Creating a complete Web service

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

Section 1. Introduction

Tutorial Purpose

The IBM Web services Toolkit *IBM Web Services Toolkit* (WSTK) on alphaWorks provides a practical introduction to hosting Web services. It also includes a companion, the *Aggregation demo*, that consists of a number of Web services, their associated visual components, and a simple portal in which to run them.

Be forewarned, that due to the alpha stage of the tools, integration is not complete. This means you may have to get your hands dirty at times with manual intervention that will not be required in real products.

Prerequisites

You should have an understanding of the technologies underlying Web services and e-business in general, including XML, Java, Web Services Description Language (WSDL), Simple Object Access Protocol (SOAP), and Universal Description Discovery and Integration (UDDI).

The tutorial leverages two technologies available on IBM's alphaWorks: the Web Services Toolkit (WSTK) and the Aggregation demo to set up a local environment for hosting Web services; and the XML and Web Services Development Environment (WSDE), to turn the existing Web-based application into a Web service based application (a Web service and its associated visual component). WSDE also deploys and publishes the Web service.
In order to make use of the tutorial, you'll need:

* Windows NT 4.0 (with Service Pack 6a) or Windows 2000 (with Service Pack 1)
* JDK 1.2.2 or JDK 1.3
* Internet Explorer 5.0 (or higher) or Netscape Communicator 4.5 (or higher)
* the WSTK and the Aggregation demo, downloaded and installed
* a local version of a UDDI-compliant Web services registry (one is available in the WSTK)
* the personal version of IBM's DB2 database
* the WSDE
* and finally, a Web-based application that you'd like to integrate into the Aggregation demo and turn into a Web service based application. You can use an existing application, create something from scratch, or simply use the example presented in the tutorial.

About the author

Greg Flurry is a member of the Software Group Emerging Technologies Area. He and other members of the area are working to improve the applicability of Web services and related technologies in e-business environments. You can reach Greg at *flurry@us.ibm.com* .

Section 2. Installing the Web Services Tools

WSDE

You can download WSDE from IBM alphaWorks at *www.alphaworks.ibm.com/tech/wsde* . Be sure you get the version that was updated on 12/20/2000. If you have trouble with this, the WSDE and installation are discussed in the Web services tutorial titled Web services: The Web's next revolution by Doug Tidwell. (Note that at this time, WSDE works only on a Windows operating system.)

You can download the WSDE zip file (`xml-wsDE.zip`, which is almost 90 MB, so be patient), or you can download the file or get it in pieces (`xml-wsDE.zip.1-9`) and follow the instructions on the site for reassembly. When you unzip the file, it creates an itp directory relative to the base directory you unzip to -- in the tutorial this is referred to as WSTK_HOME. Check your installation by "exploring" to the `itp` folder and double-clicking on `ide.exe`. If you wish, follow the instructions in Doug's tutorial to actually use the WSDE. You will need to shut down the WSDE to install some plugins after installing the WSTK.

Web services: The Web's next revolution

WSTK - base

You can download the WSTK and associated components at *www.alphaworks.ibm.com/tech/Webservicestoolkit* . You must use WSTK version 2.2.1, updated 03/21/2001, for this tutorial. You cannot successfully complete the tutorial with an older version of the WSTK.

You will need to download only the following zip files:

* **wstk221.zip** - supplies the base WSTK; you must install (unzip) this file. Once installed, you can browse the WSTK's installation guide and background material for additional information.
* **wstkuddi221.zip** - supplies some of the support necessary to implement a local UDDI registry; you will also need an installation of the IBM DB2 database. I'll discuss this at greater length below.
* **browser-plugin221.zip** - supplies a plug-in Web Services Browser for the WSDE. This must be extracted in the `itp` directory where the WSDE was installed.
* **AggregationDemo22.zip** - (not a typo, it really is AggregationDemo22.zip) supplies the Aggregation demo; I'll discuss this in greater detail below.

You must install the WSTK on the same machine as the WSDE. This is necessary only for the current alpha version of the WSDE. The rest of the tutorial will assume that both WSDE and WSTK are installed on a single machine.

I recommend installing `wstk221.zip` in the root directory; we'll refer to the path up through the `wstk-2.2` directory that gets created as WSTK_HOME. Once you have installed `wstk221.zip`, the WSTK runs with three different application servers:

* IBM WebSphere Application Server, Version 3.5.2
* Embedded WebSphere, Version 3.5

* Apache Tomcat 3.2.1

If you are already using WebSphere 3.5.2 or Tomcat 3.2.1, I suggest you complete the installation and configuration by following the appropriate instructions for your current application server. If you currently run neither of these application servers, I suggest you use Embedded WebSphere. It is included in the `wstk221.zip` file and minimizes additional configuration. I used that server while creating this tutorial.

WSTK - local UDDI

The WSTK requires a UDDI-compliant Web services registry. The WSTK offers a choice; you can use the IBM UDDI Test registry or you can set up a local UDDI registry on your machine. The tutorial assumes you have set up a local UDDI registry.

Follow the WSTK installation and configuration instructions related to the local UDDI. You must download `wstkuddi221.zip` and unzip the file into the same directory you unzipped `wstk221.zip`. Running a local UDDI requires you to install a DB2 database; IBM makes a personal version of DB2 available to you for free, but you do have to download, install and configure it. See the WSTK installation instructions for the local UDDI for details.

WSTK - testing

I recommend that, in order to check your installation, you run some of the Web services demos provided with the base WSTK after installing the local UDDI. Follow the instructions to deploy, publish, run, unpublish, and undeploy the stock quote service or the test Web service, or both. If these demos don't work, you need to check your configuration.

One item that may cause problems is the database access. If this happens, make sure that the user **wstkAdmin** with the password **wstkAdmin** has access to the WSTKDATA database. You can do this by installing DB2 as the user **wstkAdmin**. If you have installed DB2 as another user, use follow these steps to give **wstkAdmin** access:

1. Open the DB2 Control Center.
2. Expand the tree on the left until you see the database **WSTKDATA**.
3. Right click on database **WSTKDATA** and select **Connect**.
4. In the **Connect** dialog box, enter "wstkAdmin" in the **User ID** and the **Password** fields; click the **OK** button.

WSTK - Aggregation demo

Once you've successfully run the stock quote web service demo, install the Aggregation demo. Unzip `AggregationDemo22.zip` into the same directory where you unzipped `wstk221.zip`. Then follow the instructions for configuring the WSTK environment for the Aggregation demo.

Once you've finished installation and configuration of the Aggregation demo, run it following the detailed instructions included with the WSTK documentation. Here is the brief version (assuming you are running on a single system):

1. Go to `WSTK_HOME/demos/aggregation.` Run the following command: `aggregation.bat both`
2. Point a browser at the local Aggregation demo (e.g., http://demohost:8080/aggr/editor/index.html). You'll see a login page. The Aggregation Web services can run without a network connection using canned data, or they can run "live" with a network connection. It is important to deselect **Network Connected** if you are not connected. Type in any name in the **Username** field and then click the **login** button. You will see a mostly blank page with a menu at the top, as shown in Figure 1.
3. Select the **Edit Services** button. You will see a page with a palette of Web services on the left and a layout area on the right, as shown in *Figure 2* .
4. Select a Web service from the palette. An outline of the visual component for that Web service will appear in the layout area on the right. Click the **Save Layout/Return** button.

You can now interact with the Web service via its visual component. You can add other applications by clicking the **Edit Services** button and repeating steps 3 and 4 above. *Figure 3* shows the NSFNewsService application running with a SearchService application.

WSTK - browser plugin

This is the final part of the tutorial tools setup; you'll need to install the Web Services Browser plugin from the WSTK into the WSDE. Unzip `browser-plugin221.zip` into the `WSDE_HOME/itp` folder created when you installed the WSDE. This allows you to use the Web services Browser tool to browse the businesses and Web services in a UDDI-compliant registry. You can also use the tool to publish your Web service. For additional installation and configuration information, before you unzip the file, view `Readme.html` which is included in the zip file. Make sure that the WSDE version of `wstk.properties` matches the one in your WSTK environment, including the application server (`wsdl.service.port.name` property), and the credentials for the local UDDI (`uddi.userid.demohost8080` and `uddi.cred.demohost8080` properties). The best thing to do is simply copy `WSTK_HOME/lib/wstk.properties` to `WSDE_HOME/itp/wstk.properties`. When you are finished, start the WSDE (as described above) and you should see this icon

in the menu.

Section 3. The sample Web-based application

Required Characteristics

When you ran the Aggregation demo, you interacted with a visual component that in turn interacted with a Web service in order to provide a service, such as stock quotes or news. So it should come as no surprise that in order to integrate your Web service into the Aggregation demo, your Web service must also have an associated visual component.

"Web service based application" describes the combination of a Web service and its associated visual component. A Web service based application is closely analogous to a "Web-based application," which has a visual component for human interaction and typically uses some sort of "service" to provide function. Thus, many existing Web-based applications provide a starting point for creating a Web service based application. Of course, starting with a reasonably simple Web-based application is a good idea.

This Web-based application in this tutorial uses a JSP (Java Server Page) as the visual component and a JavaBean to provide the actual service. This application is then used to create a Web service based application, which is integrated with the Aggregation demo.

Although the service you create here might use a servlet instead of a JSP, for simplicity's sake we'll use JSP when referring to all visual components of a Web-based application or Web service based application.
To minimize the configuration necessary for setting up the application server Webgroups (contexts, in Apache) and class path, we can make one slight concession with respect to the location of the "existing" Web application. We'll start with the service JavaBean located in a path already in the class path of the application server, and with the JSP already in an existing Webgroup of the application server. Since we are going to integrate with the Aggregation demo,
`WSTK_HOME/demos/aggregation/Webapps/aggregation/WEB-INF/classes/aggregation`;
is a good choice for the location of the JSP is in
`WSTK_HOME/demos/aggregation/Webapps/aggregation/jsps.` The code discussed in the rest of the tutorial reflects these choices. If you are starting with an existing service, you need to move your service JavaBean class file(s) to the location suggested, and either make your service JavaBean part of the aggregation package, or modify the class path of the application server. You must move your JSP to the suggested location because of the implementation of the Aggregation demo.

The example service integrated into the test area provides weather forecasts. As with many of the Web services in the Aggregation demo, the Web service for the weather forecasts actually wraps a Web-based weather forecasting service (in this case the National Weather Service) that provides HTML. The weather forecasting application consists of a JavaBean (the wrapper) that processes the HTML from the National Weather Service to produce XML documents. The JSP then translates the XML documents into HTML, with the assistance of a helper JavaBean, to produce the desired visual result. The JSP also provides HTML forms to get the desired state and city.

The WeatherForecast JavaBean

The `WeatherForecast` JavaBean, shown in *Listing 1* , has three public methods: `getStates()`, `getCities()` and `getForecast()`. All of these methods return an XML

document. None of the documents have a DTD, and the elements in the document are contrived for this example. (A weather related standard could have been used instead, if one were to be found.)

The `getStates()` method simply returns a list of state abbreviations. As you can tell from *Listing 1* , the state information is canned, as it does not change often. For the sake of brevity, the listing does not include all of the states.

The `getCities()` method, once given a state, actually retrieves a list of cities from the NWS for which it provides forecasts in that state. And the method, as we've seen, gets HTML from the NWS. The method parses the HTML to produce the XML document returned to the caller. There are a couple of interesting aspects to this method: first, notice the check for being "off-line;" this allows operation with no network connection (with canned data). Second, the method caches the city information returned from the NWS. Let's look at this in a bit more detail.

The `getForecast()`method, given a state and city, retrieves the forecast from the NWS. It also parses the HTML to produce an XML document. The list of cities returned from the NWS contains the city name and the URL of the forecast for each city. I chose to return from the `getCities()` method only the names and not the URLs. This means that when a caller requests the forecast for a city, the request can contain only the names of the state and the city. It also means, however, that the `getForecast()` method, without caching, would have to make another request on the NWS to obtain the correct URL for the forecast. With caching, the `getForecast()` method can get the appropriate URL out of the cache, and send to the NWS only the one request needed to get the forecast.

The WeatherForecastHelper JavaBean

The `WeatherForecastHelper` JavaBean, shown in *Listing 2* , provides convenience methods that wrap the methods in the `WeatherForecast` service JavaBean. `WeatherForecastHelper` has two methods, `getStates()`and `getCities()`, both of which provide additional caching. These methods cache the information returned from the `WeatherForecast` JavaBean in order to reduce the calls made on it by the helper. The `WeatherForecastHelper` also contains the additional methods of (`formatStates()`, `formatCities()`, and `formatForecast()`) which help the JSP format its response.

The forecast JSP

The forecast JSP, shown in *Listing 3* , provides a user interface for the `WeatherService` JavaBean. The JSP follows a normal JSP structure. It creates an instance of a `WeatherForecast` class and an instance of a `WeatherForecastHelper` class and puts the latter in the session context.

It is important to use the session scope here to prevent threading problems with the city caches used by the `WeatherForecast` and `WeatherForecastHelper` objects. The JSP controls its actions by looking at the request parameters and session state. So it gets state information when appropriate, city information when appropriate, and forecast information when possible (when both city and state are valid).

Running the weather forecast service

To run the Web-based application, you must place the Java class files for your service in the class path of the application server. You can add to the class path if you need to. You must also place the JSP in the "aggr" Webgroup (or context) of the application server. These actions are application server dependent, and you should consult the documentation for your application server for further information.

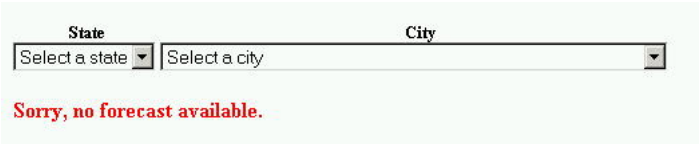
As may also be the case for your Web-based application, there are other types of files which are part of the weather forecast application. All these files are stored in a single directory with the class files (the directory is named `forecast` for the example). The properties file, `forecast.properties,` contains a single property, OFFLINE, that indicates whether the application is network connected (OFFLINE=F) or not (OFFLINE=T). The HTML files hold canned data for offline operation.

Start your application server. Then start Embedded WebSphere using one of two methods:

1. Go to `WSTK_HOME/demos/aggregation` and run `aggregationdemo.bat server`
2. Run `WSTK_HOME/bin/wstkenv` (from any directory) and then run `WSTK_HOME/bin/Websphere.bat run`

Now run your JSP. For the example in the tutorial, point your browser at the URL http://demohost:8080/aggr/jsps/forecast.jsp. You will see a page, Figure 4, on which you are able to select a state.

Figure 4

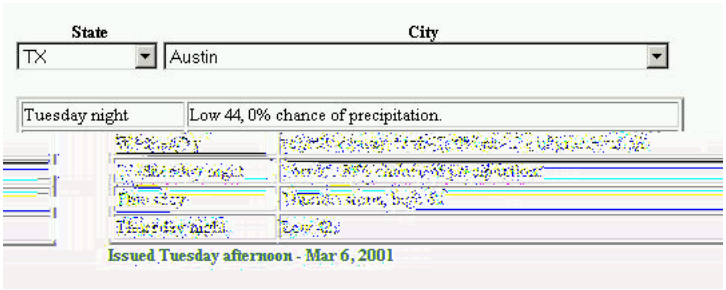


If you select a state, (TX in the example) you will see a page (Figure 5) that allows a selection from a list of cities in the selected state.

Figure 5

Once you select a city, you will see something similar to Figure 6, namely a forecast for the selected city and state.

Figure 6



Section 4. Creating the Web service

Overview

To perform this step, you must have the WSDE installed (see *Section 2: WSDE* for details). This section follows the outline in the WSDE Help under **Scenarios** > **The Stock Quote scenario**. You can consult the WSDE Help for additional instructions if necessary.

The following actions will create the Web service:

1. Create a Web project.
2. Import the JavaBean that provides the service.
3. Create the WSDL file that describes the Web service.
4. Deploy the Web service into the SOAP server; this lets the SOAP server actually run the service.
5. Create a proxy that a client uses to interact with the Web service.

Creating the Web project

Start the WSDE, go to the desktop and switch to the default perspective. Then create a new Web Project for your service. Here we will use weather forecast service for an example.

1. Click **File** > **New** > **Web Project**
2. Type a name in the **Solution name** text field (I used `WeatherForecastSolution`).
3. Type a name in the **Project name** text field (`WeatherForecast`).
4. Click **Next**. The **Web Settings** page of the Web Project wizard provides defaults for the Web application settings.
5. Click **Next** again to get to the **Java Settings** page.
6. Click **Add External Jar** and browse to locate and select the file `WSDE_HOME\itp\plugins\b2bxmlrt\xerces.jar`, where `WSDE_HOME` is your installation directory.
7. Click **Add External Jar** and browse to locate and select the file `WSDE_HOME\itp\plugins\org.apache.soap\soap.jar`. You can use the Java Settings page to add more information to the class path if your service requires it.
8. Click **Finish**. Your solution (`WeatherForecastSolution`) will appear in the **Navigator**. Expand it to see the project (`WeatherForecast`). Expand the project. Notice the two folders `Web` and `servlets`. The code for your service will be stored in the `servlets` folder.

Importing the JavaBean

Now you need to import the JavaBean that implements your service. To import the JavaBean:

1. In the **Navigator**, under the project, select the `servlets` folder.
2. Create the package structure of your Web service under the `servlets` folder folder by right-clicking on the folder and selecting **New** > **Folder**. The example requires an **aggregation** folder and then a **forecast** folder within **aggregation**, since the example code is in the `aggregation.forecast` package.
3. Select your folder (**forecast** for the example). Click **File** > **Import** to open the Import wizard.
4. Click **File system** to import the resources from the local file system. Click **Next**.
5. To enter the directory, click **Browse** to locate and select the directory where your JavaBean is located. (You will also need to select **Files of type** and select only **Java** files). Then click the **OK** button to close the type selection menu. Click the **Finish** button to import the file and close the wizard. The WSDE will attempt to compile your service JavaBean.

Note: If you have other Java files in the folder with your service JavaBean, they will also get imported, and may cause compile errors. Unless you need them to make your service JavaBean compile, delete them. Compilation error messages will show up in the All Tasks window. If your service JavaBean (or some other file) does not compile successfully, ensure that the project's build class path is complete. To add to class path, select the project (`WeatherForecast`) and right click and select **Properties** and in the dialog, select **Java Build Path** and add to the class path.

Creating the WSDL file

Once the JavaBean is imported, you can create the WSDL that describes the Web service using the Web Service Definition Wizard.

1. Select the **servlets** folder in the WSDE **Navigator**.
2. Click **File** > **New** > **WSDL Web service** to start the Web Service Definition Wizard.
3. Make sure the solution name and folder name are correct.
4. Ensure the **Create a Web service (WSDL) from an existing JavaBean** radio button is selected. Click **Next**.
5. In the **JavaBean Selection** page of the wizard click on your class file `aggregation/forecast/WeatherForecast.class` for our example) from the **Look in** list to specify the location of the class file, then click **Next**.
6. The **Method Selection** page of the wizard shows a summary of methods in your JavaBean. The wizard will select all public methods by default. The `forecast.WeatherForecast` JavaBean has its three public methods selected by default. Select the methods you wish to be available to users of your Web service and click **Next**.
7. You can specify the Web server environment to which your Web service will be deployed, and endpoint information about the Web service in the **Web Service Definition** page of the wizard. (Note that for this alpha release of the WSDE, deployment is supported only to servers running on the local host.) The wizard automatically fills in the fields in this page. For the **Deployment environment**, you should select either the **WebSphere** or **Apache** radio buttons. If you are running

Embedded WebSphere, select **WebSphere** and enter **8080** into the **Port** field.

8. The Scope is an important aspect of deployment because it determines, at least in the SOAP environment, whether your Web service will get instantiated for each request (request scope), each session (session scope), or once for all sessions and requests (application scope). The `aggregation.forecast.WeatherForecast` JavaBean is designed for session scope, so you must select **session**.
9. You can accept the rest of the defaults for testing in the WSTK environment. Click the **Next** button.

The WSDL file (`WeatherForecast.wsdl`) is created in the `servlets` folder. You can examine the WSDL file by double-clicking it. It will look something like the WSDL file for the weather forecast Web service in *Listing 4* .

Deploying the Web service

You should now be looking at the **JavaBean Web Service Deployment** page of the Web Service Definition Wizard. This page shows the properties for the Web server environment you selected on the previous page. You can deploy the Web service once you have reviewed the properties for the Web server. In particular, if you are using Embedded WebSphere, make sure the **Web Server Install Path** field reads `WSTK_HOME\WebSphere\AppServer`. First make sure your application server is running, then click the **Deploy** button. After a few seconds you should see a popup indicating success. Click the **OK** button.

Note: If you are using Embedded WebSphere, you will probably see the error panel shown in Figure 7.

Figure 7

Don't worry! Deployment actually succeeded, but the alpha version of WSDE does not expect to deploy to Embedded WebSphere, and it mistakenly thinks deployment failed. If you see the error panel, just click the **OK** button.

After the Web service is deployed, open the XML-SOAP administration console (`http://demohost:8080/soap/admin/`). You should see your Web service, which should be named `urn:<service-bean-name>`. For example, the weather forecast service is listed as `urn:WeatherForecast`. Note that you can also use the XML-SOAP administration console to deploy and undeploy Web services.

For more information on the deployment environments in WSDE, see the deploying information under the **Web services Reference** section of **Help**.

Generating the client proxy

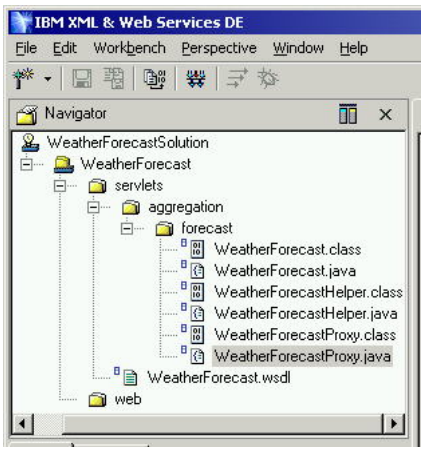
You should still be looking at the **JavaBean Web Service Deployment** page of the Web Service Definition Wizard.

Create the client proxy as follows:

1. Click the **Next** button to enter the information needed for the creation of the client-side proxy. The **Java Client Proxy Generation** panel allows you to specify the container (path) for the generated Java client proxy. It defaults to `project-name>/servlets/services` . I suggest you put the proxy in the same container as the Web service so that the package name generated for the proxy is the same as that of the service JavaBean.This minimizes the application server class path configuration. The forecast Web service uses `/WeatherServiceSolution/WeatherService/servlets/aggregation/forecast`. The name of the Java client proxy defaults to `<service-bean-name>Proxy.java`. For the example, the name is `WeatherForecastProxy.java`
2. Click the **Next** button to generate the proxy.
3. Click the **Finish** button to close the wizard, since there are no plans to run Web service in the WSDE environment. See the WSDE **Help** section for information on testing the Web service in the WSDE environment.

After creating the WSDL, deploying your web service and creating the client proxy, your WSDE **Navigator** panel should appear similar to Figure 8.

Figure 8



Don't exit the WSDE. You'll need to use it again in the next stage of the tutorial.

Section 5. Testing the Web service

Overview

Now it's time to test the Web service you've created. To test the Web service in a local WSTK environment, you need to

1. Put the class file for the client proxy created by the WSDE into the WSTK application server class path.
2. Modify the JSP and any other code that uses the your service JavaBean (the WeatherForecast JavaBean in the example) to use client proxy instead.

There are some important assumptions here, however. Since you've already tested your Web service based application in a WSTK enabled application server, that means the other files necessary to run your Web service (for example, the `aggregation.forecast.WeatherForecast` and `aggregation.forecast.WeatherForecastHelper` classes and the associated files) are already available to the application server. So your JSP is already located in a Webgroup of the application server. If you tested your Web-based application in a different application server environment, you will need to satisfy these assumptions.

Put the proxy class file in the class path

The class for the client proxy must be in the class path so that the application server can load the proxy when used by the JSP. There are two ways to put the class file in the application server class path. You can leave the code in the location where it was created by the WSDE and add that directory to the class path, or you can export the code (at least the `class` file) from the WSDE into a directory that is already in the WSTK application server class path. For the example we chose to have WSDE create the proxy in the same directory (package) as the Web service, so we will export the proxy to the directory where the Web service JavaBean resides, `WSTK_HOME/demos/aggregation/Webapps/aggregation/WEB-INF/classes/aggregation.`

Perhaps the easiest way to modify the WSTK application server class path is to add the `WSTK_HOME/bin/wstkenv.bat file`. This batch file gets executed to set up the proper class path for all of the application servers supported by the WSTK. You need to add a line of the form: `set WSTK_CP=%WSTK_CP%;<path-to-class-file>.` (Note: Due to command line length limitations in Windows, it may not be possible to add to the class path, so I recommend putting the proxy in a directory already in the class path.)

In general, you can find the files (Java and class) for the client proxy in `WSDE_HOME/itp/workbench/<solution-name>/<project-name>/servlets`, where you will also find whatever additional directories are introduced by your package structure. For the weather forecast files in this tutorial, the path is `WSDE_HOME/itp/workbench/WeatherForecastSolution/WeatherForecast/servlets/aggrega`

You can export the proxy code from the WSDE as follows:

1. In the **Navigator** panel of the WSDE, select the folder containing the proxy class file. You can find the folder by following the steps we just outlined; in our example it is **forecast**.

2. Select in the menu **File** > **Export**, which opens the Export Wizard.
3. In the **Select** page, select **File system** and then click the **Next** button.
4. In the Directory field on the File system page, select the directory to which the proxy class file must be exported. Select the **Resources of type** radio button and click the **Edit** button.
5. Select at least the **class** type, and click the **OK** button.
6. Select the **Details** button. You will see the **Resource Selection** panel that contains the directory you requested to export. It will look something like the dialog box shown in Figure 9.
7. Click on the "+" to expand the folder, and expand other folders as needed to find the folder containing the proxy. Unselect the top-level folder to ensure that you don't export anything but what you select in the next step.
8. Expand the folder containing the proxy. You will see the files for the client proxy and any other files in the folder displayed. Check the proxy class file. You will see something like the dialog box shown in Figure 10.
9. Now click the **OK** button to return to the **File system** page. Under the **Options** field, unselect **Create directory structure** and then click the **Finish** button. If you check the destination you selected for export, you will find the client proxy files in that destination.

Modify the JSP and other code

The JSP must now use the client proxy to call the Web service instead of directly calling the JavaBean that implements the service. Similarly, any other code that previously referenced the service JavaBean must be modified to reference the client proxy instead. In the weather forecast example, both the `forecast.jsp` and `WeatherForecastHelper.java` must be modified to reference the `WeatherForecastProxy` instead of `WeatherForecast`. It is important to note that the service JavaBean itself *does not* have to be modified.

Since I put the client proxy for the example in the same package as the original service JavaBean, the modifications to both files are quite simple. Using a text editor, in `WeatherForecastHelper.java`, simply change `WeatherForecast` to `WeatherForecastProxy` and then recompile. The resulting code is shown in the following partial listing of the modified `WeatherForecastHelper.java`.

```
public class WeatherForecastHelper {
     protected WeatherForecastProxy forecaster = null;
     public void setForecaster(WeatherForecastProxy forecaster){
         this.forecaster = forecaster;
     }
     public WeatherForecastProxy getForecaster(){
         return forecaster;
     }
     ...
 }
```

The JSP modifications are similar. Use a text editor to change the references to `WeatherForecast` to `WeatherForecastProxy`. The following excerpt from the modified `forecast.jsp` shows the required changes:

```
WeatherForecastProxy forecaster = forecastHelper.getForecaster();
 if (forecaster == null) {
     forecaster = new WeatherForecastProxy();
     forecastHelper.setForecaster(forecaster);
```

```
}
```

You can now attempt to run the your Web service based application by pointing your browser at the URL for its visual component. Your results should be identical to the ones you get when you run the original Web-based application. If you view `http://demohost:8080/aggr/jsps/forecast.jsp` for the weather forecast application, the results should look identical to those shown above. (Note: Remember that you must either copy or move your JSP into a specific directory so that the Aggregation demo can find it. The directory is `WSTK_HOME/demos/aggregation/Webapps/aggregation/jsps`.)

Section 6. Integrating into the Aggregation demo

Overview

In this section you will complete the integration of your Web services based application into the Aggregation demo. To complete this section, you must have installed the Aggregation demo along with the WSTK, chosen to use the local UDDI with the WSTK (see Section 2: WSTK - local UDDI), successfully run the Aggregation demo (see *Section 2: WSTK - Aggregation demo*), and created and tested your own Web service based application (see *Section 3: Running the weather forecast service*) in the WSTK environment.

The following actions are required to integrate with the local Aggregation demo:

1. Modify the WSDL document describing your Web service so that the Aggregation demo can identify your Web service as having a related visual component (that is, as part of a Web service based application).
2. Actually publish information about your Web service in the local UDDI so the Aggregation demo can find it.

Modify the WSDL

The Aggregation demo looks for special `port` elements in the `service` element in the WSDL document that describes a Web service. These `port` elements, named `DesignPresentation` and `RuntimePresentation`, identify the Web service as having an associated visual component and as capable of participating in the Aggregation demo. If it is not already running, start the WSDE. Follow these steps:

1. Make sure you are viewing the default perspective.
2. Expand your solution (`WeatherForecastSolution` for the example), expand the project (`WeatherForecast`), and expand the `servlets` folder containing the WSDL document.
3. Double-click on your WSDL document to open it. Scroll down until you find the `service` element.
4. Make sure the `location` attribute of the `service.soap:address` equals the URL of your local SOAP server. The WSDE generates this value when you created your Web service. For Embedded WebSphere, it should be `<soap:address location="http://localohst:8080/soap/servlet/rpcrouter"/>`
5. Add the special `port` elements just before the `service` element end tag. You will need to use the appropriate binding name (the value of the `name` attribute of the `binding` element in your WSDL document) for the `binding` attribute. The local version of the Aggregation demo in WSTK 2.2 expects the URL in the `location` attribute to be relative to an application server Webgroup defined for the Aggregation demo (it is named `aggr`), so you *must* use a URL of the form `jsps/<your-jsp-name>.jsp`. The following listing shows what was added for the weather forecast Web service.

```
    <port name="DesignPresentation"
>
       <soap:address location="jsps/forecast.jsp"/>
     </port>
     <port name="RuntimePresentation"
>
```

```
    <soap:address location="jsps/forecast.jsp"/>
  </port>
```

6. Press the **Ctrl-S** buttons, or **File** > **Save**, to save the WSDL file. The WSDE will do some rebuilding, but you can ignore its actions.

If you have not already done so, put your JSP in the directory `WSTK_HOME/demos/aggregation/Webapps/aggregation/jsps`.

Publish the Web service

Your WSTK enabled application server and your local UDDI database must be running to continue. In the WSDE, click the icon

representing the Web Services Browser. You should see the browser tool open in a new window. It should look like *Figure 11* . If is does not, click the **Browse UDDI Registry** tab. Both the **Inquiry URL** and **Publish URL** fields should read `http://demohost:8080/uddi/servlet/uddi` to indicate the tool is working with the local UDDI. If they do not, make sure that you have correctly set up the `wstk.properties` file in the `WSDE_HOME/itp` directory, as described above.

Once the browser tool is open in browse mode, you can expand the **Service Providers** folder to see **AggregationDemo** listed. Expand the **AggregationDemo** folder and you should see the various Web services that are part of the Aggregation demo. Now expand the **Service Interfaces**. You should see the various interface definitions for the Web services in the Aggregation demo. If you click on one of the service interfaces, you will see the "interface WSDL" for that service on the right side of the tool. Now expand one of the services under the Aggregation Demo service provider. You will see an entry called "Implementation". Click on it and you will see the "implementation WSDL" for that service implementation. You can have some fun reading those WSDL documents, but I'll get to publishing.

To publish your Web service implementation:

1. Select the **Publish Service Implementation** tab. You'll see the appropriate dialog for Web service publishing. It should look like *Figure 12* .
2. In the **Business Name** field, type "AggregationDemo" to indicate that the service will be "owned" by the AggregationDemo business. The **Business Description** will be filled in automatically.
3. To fill in the **Select a WSDL File** field, expand the **Computer** folder, and go to a location where you have the WSDL file for your Web service. You should use the WSDL file you just edited with the WSDE. The example uses `WSDE_HOME/itp/workbench/WeatherForecastSolution/WeatherForecast/servlets/Wea`
4. Next you need to categorize the Web service. Click the **Add** button to bring up the **Category Selection** dialog box (see Figure 13). In the **Select TModel Key** field, select either **NAICS** or **UNSPSC**. This will bring up the **Select a Category from the Tree Below** dialog box (see *Figure 14*).
5. In the **Select a Category from the Tree Below** dialog, you can attempt to find something that matches the semantics of your Web service, or simply expand an interesting category. When you get to a leaf just select it and click the **OK** button. Your selection will appear in the **Category Selection** dialog. You can add additional categories, but it's not necessary. Click the **OK** button in the **Category Selection** dialog.

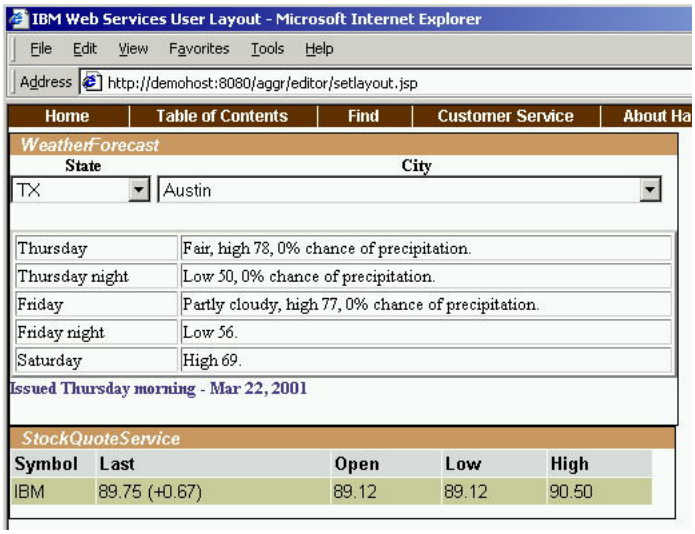
6. Now click the **Publish** button in the **Publish Service Implementation** dialog. After a few seconds you should see a popup confirming that your service is published. Click the **OK** button.
7. Now click the **Browse** tab in the Web Service Browser. Click the **Refresh** button. Expand the **UDDI** folder and the **AggregationDemo** folder, and you will see your Web service implementation listed. Expand the **Service Interfaces** folder, and you will see your Web service interface listed. *Figure 15* shows the results after publishing the weather forecast Web service.
8. Now you can exit the Web Service Browser and WSDE.

Run the Web service in the Aggregation demo

If your WSTK enabled application server and your local UDDI database are running, you are ready to to bring up the Aggregation demo that includes your Web service. To do so:

1. Point a browser at the Aggregation demo (e.g., `http://demohost:8080/aggr/editor/index.html`). You'll see the log-in page. Remember that it is important to deselect **Network Connected** if you are not connected. Note that the selection has no bearing on the weather forecast Web service. You must change the OFFLINE property in the `forecast.ini` file. If you change the property, you will have to restart your application server.
2. Select the **Edit Services** button. You will see a page, similar to *Figure 16* , with a palette of Web services applications on the left and a layout area on the right. Your application will be in the palette. Notice the WeatherForecast application in *Figure 16* .
3. Select your application in the palette. An outline of your application will appear in the layout area on the right. Click the **Save Layout/Return** button. Your service will now be functional. You can also add other applications. The page, Figure 17, shows the WeatherForecast application running with the StockQuoteService application.

Figure 17



Congratulations! You have succeeded in creating a Web service and integrating it with the Aggregation demo.

Section 7. Resources and Feedback

Resources

* Download the *source code* for this tutorial.

Your feedback

Please let us know whether this tutorial was helpful to you and how we could make it better. Feel free to suggest improvements or other tutorial topics you'd like to see covered. Thanks!

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The Toot-O-Matic tool is a short Java program that uses XSLT stylesheets to convert the XML source into a number of HTML pages, a zip file, JPEG heading graphics, and PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML.