

Build Lotus Web services into a portal

Make your apps easy to access for users

Skill Level: Intermediate

[George Brichacek \(george_brichacek@us.ibm.com\)](mailto:george_brichacek@us.ibm.com)

Technology Advocate

EMC

27 Mar 2003

In this tutorial, you'll learn how to create a portlet for WebSphere Portal that will access Domino using Web services. We'll walk you through each step of the process, showing you how to access your Domino applications from any Web browser.

Section 1. Introduction

What is this tutorial about?

In this tutorial, you'll learn how to create a portlet for WebSphere Portal that will access Lotus Domino using Web services. We will also set up Domino to host Web services -- in other words, we'll turn our Domino installation into a service provider.

Software requirements

The following software is required in order to complete the steps demonstrated in this tutorial:

- Lotus Domino Server 6, Lotus Notes 6 and Domino Designer 6 and Administrator 6.
[Free trial downloads of the latest versions](#) are available.

- WebSphere Studio V5
We are using the Application Developer configuration of WebSphere Studio V5, but other configurations will also work.
- Portal Toolkit 4.2.5
Download a free [trial version](#).

You should also download this tutorial's [sample code](#). All the sample packages, including WAR files, that we reference throughout this tutorial will be included in this zip archive.

Who should take this tutorial?

This tutorial will be useful for developers working in an environment where Lotus Domino is deployed and Domino applications have already been developed. You may find it particularly useful if you've already set up Web services on the Domino server as outlined in the tutorial "Building Web services for Domino 6" (see [Resources](#) for a link). We will provide an example here showing you how to create a portlet for WebSphere Portal that will access an application on Domino using Web services.

This tutorial is also intended for developers looking to extend their existing portal applications with portlets that will access Domino applications that have already been developed using Web services. Refer to the tutorial titled "Turn your Lotus applications into Web services" (see [Resources](#) for a link) for more information on what the Domino server can do for you.

After completing this tutorial, you should know how to deploy a portlet that accesses the collaborative capabilities of Lotus Domino. This is *contextual collaboration* -- the placing of collaborative Web services in business applications where they make sense.

Section 2. Contextual collaboration

Introduction

Today, collaboration tends to take place via collaborative applications like e-mail, discussions, or instant messaging, or within collaborative workspaces like Lotus QuickPlace. The user has to go to a specific location to collaborate with others. In the coming era, collaboration will become less a place and more a capability to

interact with others within any context of a business application.

For example, you're taking this tutorial, and you may want to interact with me directly. You should be able to find out whether or not I am online and willing to talk to you; if I am, then you should be able to open a chat session with me without leaving the context of the tutorial. You might want to provide feedback on this page of the tutorial and schedule a follow-up meeting to discuss it -- and you should be able to do all that without ever leaving the page, too.

Another important aspect of contextual collaboration is that it enables people to collaborate with one another from within different contexts. For instance, one person might be working in a project management application and want to collaborate with someone else who is using a CRM application. Ongoing discussion should be accessible from a variety of contexts, too, such as from non-PC devices like wireless phones and PDAs.

IBM is demonstrating contextual collaboration from within its WebSphere portal environment now. Users have access to their calendars, tasks, and inboxes, and can chat, discuss, and exchange instant messages with others, all directly from the portal interface.

IBM Lotus Software will make collaborative services available, serving them from the IBM WebSphere Application Server. Some of these collaborative services are:

- Presence awareness, which lets users know who is online and available to collaborate
- Instant messaging and chat
- Threaded discussion
- Calendar/scheduling
- E-mail inbox management
- Shared screens/applications
- User-created/managed collaborative workspaces
- Document management
- Routing and workflow

Contextual collaboration, as it was dubbed by analysts such as Meta and IDC, points to a new era of collaboration, in which these collaborative services are provided to business applications as on-demand Web services.

Book catalog scenario: A sample Web service

In the next section, we will create a portlet that will access Lotus Domino using Web services. We'll set up a book catalog Web service on Domino. (For more information about Web services and Lotus Domino, read the tutorial "Building Web services using Lotus Domino 6" -- see [Resources](#) for a link.) The book catalog is a Domino database that contains book documents, each containing the title, author, publisher, price, and number of available copies of each book. The key to a document is the International Standard Book Number (ISBN). The end user will enter an ISBN in our portlet, which will then access Domino via Web services; Domino will return the title, author, publisher, price, and number of available copies to our portlet.

We've provided the code for the book catalog database as part of this tutorial's sample; you can download it from [Software requirements](#).

We'll walk through the following steps to create our portlet:

- Create the Web service in Domino. The tutorial "Building Web services using Lotus Domino 6" (see [Resources](#)) goes into more detail than we will here.
- Create a portlet project in WebSphere Studio Application Developer.
- Create a Web service client in our portlet project.
- Create a JavaBeans component to store our input and output parameters.
- Create two JSPs, one for our request and one for our response.
- Create our portlet application.
- Export the portlet as a WAR file.
- Install the portlet in WebSphere Portal.

Section 3. Create a Web service in Domino

The book catalog application

The book catalog application demonstrates how to set up a Web service on Domino. A Web service requester (client) sends a SOAP message over HTTP to an agent in the Domino database. The agent parses the SOAP message, looking for the *namespace* and *method*. The namespace begins with *uri*, followed by the name of a LotusScript library in the Domino database; the method is the name of a function in that LotusScript library. The agent then invokes the function in the library and passes

the entire SOAP message as a parameter. In the figure below, this functionality is encapsulated in the component labeled *Business Logic*.

The function parses the SOAP message using some new LotusScript classes that invoke the DOM parser. The code looks for the book number in the SOAP message, then gets the appropriate document in the database by using the book number as a view key. From this document, the code can retrieve the title, price, authors, publisher, and number of available copies.

Finally, the code creates a response containing all this information within an XML document, and passes that document back to the agent. The agent then creates a SOAP message with all the book information and sends it back to the client. This complete architecture is illustrated in the figure below.

Install the book catalog application

Deploy the book catalog application to your Domino server by copying the file "bookcatalog.nsf" to the "\Domino\Data" directory. The Domino server needs to be version 6. The script library function uses the new DOM parser class in LotusScript to parse the incoming SOAP message.

There should be four documents in the database, describing four books, as shown in the figure below.

The WSDL file

A WSDL (Web Service Description Language) file describes a Web service. As the WSDL specification abstract puts it:

"WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate; however, the only bindings described in this document describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME."

There's a link to the complete specification in [Resources](#).

To make life a little easier, in the book catalog database, there is a form to generate the WSDL file for you. Open the book catalog database from the Lotus Notes client. Select the **WSDL** view and click **GenerateWSDL**. Fill in all the fields as illustrated in

the figure below, making any changes necessary for your environment. The output names are separated by commas. If you start to develop your own Web services, you can separate the input names by commas to create multiple values, too.

Below is the WSDL file that was created.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  name="ISBNSearch"
  targetNamespace="http://geo2003.lotus.com/bookcatalog.nsf/WSDL/ISBNSearch.wsdl"
  xmlns:tns="http://geo2003.lotus.com/bookcatalog.nsf/WSDL/ISBNSearch.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

  <message name="ISBNSearch">
    <part name="isbn" type="xsd:string" />
  </message>

  <message name="ISBNSearchResponse">
    <part name="title" type="xsd:string" />
    <part name="price" type="xsd:string" />
    <part name="authors" type="xsd:string" />
    <part name="publishers" type="xsd:string" />
    <part name="copies" type="xsd:string" />
  </message>

  <portType name="ISBNSearchPortType">
    <operation name="ISBNSearch">
      <input message="tns:ISBNSearch" />
      <output message="tns:ISBNSearchResponse" />
    </operation>
  </portType>

  <binding name="ISBNSearchBinding" type="tns:ISBNSearchPortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="ISBNSearch">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="encoded" namespace="uri:Domino"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
      <output>
        <soap:body use="encoded" namespace="uri:Domino"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  </binding>

  <service name="ISBNSearch">
    <documentation>ISBNSearch</documentation>
    <port name="ISBNSearchPort" binding="tns:ISBNSearchBinding">
      <soap:address
        location="http://geo2003.lotus.com/bookcatalog.nsf/WebService?OpenAgent" />
    </port>
  </service>

</definitions>
```

Test the Web service book catalog application in Domino

Use the following Java code to create a Java application to test the Web service book catalog on Domino.

```
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Vector;
import java.util.Iterator;

import org.apache.soap.Constants;
import org.apache.soap.Fault;
import org.apache.soap.SOAPException;
import org.apache.soap.rpc.Call;
import org.apache.soap.rpc.Parameter;
import org.apache.soap.rpc.Response;
import org.apache.soap.encoding.SOAPMappingRegistry;
import org.apache.soap.encoding.soapenc.BeanSerializer;
import org.apache.soap.util.xml.QName;
import org.apache.soap.util.xml.Deserializer;
import org.apache.soap.encoding.soapenc.StringDeserializer;

public class BookSearch {

    public static void main(String args[]) {
        try {
            String isbn = "0789728486";
            URL url = new URL("http://geo2003.lotus.com/bookcatalog.nsf/WebService?OpenAgent");
            Call call = new Call();
            call.setTargetObjectURI("uri:Domino");
            call.setMethodName("ISBNSearch");
            call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
            Vector params = new Vector();
            params.addElement(new Parameter("isbn", String.class,
                isbn, Constants.NS_URI_SOAP_ENC));
            call.setParams(params);
            Response response = call.invoke(url, "ISBNSearch");
            System.out.println("\nISBN: " + isbn);
            if (!response.generatedFault()) {
                Parameter result = response.getReturnValue();
                Vector allResults = response.getParams();
                Object value = result.getValue();
                Iterator i = allResults.iterator();
                while (i.hasNext()) {
                    Parameter p = (Parameter) i.next();
                    String name = p.getName().toUpperCase();
                    if ( "TITLE".equals(name) ) {
                        String title = p.getValue().toString();
                        System.out.println("Title: " + title);
                    }
                    if ( "PRICE".equals(name) ) {
                        String price = p.getValue().toString();
                        System.out.println("Price: " + price);
                    }
                    if ( "AUTHORS".equals(name) ) {
                        String authors = p.getValue().toString();
                        System.out.println("Authors: " + authors);
                    }
                    if ( "PUBLISHER".equals(name) ) {
                        String publisher = p.getValue().toString();
                        System.out.println("Publisher: " + publisher);
                    }
                    if ( "COPIES".equals(name) ) {
                        String copies = p.getValue().toString();
                        System.out.println("Copies: " + copies);
                    }
                }
            }
        }
    }
}
```

```
        else {
            Fault f = response.getFault();
            System.err.println("Fault = " + f.getFaultCode() + ", " + f.getFaultString());
        }
    }
    catch (SOAPException se) {
        System.err.println("SOAPException: " + se.getFaultCode() + ", " + se.getMessage());
    }
    catch (MalformedURLException mue) {
        System.err.println("MalformedURLException: " + mue.getMessage());
    }
}
}
```

Compile it and enter `java BookSearch` in a DOS window. This application is a bare-bones Web service client. Its output should look like the following figure.

Now that you've confirmed that your Domino Web service is up and running, we're ready to build a portlet that can act as client for it.

Section 4. Create a portlet

WebSphere Studio and Portlet Toolkit

Whether you're extending existing J2EE applications or starting from scratch, IBM WebSphere Studio provides you with a comprehensive, easy-to-use development environment that makes building Java applications, application adapters, and Web services a snap. We will be using WebSphere Studio Application Developer, along with the Portlet Toolkit, to develop our portlet. You should assume that all of the instructions and descriptions in the remainder of this tutorial take place in the context of WebSphere Studio Application Developer unless we specifically say otherwise.

The Portal Toolkit is a comprehensive toolkit for developing portlet applications. It is provided in the form of plug-ins to WebSphere Studio Site Developer Advanced or WebSphere Studio Application Developer. Version 4.1 of the Portal Toolkit provides the following:

- Portlet projects, with which you can create basic portlets, JSP portlets, servlet invoker portlets, XSL portlets, and MVC portlets.
- Portal server configuration, with which you can publish your portlet application onto your target WebSphere Portal server. Your portlet will appear on the debug page of your WebSphere Portal server.
- The capability to debug a portlet at the source level.

- Portlet application samples for enterprise application.

Installation requirements

Download the Portal Toolkit (see [Software requirements](#)) and install as directed in their accompanying documentation. Note the prerequisites.

If you have WebSphere Application Server installed on the same machine as WebSphere Studio Application Developer, you should add the WebSphere Application Server JVM to WebSphere Studio Application Developer's list of available JVMs. Follow these steps to add it after all the products are installed:

- In WebSphere Studio Application Developer, select **Window -> Preferences**.
- Expand **Java** by clicking the plus sign.
- Select **Installed JREs**.
- If you don't see the name **WAS JRE** selected, click **New...**
- Click **Browse** and choose the path where the JVM is installed.
- Once the JVM shows up under the installed JRE list, check the box next to **WAS JRE** to make it the default JRE.
- Click **OK** to save the changes.

Create a portlet project

Your first step is to create a portlet project. From within WebSphere Studio Application Developer, select **File -> New -> Portlet application project**. We will name the project `DominoBookCatalogPortal` and a new enterprise application project `DominoBookCatalogEAR`. Click **Next**.

Select a portlet type

To install a basic portlet infrastructure, select **Portlet Type -> None**. Click **Finish** when done.

On the console at the bottom half of the screen, you should see two errors: one indicates that you need to add a servlet reference to the `portlet.xml` file, and the other indicates that there is an incorrect reference to the portlet in the `portlet.xml` file. We will fix these problems later.

Create a WSDL file

In order to access our Web service in Domino, we need to create a Web service client in our portlet. We will use the wizard in WebSphere Studio Application Developer to do this. The Web service client wizard needs a WSDL file that describes the service's location, its input and output parameters, and other things about our Web service in Domino. If you created the WSDL file in the Domino database using the form we described earlier in this tutorial (see [The WSDL file](#)), you can simply point to it in the Web service client wizard. Enter the following URL for the WSDL file, changing the server name as appropriate:

```
http://geo2003.lotus.com/bookcatalog.nsf/WSDL/ISBNSearch.wsdl
```

If you want to store a WSDL file instead in our project, you'll first need to create a folder in your project. Browse to the Web Content folder and then select **File ->New ->Folder**. Name the new folder `wsdl`.

There are several ways to create the WSDL file in this folder. One of them is to select **File ->New ->Other ->XML ->XML File** and then enter the name as `ISBNSearch.wsdl`. Click **Finish** when done.

Now, just copy and paste the code below into the new file. My server name is `geo2003.lotus.com`, which is reflected in the code below; change this value as appropriate for your environment.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  name="ISBNSearch"
  targetNamespace="http://geo2003.lotus.com/bookcatalog.nsf/WSDL/ISBNSearch.wsdl"
  xmlns:tns="http://geo2003.lotus.com/bookcatalog.nsf/WSDL/ISBNSearch.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

  <message name="ISBNSearch">
    <part name="isbn" type="xsd:string" />
  </message>

  <message name="ISBNSearchResponse">
    <part name="isbn" type="xsd:string" />
    <part name="title" type="xsd:string" />
    <part name="price" type="xsd:string" />
    <part name="authors" type="xsd:string" />
    <part name="publisher" type="xsd:string" />
    <part name="copies" type="xsd:string" />
  </message>

  <portType name="ISBNSearchPortType">
    <operation name="ISBNSearch">
      <input message="tns:ISBNSearch" />
      <output message="tns:ISBNSearchResponse" />
    </operation>
  </portType>
</definitions>
```

```

</portType>

<binding name="ISBNSearchBinding" type="tns:ISBNSearchPortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="ISBNSearch">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="encoded" namespace="uri:Domino"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
    <output>
      <soap:body use="encoded" namespace="uri:Domino"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
</binding>

<service name="ISBNSearch">
  <documentation>ISBNSearch</documentation>
  <port name="ISBNSearchPort" binding="tns:ISBNSearchBinding">
    <soap:address
      location="http://geo2003.lotus.com/bookcatalog.nsf/WebService?OpenAgent" />
  </port>
</service>

</definitions>

```

Create a Web service client

We are now ready to create the Web service client. Select **File ->New ->Other ->Web services ->Web service client**. Then click **Next**. The client proxy type should be **Java proxy**. click **Next**. For the WSDL file name, click the **Browse** button and find the `ISBNSearch.wsdl` file we created earlier. Click **OK** on the **Browse Files** dialog box. Click **Next** on the **Web service WSDL file selection** dialog box. Click **Next** on the **Web service binding proxy generation** dialog box. This will create the Java proxy application for us. Click **Finish** on the **Web service test** dialog box; we don't need to launch the universal test client.

We should now have our Java proxy application. If you navigate to `Java Source\proxy.soap`, you should see `ISBNSearchProxy.java`. This is our Web service client application. We will be invoking the `ISBNSearch` method and passing in as a parameter an ISBN book number from our portlet. The Domino Web service will return a vector that contains the title, author, publisher, price, and number of copies of the book that corresponds to the ISBN.

First, however, we need to make a few changes to the Web service client code. In `ISBNSearchProxy.java`, change the method `ISBNSearch` to return a `Vector` instead of a `String`. Change the following line:

```
public synchronized java.lang.String ISBNSearch(java.lang.String isbn) throws Exception
```

To:

```
public synchronized java.util.Vector ISBNSearch(java.lang.String isbn) throws Exception
```

And now, one more change. Change the last two lines of the method from this:

```
Parameter refValue = resp.getReturnValue();  
return ((java.lang.String)refValue.getValue());
```

To this:

```
java.util.Vector book = resp.getParams();  
return (book);
```

A shortcut to format your code listing, lines everything up nice and neat, enter **Ctrl+Shift+F**. Then save the Java application.

Create a JavaBean component

We will develop a portlet that uses a JavaBean component called `DominoBookCatalogBean` to store and retrieve the values for our ISBN, title, price, authors, publisher, and number of copies.

To create a Java class called `DominoBookCatalogBean`, select the folder `Java Source`. Then select **File ->New ->Other ->Java ->Java Class** and click **Next**. Enter **portal** as the package name and **DominoBookCatalogBean** as the bean name, then click **Finish**.

JavaBean code

The JavaBean component is very simple. It has several set and get methods for all the information about a book. Copy and paste the following code into `DominoBookCatalogBean`. When done, press **Ctrl-S**.

```
package portal;  
  
import java.util.*;  
  
public class DominoBookCatalogBean {  
  
    private String userName = "";  
    private String submitURI = "";  
    private String isbn = "";  
    private String title = "";  
    private String price = "";  
    private String authors = "";  
    private String publisher = "";  
  
}
```

```
private String copies = "";

public void setUsername(String s) {
    userName = s;
}

public String getUsername() {
    return (userName);
}

public void setSubmitURI(String s) {
    submitURI = s;
}

public String getSubmitURI() {
    return (submitURI);
}

public void setisbn(String s) {
    isbn = s;
}

public String getisbn() {
    return (isbn);
}

public void setttitle(String s) {
    title = s;
}

public String gettitle() {
    return (title);
}

public void setprice(String s) {
    price = s;
}

public String getprice() {
    return (price);
}

public void setauthors(String s) {
    authors = s;
}

public String getauthors() {
    return (authors);
}

public void setpublisher(String s) {
    publisher = s;
}

public String getpublisher() {
    return (publisher);
}

public void setcopies(String s) {
    copies = s;
}

public String getcopies() {
    return (copies);
}
}
```

Import the portlet custom tag library

Before we create the JSPs, we need to import the portlet custom tag library. First, create a new folder under `WEB-INF` and name it `tld`. Then select this new folder and select **File ->Import -> File System**. Click **Next**, then click **Browse** and browse for the file `portlet.tld`. It should be in the `C:\Program Files\IBM\WebSphere Studio\wstools\eclipse\plugins\com.ibm.wps_4.1.0` directory. Click **Finish** when you're done.

Create two JSPs

The portlet API provides support for the use of JSP components to render portlet markup. Our portlet will create a JavaBean component containing data that the JSP will store and retrieve. We will create two JSPs: one for the request, which will be for the ISBN that the end user will enter, and the second for the response, which will be the results of the book search -- the information on the title, price, authors, publisher, and number of copies retrieved from Domino.

To create our two JSPs, follow these steps twice. Select the folder `Web Content`; right-click and select **New ->JSP File**. For **File name**, enter `View_Input`. The second name will be `View_Result`. Click **Next**.

Add a tag library

On the next dialog box, click the **Add Tag Library** button. This will bring up the **Select a tag library** dialog box.

Add a prefix

Select the check box next to the `portlet.tld` file. For the prefix, enter `portletAPI`. Select **OK** and then **Finish** when done.

Insert JSP bean

We need to insert a JSP bean. The `useBean` tag enables us to create a bean. Insert a blank line after the `<HEAD>` tag. Select **JSP => Insert Bean**. Enter `DominoBookCatalogBean` in the ID field and `portal.Domino.BookCatalogBean` in the Class field; In the Scope menu, select **Request**. Click **OK** when done.

To initialize the portlet API, enter the following line into both JSPs after the taglib statement. Type-ahead should work. Press **Ctrl+Spacebar** will display the type-ahead dialog box. `<portletAPI:init/>`.

Copy in the code for the input JSP

The input JSP is where the end user will enter an ISBN.

Copy and paste the following code for the `View_Input.jsp` file in the body section of the JSP. I placed all the ISBNs that are in the Domino book catalog database into a select list to make it easier for you to test. Instead of entering a book number every time, you can just select a book number from the list. Enter **Ctrl+S** to save the file.

```
<FORM method="POST" action="<%=DominoBookCatalogBean.getSubmitURI() %>"
      name='<%=portletResponse.encodeNamespace("myForm")%>'>
<TABLE>
<TR>
<TD ALIGN="left">Book Number:</TD>
<TD ALIGN="left">
<SELECT name="isbn">
<OPTION value=" "></OPTION>
<OPTION value="0789728486">0789728486</OPTION>
<OPTION value="0789726750">0789726750</OPTION>
<OPTION value="0672325020">0672325020</OPTION>
<OPTION value="078972796X">078972796X</OPTION>
</SELECT>
</TD>
</TR>
</TABLE>
<INPUT TYPE="submit" VALUE="Submit">
</FORM>
```

Copy in the code for the results JSP

The results JSP will display all the information about the book: the title, price, authors, publisher, and number of copies. It will also display an input field where the user can enter another ISBN.

Copy and paste the following code for the `View_Result.jsp` file in the body section of the JSP. Enter **Ctrl+S** to save the file.

```
<TABLE>
<TR>
<TD COLSPAN="1" ALIGN="LEFT">Book Number:</TD>
<TD ALIGN="left" VALIGN="TOP"><%=DominoBookCatalogBean.getisbn()%></TD>
</TR>
<TR>
<TD COLSPAN="1" ALIGN="LEFT">Title:</TD>
<TD ALIGN="left" VALIGN="TOP"><%=DominoBookCatalogBean.gettitle()%></TD>
</TR>
<TR>
```

```

<TD COLSPAN="1" ALIGN="LEFT">Price:</TD>
<TD ALIGN="left" VALIGN="TOP"><%=DominoBookCatalogBean.getprice()%></TD>
</TR>
<TR>
<TD COLSPAN="1" ALIGN="LEFT">Authors:</TD>
<TD ALIGN="left" VALIGN="TOP"><%=DominoBookCatalogBean.getauthors()%></TD>
</TR>
<TR>
<TD COLSPAN="1" ALIGN="LEFT">Publisher:</TD>
<TD ALIGN="left" VALIGN="TOP"><%=DominoBookCatalogBean.getpublisher()%></TD>
</TR>
<TR>
<TD COLSPAN="1" ALIGN="LEFT">Copies:</TD>
<TD ALIGN="left" VALIGN="TOP"><%=DominoBookCatalogBean.getcopies()%></TD>
</TR>
</TABLE>
<FORM method="POST" action="<%=DominoBookCatalogBean.getSubmitURI()%>"
      name='<%=portletResponse.encodeNamespace("myForm")%>'>
<TABLE>
<TR>
<TD COLSPAN="1" ALIGN="LEFT">Book Number:</TD>
<TD ALIGN="left">
<SELECT name="isbn">
<OPTION value=" "></OPTION>
<OPTION value="0789728486">0789728486</OPTION>
<OPTION value="0789726750">0789726750</OPTION>
<OPTION value="0672325020">0672325020</OPTION>
<OPTION value="078972796X">078972796X</OPTION>
</SELECT>
</TD>
</TR>
</TABLE>
<INPUT TYPE="submit" VALUE="Submit">
</FORM>

```

Create the portlet

Now it's time to create our portlet application. Portlets are special reusable Java servlets that appear on portal pages. Portlets provide access to many different applications. Our portlet will access Domino via Web services. It will take an ISBN as input and invoke the Web service proxy client.

Select the portal folder under the Java Source folder, right-click, and then select **New -> ->Class**. Click **Next**. For the name, enter DominoBookCatalogPortlet. Click **Finish** when you're done.

Copy in the code

Copy and paste in the following code for our portlet; name it DominoBookCatalogPortlet.

```

package portal;

import java.io.*;
import java.util.*;

```

```

import org.apache.soap.rpc.*;
import org.apache.jetspeed.portlet.*;
import org.apache.jetspeed.portlet.event.*;
import org.apache.jetspeed.portlets.*;
import proxy.soap.*;

public class DominoBookCatalogPortlet extends PortletAdapter implements ActionListener {
    protected static final String jsp1 = "View_Input.jsp";
    protected static final String jsp2 = "View_Result.jsp";
    int viewMode = 0;
    String isbn = "";
    String title = "";
    String price = "";
    String authors = "";
    String publisher = "";
    String copies = "";
    Vector book = new Vector();

    public DominoBookCatalogPortlet() {
        super();
    }

    public void init(PortletConfig portletConfig) throws UnavailableException {
        super.init( portletConfig );
    }

    public void service( PortletRequest request, PortletResponse response) throws
        PortletException, IOException {
        DominoBookCatalogBean bean = new DominoBookCatalogBean();
        PortletAction submitAction = new DefaultPortletAction("Submit");
        PortletURI submitURI = response.createURI();
        submitURI.addAction(submitAction);
        bean.setSubmitURI(submitURI.toString());
        PortletContext context = getPortletConfig().getContext();
        User user = request.getUser();
        String userName = user.getFullName();
        if( userName == null ) {
            userName = "Unknown WebSphere Portal User";
        }
        bean.setisbn(isbn);
        bean.settitle(title);
        bean.setprice(price);
        bean.setauthors(authors);
        bean.setpublisher(publisher);
        bean.setcopies(copies);
        request.setAttribute("DominoBookCatalogBean", bean);
        if ( viewMode == 0 ) {
            context.include(jsp1, request, response);
        }
        else {
            viewMode = 0;
            isbn = "";
            title = "";
            price = "";
            authors = "";
            publisher = "";
            copies = "";
            context.include(jsp2, request, response);
        }
    }

    public void actionPerformed(ActionEvent anEvent) throws PortletException {
        viewMode = 1;
        isbn = (String) anEvent.getRequest().getParameter("isbn");
        ISBNSearchProxy beanProxy = new ISBNSearchProxy();
        try {
            book = beanProxy.ISBNSearch(isbn);
            Iterator i = book.iterator();
            while (i.hasNext()) {

```

```
        Parameter p = (Parameter) i.next();
        String name = p.getName().toUpperCase();
        if ( "TITLE".equals(name) ) title = p.getValue().toString();
        if ( "PRICE".equals(name) ) price = p.getValue().toString();
        if ( "AUTHORS".equals(name) ) authors = p.getValue().toString();
        if ( "PUBLISHER".equals(name) ) publisher = p.getValue().toString();
        if ( "COPIES".equals(name) ) copies = p.getValue().toString();
    }
}
catch (Exception e) {
    e.printStackTrace();
}
}
```

Section 5. Update web.xml and portlet.xml

Update web.xml

Now, we'll fix the problem with the `portlet.xml` file that we encountered earlier (see [Select a portlet type](#)). In order to do that, we must add a servlet entry to the `web.xml` file. Double-click on `web.xml`, which will open it up for editing. Along the bottom, click on the **Servlets** tab and then click **Add**. Locate `DominoBookCatalogPortlet` by entering **DominoBookCatalogPortlet** for the choose a servlet name, select it in the matching servlets area, and click **OK**.

URL mappings

Under URL Mappings in the Servlets and JSPs window, click **Add**. Edit the new URL and change the value to `/DominoBookCatalogPortlet/*`. Press Ctrl-S to save our changes.

Update portlet.xml

Open `portlet.xml` for editing. Expand **Portlet Application** by clicking on the plus sign. Select **Portlet_1**. Next to **Servlet**, click **Browse**. In the Select Servlet dialog box, select **DominoBookCatalogPortlet - Not used** and click **OK** when done. Press Ctrl-S to save our changes. The two errors we saw previously should be gone now.

Rebuild our project

To incorporate all the changes, it's a good idea to rebuild our project. Select **DominoBookCatalogPortal** in the navigator window, right-click, and select **Rebuild Project**.

Section 6. Deploying portlets

Export as a WAR file

The first step to deploying our portlet to WebSphere Portal is to export our project as a WAR file. Select **DominoBookCatalogPortal** in the navigator window, right-click, and select **Export**. Select **WAR file** and then click **Next**.

For the location of the WAR file, enter

`C:\temp\DominoBookCatalogPortal.war` (or any other appropriate location).

For options, you can select **Export source files** and **Overwrite existing resources without warning** if you'd like.

Log on to portal as administrator

Log on to the portal as a portal administrator. Open Internet Explorer and enter the following address, changing the server name as appropriate for your environment:

```
http://geo2003.lotus.com/wps/portal
```

Click the key icon in the upper right-hand corner to log in. For testing purposes, select wpsadmin as the user ID; the password is also wpsadmin.

Click on the drop-down menu at the upper left-hand corner of the browser window -- it should have an entry for **Home** currently displayed -- and select **Portal Administration**. Select the **Portlets** and **Install Portlets** tabs.

Install the portlet

Click the **Browse** button to locate the `DominoBookCatalogPortal.war` file that we exported earlier (see [Export as a WAR file](#)). Click **Install** to install our portlet. If everything worked, you should see a message telling you that the portlets were successfully installed.

Verify that the portlet is active

To verify that the portlet is installed and active, select the second tab, **Manage Portlet Applications**, under the **Portlets** tab. Select `DominoBookCatalogPortal.war` in the upper box; you should see that the portlet application is active in the lower box.

Manage pages within a place

We are now going to customize the portal. WebSphere Portal introduces the concept of *pages* that exist within *places*.

- A *place* is a collection of portal pages. The portal administrator creates places, determines which portal pages are in each place, and gives the appropriate users authority to access the place and pages.
- A portal *page* displays content. A page can contain one or more portlets, which are the applications that display the page content. For example, a World Market page might contain two portlets that displays stock tickers for popular stock exchanges and a third portlet that displays the current exchange rates for world currencies. To view a page in the portal, you select its place from the place selector and then click the page within the place.

You can put a portlet on any page in any place in the portal where it makes sense. Let's create a new page for our portlet in the **Home** place. Click on the drop-down menu at the upper left-hand corner of the browser window and select **Work with Pages**. Select the second tab, **Manage Places and Pages**. Under **Places you can manage**, select **Home** and click **Manage pages** on the right-hand side.

Create a page

Click **Create page** on the right-hand side next to **Pages you can manage**. Then click **Create new** to create a new blank page.

Enter portlet title

For **Administrative name and default locale title**, enter `Domino Book Catalog`. For **Layout**, select a single column, and for **Supported markups**, select **HTML**. Click **OK** and then click **Done** when you're finished.

Edit layout and content

Click the first tab, **Edit Layout and Content**. The place should be **Home**; select our new page, **Domino Book catalog**.

Get the portlet

Click **Get portlets**. Select **Show all portlets** and click **Go**. Click the plus sign to the left of **DominoBookCatalogPortal portlet** to add the portlet to the list. You should see the portlet in the portlet list at the top and to the right. Click **OK** when done.

Activate the portlet

You should now see **DominoBookCatalogPortal portlet** in the portlet list. Select it. Click the icon to **Add selected portlets from the portlet list above to this container**. You should now see the portlet in the container. Click **Activate** when finished.

View the portlet

Select **Home** from the drop-down menu at the upper left hand corner. Select the page **Domino Book Catalog**.

If everything has gone properly, you should see our `View_Input.jsp` displayed.

View the results

Drop down the book number input box and select a book number, then click **Submit**.

You should now see the results of our book number request, if everything has worked properly.

Our portlet invoked the ISBNSearch proxy Web services client, which submitted a SOAP message to Domino. The Web service agent parsed the message for the book number, looked up the document in the database, retrieved the title, price, authors, publisher, and copies, created a SOAP message containing this information, and sent it back to our portlet. The portlet parsed the results and stored the values in a JavaBean component, from which our `View_Result.jsp` retrieved the values and then displayed them in the portal.

Section 7. Troubleshooting

Examining the WAR file

If the portlet didn't work for whatever reason, download the DominoBookCatalogPortal.war file (it's in the zip archive that accompanies this tutorial; see [Software requirements](#)). Then, log on as the portal administrator using wpsadmin/wpsadmin or your administrator's user ID. Select **Portal Administration** from the drop-down menu and click the **Manage Portlet Applications** tab. Select **DominoBookCatalogPortal.war** under Web modules. You can either **Update** or **Uninstall** the portlet. Follow the prompts to update or reinstall the portlet.

You can also import this WAR file into WebSphere Studio Application Developer and take a look at all the files.

Section 8. Wrap up

Summary

Contextual collaboration is all about providing collaborative services -- messaging/inbox, calendar/scheduling, workflow, online awareness, chat, e-meetings, document library, expertise, or online courses -- within business applications where the end user needs it. But, most importantly, it's about providing your own collaborative services based on the Lotus Domino applications you've developed in your business applications to portals and other business applications.

This tutorial has provided you with information on how to create a portlet that accesses Domino using Web services. You can use this information now to add any Web service from an inventory, procurement, or customer relationship management application to a portal. These systems can run on Domino, .NET, or any other platform that supports Web services.

Resources

Learn

- [Building Web services using Lotus Domino 6](#)
- [Turn your Lotus apps into Web services](#)
- [Web services -- the Web's next revolution](#)
- [Develop portlets that use Web services to obtain data from remote systems](#)
- [Implementing Web services with the WSTK 3.0.1](#)
- [Jetspeed, Part 2: Advanced portlet technology](#)
- [Developing portlets that use JavaBeans and JSP components](#)
- [WebSphere Portal V4 programming, Part 1: Portlet application programming](#)
- [WebSphere Portal V4 programming, Part 2: Portlet application programming](#)
- You should also check out the specification for the [Web Services Description Language \(WSDL\)](#).
- Here are a few IBM redbooks on Web services, portlets, and Domino development:
 - [Self-Study Guide: WebSphere Studio Application Developer and Web Services](#)
 - [Web Services Wizardry with WebSphere Studio Application Developer](#)
 - [Domino Designer 6: A Developer's Handbook](#)
 - [Domino and WebSphere Together Second Edition](#)
 - [IBM WebSphere Portal Developers Handbook](#)
- Check out this great book on Web programming: [Domino 5 Web Programming with XML, Java, and JavaScript](#), Randall A. Tamura (QUE Publishing, 2000).
- "From ICE age to contextual collaboration," Robert Mahowald (*CIO*, June 2001).
- Stay current with [developerWorks technical events and Webcasts](#).

Get products and technologies

- Build your next development project with [IBM trial software](#), available for download directly from developerWorks.

Discuss

- [Participate in the discussion forum for this content.](#)

About the author

George Brichacek



George Brichacek is a technology advocate with IBM Lotus Software and the IBM Software Group, focusing on application development and enterprise integration. He has presented and developed several memorable demos for the Lotus Masters Broadcast, as well as for customers and for internal field enablement. He works with Lotus product management to bring the latest and greatest technologies in Lotus application development and enterprise integration to you. During his spare time, he supports a Lotus Notes utility, called CalPrint, that he developed. George resides in the Chicago area. You can reach him at george_brichacek@us.ibm.com.