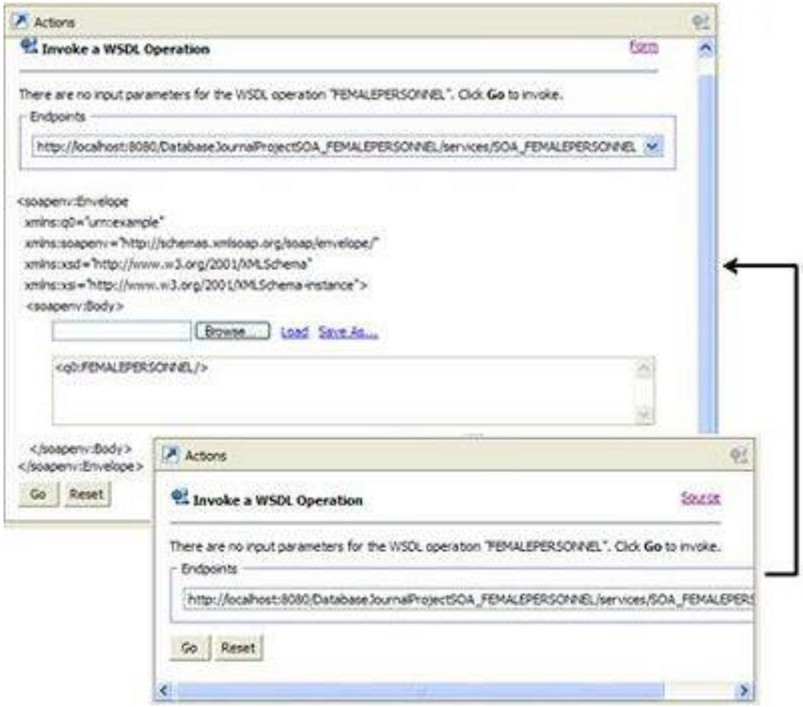
How-To: Test a Web Service using the Web Services Explorer IBM Data Studio Part 2

If you consider the transactional nature of Web services and the new era of application opportunities that they create, it should become clear why the handling of XML data in a data server is poised to become one of the most differentiating features of a data server. This is one of the main reasons why IBM invested in the pureXML technology found in DB2 software because it enables IT shops to store and search the XML in ways never available before. Now imagine a business that wants to store its credit risk scores or transactions for governance and reporting purposes; that business operation's output is in XML if it comes from a SOAP-invoked Web service and the actual transaction can be stored in DB2 software with all its fidelity in its natural form.

To return to the form view, click **Form**.

You use the **Actions** window to test the Web service. This pane can display either the source code or the form that IBM Data Studio automatically provides to test the Web service. As with the **Status** pane, you can toggle between the form and the source views using the appropriate links:



There are no input parameters for the WSDL operation "FEMALEPERSONNEL". Click **Go** to invoke.

http://localhost:8080/DatabaseJournalProjectSOA_FEMALEPERSONNEL/services/SOA_FEMALEPERSONNEL

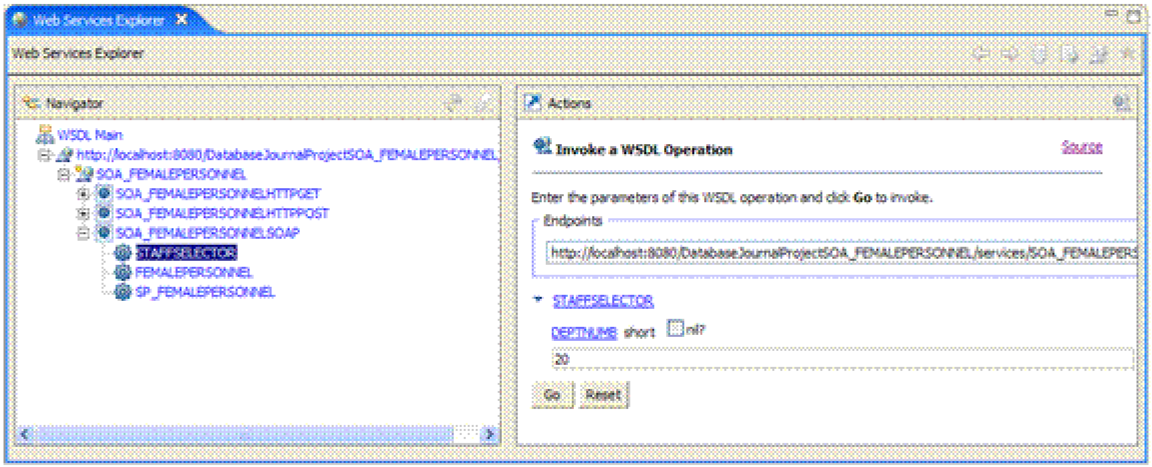
```
<soapenv:Envelope
 xmlns:q0="urn:example"
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
    <q0:FEMALEPERSONNEL/>
 </soapenv:Body>
</soapenv:Envelope>
```

There are no input parameters for the WSDL operation "FEMALEPERSONNEL". Click **Go** to invoke.

http://localhost:8080/DatabaseJournalProjectSOA_FEMALEPERSONNEL/services/SOA_FEMALEPERS

The automatic form generation to test Web services in IBM Data Studio is a very useful feature that allows database administrators (DBAs) to quickly test their Web services without the need for help from a development team to invoke the service. This way, DBAs can be assured that the Web service is returning data that they expect before passing it to the Web development team; they don't have to interrupt the Web development for basic testing.

The Web services built in this tutorial don't take input parameters, and therefore the benefits of the auto-form generation may not be as apparent. Consider exposing a stored procedure, called STAFFSELECTOR, that was built from the following SQL statement: SELECT * FROM STAFF WHERE DEPT=:DEPTNUMB. This stored procedure returns personnel details of an enterprise's staff based on a department number. If you exposed this stored procedure as a Web service, IBM Data Studio would auto-generate a form that looks similar to this one:



Enter the parameters of this WSDL operation and click **Go** to invoke.

http://localhost:8080/DatabaseJournalProjectSOA_FEMALEPERSONNEL/services/SOA_FEMALEPERS

DEPTNUMB short □ nil?

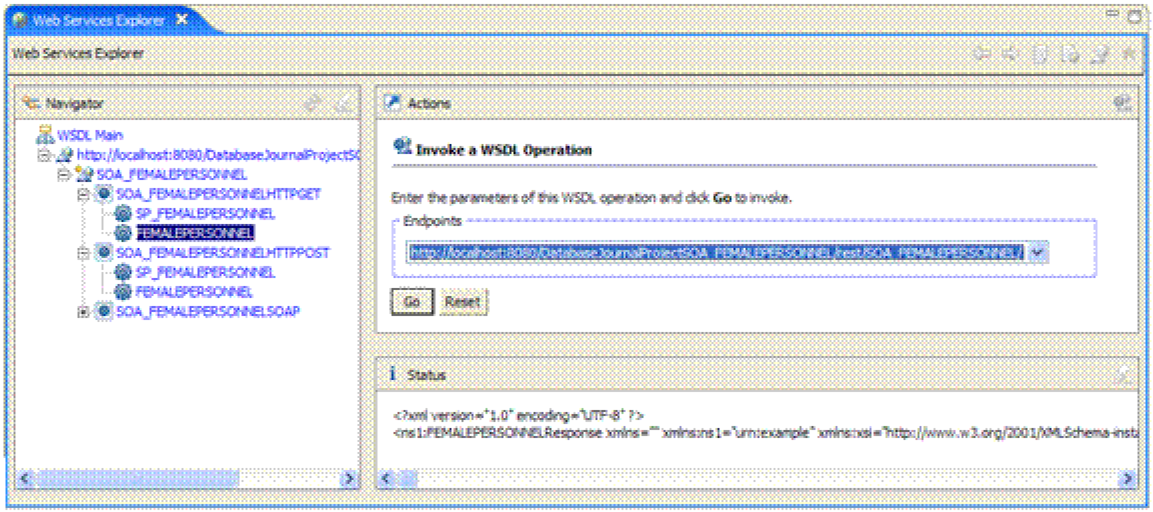
20

As you can see in the previous figure, the auto-generated form has the **DEPTNUMB** field where you can specify the input parameter. To test this Web service, a DBA simply enters an input parameter in this field and clicks **Go** to pass the input parameter to the Web service and invoke it.

So far I've shown you how to test your SOAP-based Web services. You can use the Web Services Explorer to test your REST-based Web services as well. It's outside the scope of this article to delve into the technical aspects of REST-based Web services, but they are invoked using HTTP GET and POST methods using a simple URL. (This should be apparent if you look closely at the artifacts that IBM Data Studio automatically creates for you in the **Navigator** pane.)

You can see in the previous figure that the **Navigator** pane in the Web Services Explorer has three expandable artifacts. Each has a suffix that identifies the way in which the Web service will be invoked (**SOA_FEMALEPERSONNELHTTPGET** method using REST, **SOA_FEMALEPERSONNEL HTTPPOST** method using REST, or **SOA_FEMALEPERSONNEL SOAP** using SOAP).

For example, to test your Web service using the REST protocol's GET method, expand the appropriate artifact in the **Navigator** pane and invoke it in the same manner you tested the SOAP invocation of the same Web service:



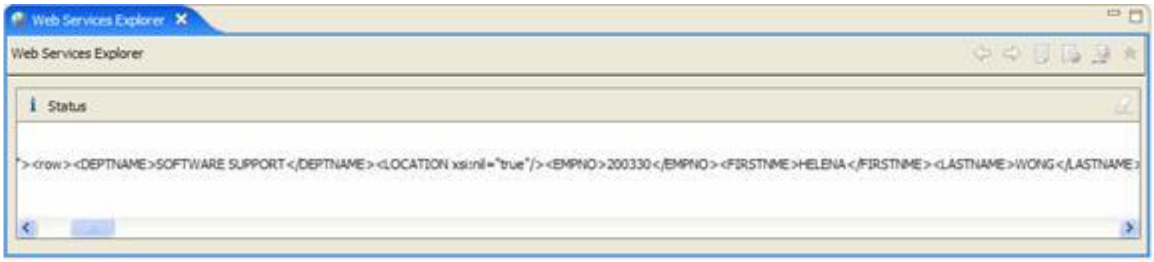
Enter the parameters of this WSDL operation and click **Go** to invoke.

Endpoints: http://localhost:8080/DatabaseJournalProjectSOA_FEMALEPERSONNEL/rest/SOA_FEMALEPERSONNEL/

```
<?xml version="1.0" encoding="UTF-8" ?>
<ns1:FEMALEPERSONNELResponse xmlns="" xmlns:ns1="urn:example" xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
```

Notice that when you invoke this Web service using the REST protocol, there is no **Source** link as was the case in the SOAP-invoked Web service test. That's because when you invoke a Web service using the REST protocol, you simply pass a URL from the requesting application to the data server.

You can see the output of a REST-based Web service is also in XML format:



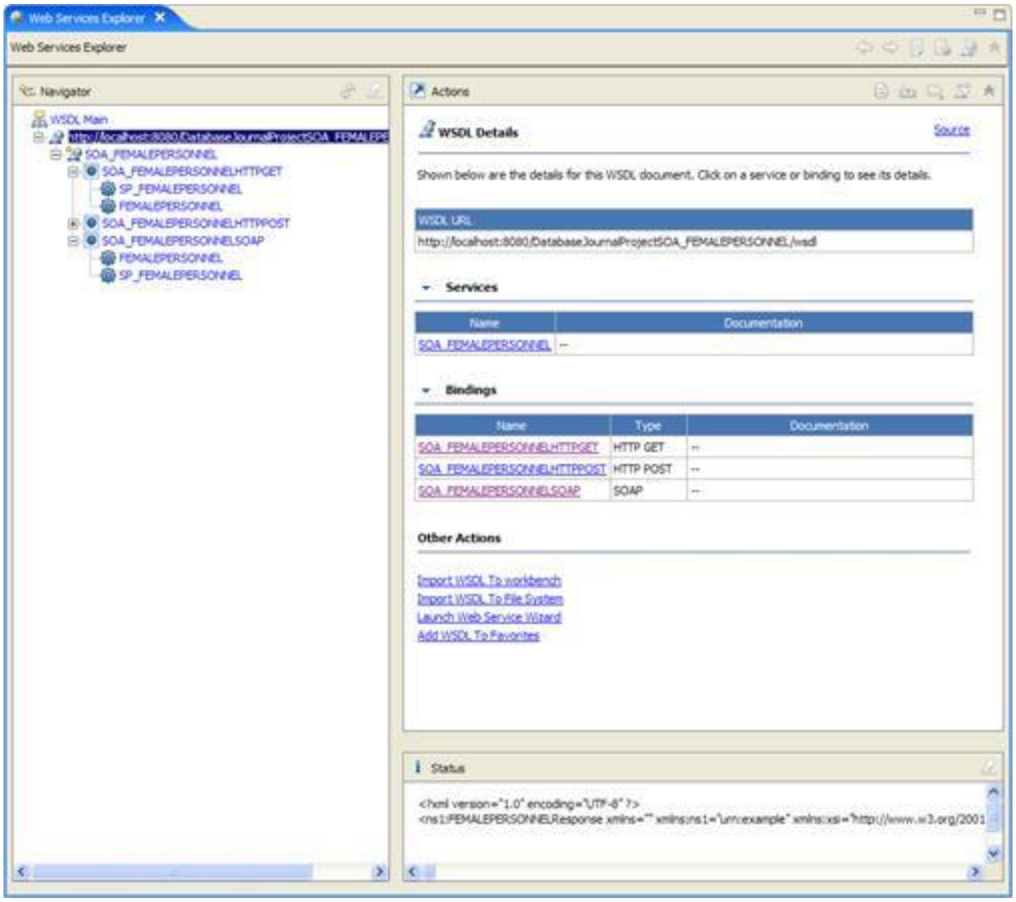
```
"><row><DEPTNAME>SOFTWARE SUPPORT</DEPTNAME><LOCATION xsi:nil="true"/><EMPNO>200330</EMPNO><FIRSTNME>HELENA</FIRSTNME><LASTNAME>WONG</LASTNAME>
```

As I've previously noted, REST-based Web services are invoked with a simple URL; therefore, it stands to reason that you should be able to open a Web browser and invoke the Web service that you built and locally deployed on the started IBM WebSphere Application Server Community Edition application server that's running on your server.

IBM Data Studio generated a WSDL file for your Web service invocations, and within that file you can quickly copy and paste the URL for the corresponding Web service. The convenience of this feature will help immensely when you test your Web service using a REST-based invocation.

To test your REST-based Web services directly from a Web browser, perform the following steps:

1. Select the URL of the deployed Web service project, as shown below:



Shown below are the details for this WSDL document. Click on a service or binding to see its details.

WSDL URL: http://localhost:8080/DatabaseJournalProjectSOA_FEMALEPERSONNEL/wsdl

Name	Documentation
SOA_FEMALEPERSONNEL	--

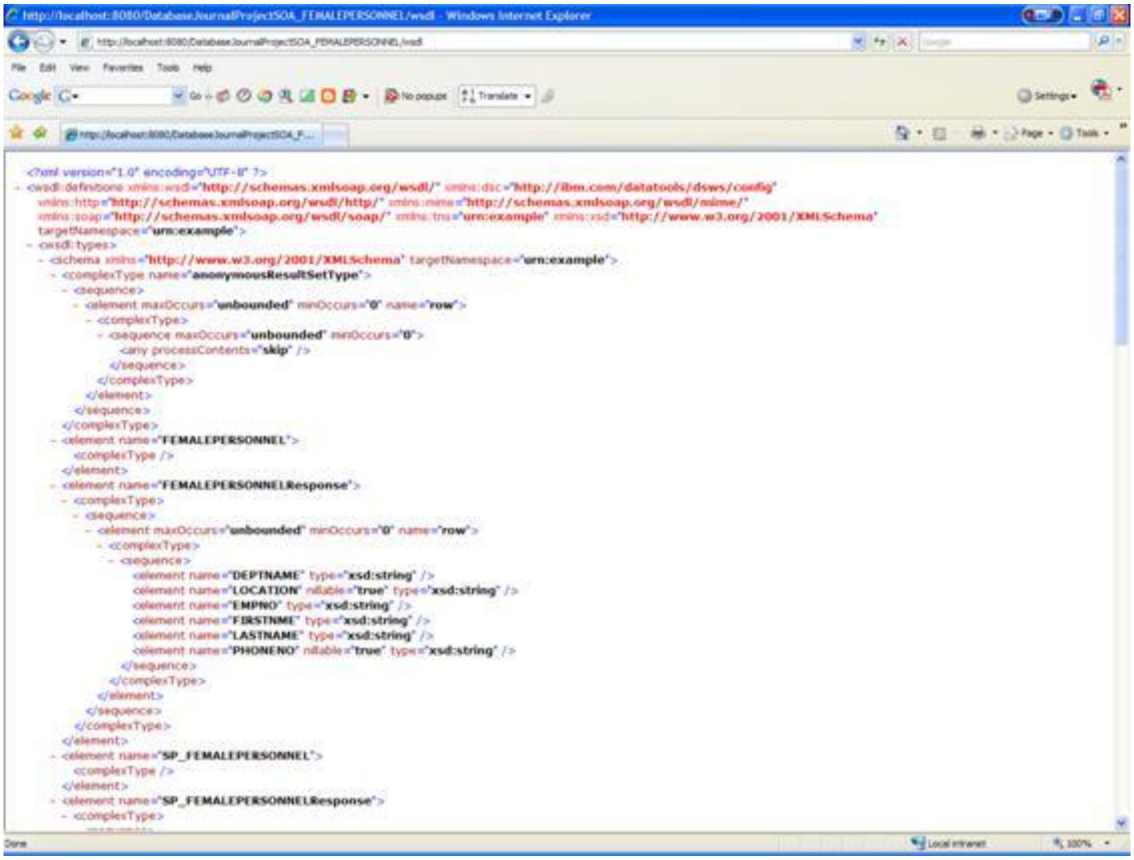
Name	Type	Documentation
SOA_FEMALEPERSONNELHTTPGET	HTTP GET	--
SOA_FEMALEPERSONNELHTTPPOST	HTTP POST	--
SOA_FEMALEPERSONNELSOAP	SOAP	--

```
<?xml version="1.0" encoding="UTF-8" ?>
<ns1:FEMALEPERSONNELResponse xmlns="" xmlns:ns1="urn:example" xmlns:xsi="http://www.w3.org/2001
```

2. Copy and paste the URL from the **WSDL URL** field in the **Actions** pane. In this example, the URL that points to the WSDL file is: http://localhost:8080/DatabaseJournalProjectSOA_FEMALEPERSONNEL/wsdl.

3. Open a Web browser window.

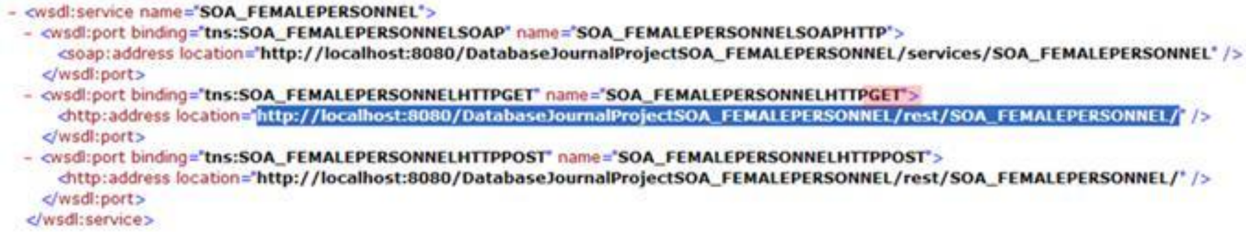
4. Enter the URL from Step 2 into the browser's address field, and press **Enter**. The WSDL file is displayed:



http://localhost:8080/DatabaseJournalProjectSOA_FEMALEPERSONNEL/wsdl - Windows Internet Explorer
File Edit View Favorites Tools Help
Google
No popups
Translate
Settings
Page
Tools
Done
Local intranet
100%

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:dsc="http://ibm.com/datatools/dsws/config"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="urn:example" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="urn:example">
  - <wsdl:types>
    - <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:example">
      - <complexType name="anonymousResultSetType">
        - <sequence>
          - <element maxOccurs="unbounded" minOccurs="0" name="row">
            - <complexType>
              - <sequence maxOccurs="unbounded" minOccurs="0">
                  <any processContents="skip" />
                </sequence>
              </complexType>
            </element>
          </sequence>
        </complexType>
      - <element name="FEMALEPERSONNEL">
          <complexType />
        </element>
      - <element name="FEMALEPERSONNELResponse">
        - <complexType>
          - <sequence>
            - <element maxOccurs="unbounded" minOccurs="0" name="row">
              - <complexType>
                - <sequence>
                    <element name="DEPTNAME" type="xsd:string" />
                    <element name="LOCATION" nillable="true" type="xsd:string" />
                    <element name="EMPNO" type="xsd:string" />
                    <element name="FIRSTNME" type="xsd:string" />
                    <element name="LASTNAME" type="xsd:string" />
                    <element name="PHONENO" nillable="true" type="xsd:string" />
                  </sequence>
                </complexType>
              </element>
            </sequence>
          </complexType>
        </element>
      - <element name="SP_FEMALEPERSONNEL">
          <complexType />
        </element>
      - <element name="SP_FEMALEPERSONNELResponse">
        - <complexType>
```

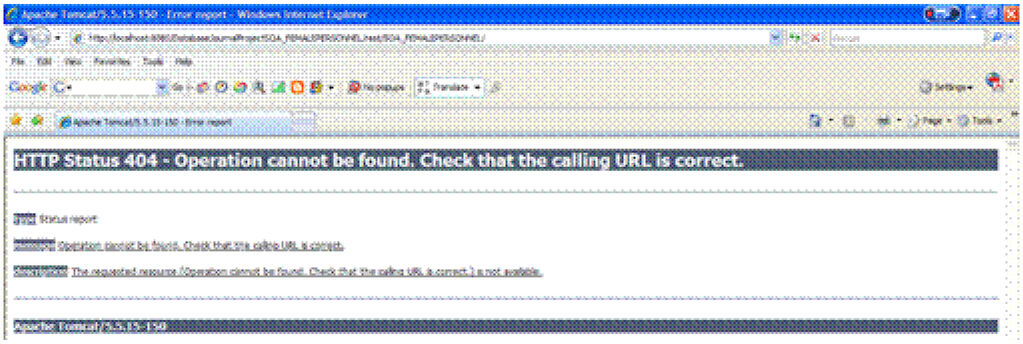
5. Since we want to invoke our Web service using the REST protocol and the associated HTTP GET method, scroll to the bottom of the WSDL file and locate the <http:address location> element for the GET <wsdl:port binding> invocation of the Web service:



```
- <wsdl:service name="SOA_FEMALEPERSONNEL">
  - <wsdl:port binding="tns:SOA_FEMALEPERSONNELSOAP" name="SOA_FEMALEPERSONNELSOAPHTTP">
      <soap:address location="http://localhost:8080/DatabaseJournalProjectSOA_FEMALEPERSONNEL/services/SOA_FEMALEPERSONNEL" />
    </wsdl:port>
  - <wsdl:port binding="tns:SOA_FEMALEPERSONNELHTTPGET" name="SOA_FEMALEPERSONNELHTTPGET">
      <http:address location="http://localhost:8080/DatabaseJournalProjectSOA_FEMALEPERSONNEL/rest/SOA_FEMALEPERSONNEL/" />
    </wsdl:port>
  - <wsdl:port binding="tns:SOA_FEMALEPERSONNELHTTPPOST" name="SOA_FEMALEPERSONNELHTTPPOST">
      <http:address location="http://localhost:8080/DatabaseJournalProjectSOA_FEMALEPERSONNEL/rest/SOA_FEMALEPERSONNEL/" />
    </wsdl:port>
  </wsdl:service>
```

<u>Note</u>: Ensure that you only select the URL highlighted in the previous figure; note that this <u>does not include</u> the quotation marks (" ") that surround the URL.

6. Copy and paste the URL from the previous step into your Web browser. This <u>will</u> create an error:



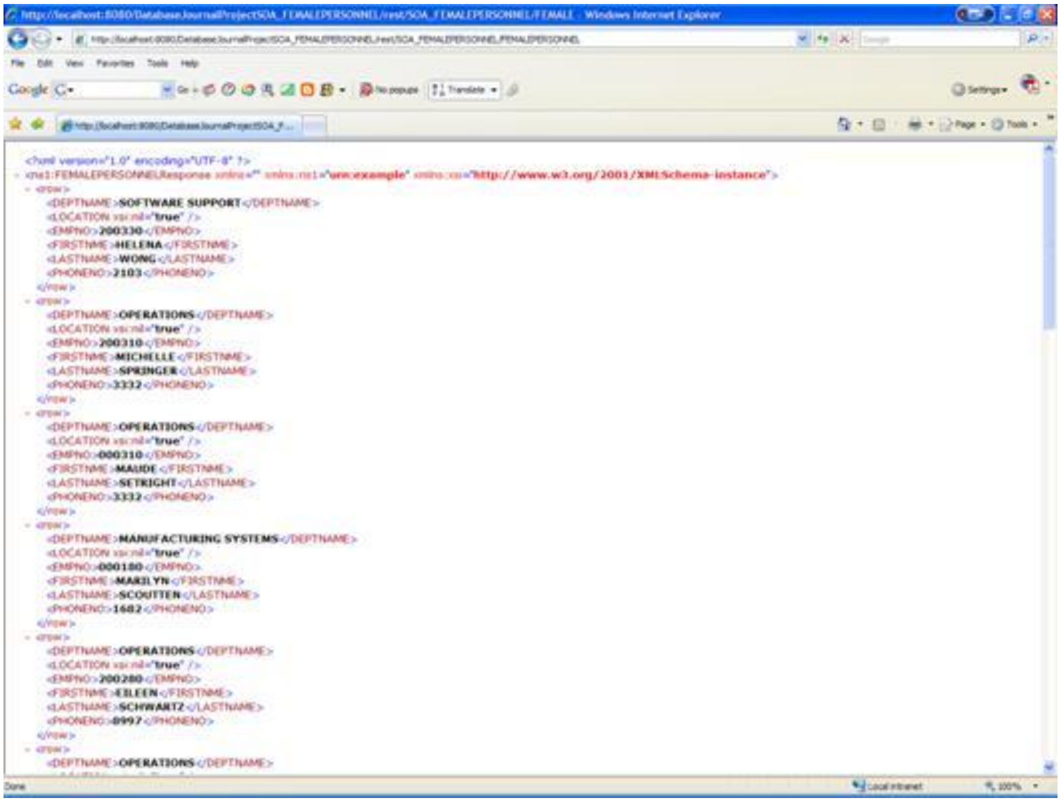
Apache Tomcat/5.5.15-150 - Error report - Windows Internet Explorer

```
HTTP Status 404 - Operation cannot be found. Check that the calling URL is correct.

type Status report

message Operation cannot be found. Check that the calling URL is correct.

description The requested resource (Operation cannot be found. Check that the calling URL is correct.) is not available.

Apache Tomcat/5.5.15-150
```

<u>Note</u>: I'm only including this step to make the invocation of the Web service more procedural and easier to follow. As you become more accustomed to calling REST-based Web services, you can simply skip this step.

7. Append the name of the Web service created in this series (FEMALPERSONNEL) to the URL entered in the previous step, and then press **Enter**. For example: http://localhost:8080/DatabaseJournalProjectSOA_FEMALEPERSONNEL/FEMALPERSONNEL.

As you can see, your Web service was invoked using the REST protocol directly from your Web browser:



http://localhost:8080/DatabaseJournalProjectSOA_FEMALEPERSONNEL/rest/SOA_FEMALEPERSONNEL/FEMALE - Windows Internet Explorer

```
<?xml version="1.0" encoding="UTF-8" ?>
- <ns1:FEMALEPERSONNELResponse xmlns="" xmlns:ns1="urn:example" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <row>
      <DEPTNAME>SOFTWARE SUPPORT</DEPTNAME>
      <LOCATION xsi:nil="true" />
      <EMPNO>200330</EMPNO>
      <FIRSTNME>HELENA</FIRSTNME>
      <LASTNAME>WONG</LASTNAME>
      <PHONENO>2103</PHONENO>
    </row>
  - <row>
      <DEPTNAME>OPERATIONS</DEPTNAME>
      <LOCATION xsi:nil="true" />
      <EMPNO>200310</EMPNO>
      <FIRSTNME>MICHELLE</FIRSTNME>
      <LASTNAME>SPRINGER</LASTNAME>
      <PHONENO>3332</PHONENO>
    </row>
  - <row>
      <DEPTNAME>OPERATIONS</DEPTNAME>
      <LOCATION xsi:nil="true" />
      <EMPNO>000310</EMPNO>
      <FIRSTNME>MAUDE</FIRSTNME>
      <LASTNAME>SETRIGHT</LASTNAME>
      <PHONENO>3332</PHONENO>
    </row>
  - <row>
      <DEPTNAME>MANUFACTURING SYSTEMS</DEPTNAME>
      <LOCATION xsi:nil="true" />
      <EMPNO>000180</EMPNO>
      <FIRSTNME>MARILYN</FIRSTNME>
      <LASTNAME>SCOUTTEN</LASTNAME>
      <PHONENO>1682</PHONENO>
    </row>
  - <row>
      <DEPTNAME>OPERATIONS</DEPTNAME>
      <LOCATION xsi:nil="true" />
      <EMPNO>200280</EMPNO>
      <FIRSTNME>EILEEN</FIRSTNME>
      <LASTNAME>SCHWARTZ</LASTNAME>
      <PHONENO>8997</PHONENO>
    </row>
  - <row>
      <DEPTNAME>OPERATIONS</DEPTNAME>
```

Wrapping it all up

In this article, I showed you how to use the IBM Web Services Explorer to test multiple message protocols to invoke the Web service that you created in this series. What's more, I showed you how easy it is to invoke your Web service using the REST protocol directly from any Web browser by leveraging the WSDL file that IBM Data Studio automatically created for you. In the next article, I'm going to introduce you to the Data Web Services Test Client; I feel this is a more feature-rich and capable testing tool, but it's only available for IBM Data Studio Version 1.2 and later, which is why I wanted to ensure that you knew how to use the IBM Web Services Explorer too.

Disclaimers

The opinions, solutions, and advice in this article are from the author's experiences and are not intended to represent official communication from IBM or an endorsement of any products listed within. Neither the author nor IBM is liable for any of the contents in this article. The accuracy of the information in this article is based on the author's knowledge at the time of writing.

Courtesy:
https://www.databasejournal.com/features/db2/article.php/10896_3789021_2/Testing-your-Web-Service-using-the-Web-Services-Explorer.htm

Modified: 2021.10.04.7.15.AM

Dököll Solutions, Inc.