastore type must be *Adapter*. Job Server and Adapter instance name will be specific to each environment. URL of Web Service (actually this is to specify the location of the WSDL) should be `http://localhost:28080/diAdmin/servlet/webservices?wsdlxml` (replace localhost:28080 with your web admin host and port).

![Create New Datastore](image)

**Figure 38 - Create Datastore**

2. Click **OK** to create the new datastore and then double-click the new datastore in the object library.

![Local Object Library](image)

**Figure 39 - New Datastore In Library**

DI should read the WSDL and display the metadata, as shown below.
3. Right-click the **PostcodeLookup** operation and import it. The imported operation appears as a function in the datastore.

![Import Operation](image)

*Figure 40 - Import Operation*

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DM_TARGET</td>
<td>Name: DM_Target\Modu...</td>
</tr>
<tr>
<td>DS_Postcode Lookup</td>
<td></td>
</tr>
<tr>
<td>Functions</td>
<td></td>
</tr>
<tr>
<td>PostcodeLookup(DS_Postcode_Lookup...)</td>
<td>for operation: P...</td>
</tr>
<tr>
<td>DS_Test</td>
<td></td>
</tr>
<tr>
<td>New_Dataset</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 41 - The Imported Function*

The web service operation can now be invoked.
Building a Batch Job to Call a Web Service

1. Create a new batch job with a single dataflow.

![Batch job to call WS](image1)

**Figure 42 - batch job to call WS**

2. Within the dataflow add a row generation as the source and a query.

![Row Generation](image2)

**Figure 43 – Row Generation**

When performing a function call in DI, the input schema must match the function input schema, and it can be tricky to create the exact structure. The easiest way to do this is to copy it from the function metadata.
3. Copy the `postcodeLookupRequest` node from the `$REQUEST_SCHEMA` as shown in Figure 44 and paste in into the output schema of `nest_data` as shown in Figure 45.
4. Make the `postcodeLookupRequest` node current and add `Row_Generation` to the `from` clause.

![Figure 46 - From Clause For postcodeLookupRequest Node](image)

5. Specify a hard-coded postcode for the `postcode` node.

![Figure 47 - Mapping for Postcode Node](image)

6. Add another query to the dataflow.

![Figure 48 - Invoke Query](image)
7. Within the second query, we now use the imported web service function, right-click and choose **New Function Call**.

![Figure 49 - New Function Call](image)

8. Select the **postcodeLookup** function from the adapter datastore and click **Next**.

![Figure 50 - Select Function](image)
9. We must now map the schemas, the easiest way to do this is to drag the postcodeLookupRequest node from the input schema onto the same node in the input schema of the function (see Figure 51).

![Figure 51 - Define Input Parameters](image)

10. Click Finish. The mapping is complete as shown in Figure 52.

![Figure 52 - Function Call Mapping](image)
11. Add an XML Template to the dataflow to capture the output of the function call. Note that we are only using this to see the results, in reality the output might be fed into other queries and tables, for example.

![Diagram of Dataflow]

\textit{Figure 53 - Add XML Template}

12. Specify an output filename for the XML template. The batch job is now complete.

![Screenshot of XML Template Filename]

\textit{Figure 54 - XML Template Filename}
Executing the Batch Job

Execute the batch job as you would normally and then open the output file specified for the template XML. You should see the request postcode (specified in the query) together with the relevant addresses from the lookup tables. If you do, you have successfully called the DI web service (and the real-time service) to perform the lookup.

Figure 55 - Lookup Result

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<write_result>
  - <postcodeLookupReply>
    <postcode>DE15QP</postcode>
    - <postcodeLookupReply_nt_1>
      <address>Eaton Hall, Eaton Estate</address>
    </postcodeLookupReply_nt_1>
    - <postcodeLookupReply_NT_1>
      <address>The Gardeners Cottage, Eaton Estate</address>
    </postcodeLookupReply_NT_1>
  </postcodeLookupReply>
</write_result>
```
In the web administrator you see that the requests have changed from 0 to 1 for the real-time service and the web service operation.

Try changing the hard-coded request postcode and re-running the job.
Related Content

BusinessObjects Information Management Community

Supplemental Files for this Article
Building Data Integrator Real-time Jobs and Calling Web Services

Copyright

© Copyright 2009 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG.

The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Excel, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, System i, System i5, System p, System p5, System x, System z, System z10, System z910, z10, z9, iSeries, pSeries, xSeries, zSeries, eServer, zVM, z/OS, i5/OS, S/390, OS/390, OS/400, AS/400, S/390 Parallel Enterprise Server, PowerVM, Power Architecture, POWER6+, POWER6, POWER5+, POWER5, POWER, OpenPower, PowerPC, BatchPipes, BladeCenter, System Storage, GPFS, HACMP, RETAIN, DB2 Connect, RACF, Redbooks, OS/2, Parallel Sysplex, MVS/ESA, AIX, Intelligent Miner, WebSphere, Netfinity, Tivoli and Informix are trademarks or registered trademarks of IBM Corporation.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

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

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C®, World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

SAP, R/3, SAP NetWeaver, Duet, PartnerEdge, ByDesign, SAP Business ByDesign, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and other countries.

Business Objects and the Business Objects logo, BusinessObjects, Crystal Reports, Crystal Decisions, Web Intelligence, Xcelsius, and other Business Objects products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of Business Objects S.A. in the United States and in other countries. Business Objects is an SAP company.

All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.