
Use Eclipse to create Web service clients for Domino

by Paul T. Calhoun



Paul T. Calhoun
Chief Technology Officer
NetNotes Solutions Unlimited, Inc.

Paul T. Calhoun is a 16-year veteran of the IT industry. Currently, as CTO of NetNotes Solutions Unlimited, Inc. (www.nnsu.com), Paul's primary focus is on retooling non-Java/J2EE developers for Java and J2EE development through online courses, seminars, and classroom instruction. Paul holds certifications in both systems administration and application development from Microsoft, Lotus, and IBM, including Solution Developer (WebSphere Studio 5.0) and Application Developer (Rational Application Developer for WebSphere Software V6.0). Paul may be contacted at pcalhoun@nnsu.com.

Because Domino 7 now is capable of serving Web services (LotusScript or Java), it also provides an opportunity to create clients on the Notes/Domino platform to consume those Web services. The problem is finding the right tools that will help you create Web service clients without expending the majority of your budget. You may already have an integrated development environment that includes these tools, but what if you don't? Wouldn't it be great if you could use a free tool to help create these clients? Well, you're in luck. Eclipse 3.2 includes all of the tooling necessary to create Web service clients for Domino without spending any of your information technology budget.

In this article, developers learn how to write code for a Web service client that makes a request of a Web service (in this case, a Domino Web agent that queries the Domino Directory) and handles the response. I demonstrate step-by-step how to use the open-source tools of Eclipse 3.2 to create the Domino 6.x Web service client, a Java agent that takes inputs from browser users via a Domino form. Then I show you slight modifications for the same solution that take advantage of Domino 7.x. You don't need to know Web Services Description Language (WSDL) or Simple Object Access Protocol (SOAP) Extensible Markup Language (XML) — Eclipse (or Domino 7) handles it for you.

I don't go through the steps of writing code for a Web service — there are plenty of articles published on the topic.¹ I supply the Web service for the examples as a Domino agent (for 6.x users) and as a Web service (for 7.x users) in a downloadable database.² When you install the database on a

¹ For example, see Michael Thomas Mohen's article "Creating and testing Web services with the new design element in Domino 7" (*THE VIEW*, March/April 2006).

² You can download the sample WebServiceClient database (wsclient.nsf) at www.eVIEW.com. From the home page, click Search → Browse Issue → November/December 2006 → Use Eclipse to create Web service clients for Domino. Scroll to the bottom of the page for the download files.

Domino 6 or 7 server in the root of the data directory, you can then call the Web services by the appropriate URL. The download file also includes the code for the Web service clients that you'll create and the forms (6.x and 7.x versions) that collect the input for the service.

Note!

I created wsclient.nsf in Domino 7. Because Domino 7.x databases have the same operational data source as Domino 6.x databases, they are largely backward compatible, except for unique design elements and functions. If you use Domino 6.x to open the wsclient.nsf application, the Web service design element (GetInetAddress) will be obscured but an agent (GetInetAddressWebService) will provide the same service.

Let's start by reviewing the example application and the environment you need to develop such an application.

The Web service client example application

The example Web service listens for a Web service client to make a request of it and then sends a response. The Web service provided takes a user name as input and goes to the Domino Directory to obtain the user's valid Internet address. The diagram in **Figure 1** shows the six-step process flow of the request and response the Web service client implements:

1. A browser user enters a value (in the example, a name to look up in the Domino Directory) in a single text field on a Domino HTML form and clicks a Submit button. The button code uses the WebQuerySave event to submit the value to the Web service client agent.

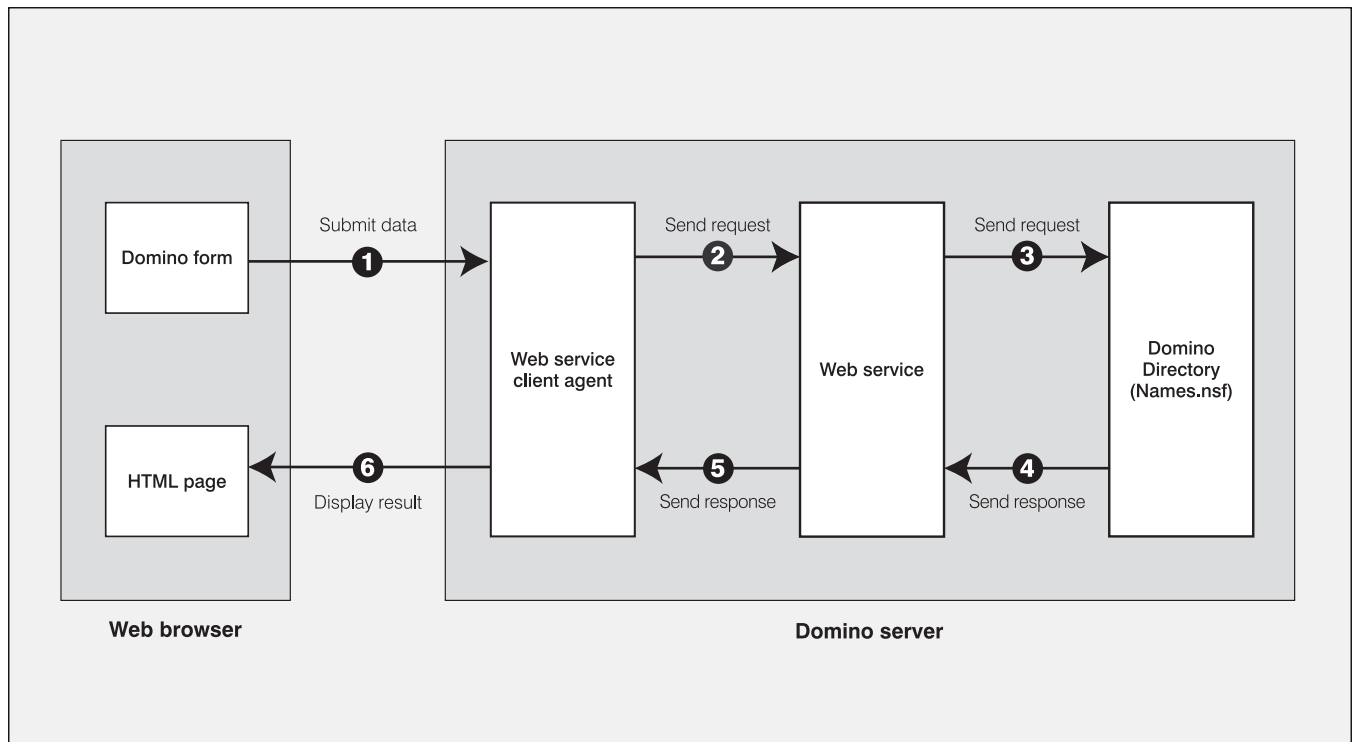


Figure 1 Process flow for the example application

2. The Web service client agent sends the request to the Web service.
3. The Web service processes the request and sends it to the Domino Directory.
4. The Domino Directory returns the response (the Internet address for the provided name) to the Web service.
5. The Web service sends the response to the Web service client agent.
6. The Web service client agent sends the response to the browser as an HTML page that displays the address to the user.

The Web service returns the Internet address from the Domino Directory only when the user supplies a valid user name in the text field.

In the example application in this article, a Domino server hosts both the Web service (a server-based process) and the client we'll write for it (in this case, an agent). The example does not use third-party Web services, Java classes, or XML utilities. All of the Java objects are native to the Domino environment.

Requirements for following the examples to develop a Web service client

In order to follow along with the demonstrations, you must have a firm grasp of basic Domino development skills — for example, you should know how to create forms and agents. You also need to install the following tools:

- Domino test server (version 6.x or 7.x)
- Domino Designer (version 6.x or 7.x)
- Eclipse 3.2 with the Web Tools Platform (WTP) Project plug-in³
- Apache Web Server with the Tomcat 5.5 add-in

For simplicity, you can install and configure all of these tools on the same computer (such as your

developer workstation). You could also use a separate machine for the Domino test server.

You must install the Apache Web Server with the Tomcat add-in on the computer where Domino Designer and Eclipse are installed. You also have to configure Eclipse to use Tomcat as the default test server.

The most significant distinction between writing client code for different Domino versions is that Domino 7.x can automatically generate a WSDL file from the Web service code, and Domino 6.x cannot generate the WSDL file. The WSDL file is necessary for producing the SOAP structure needed to communicate with the example Web service. In the example case, Eclipse generates the WSDL file from the Web service for Domino 6.x users.

I'll show you how to create the Web service client for Domino 6.x first. Then, I'll show how the steps differ when creating a Domino 7.x Web service client.

Tip!

If you use WebSphere Studio Application Developer or Rational Application Developer instead of Eclipse, you can use the same basic steps to create your Web service clients.⁴

Developing the Web service client for Domino 6.x

In order to create a Web service client that executes as an agent on the Domino 6.x server, you have to complete the steps that are outlined below:

1. Create a dynamic Web project using the Eclipse workbench.

³ Download the complete Eclipse 3.2 platform with the WTP Project plug-in from this Web site: <http://download.eclipse.org/webtools/downloads/drops/R-1.5.0-200606281455/>

⁴ See Jochen Finkbeiner's two-part article "Consume Web services in Lotus Notes applications" (*THE VIEW*, July/August 2006 and September/October 2006) where he creates a Web service client using Rational Software Architect.

2. In Eclipse, create the WSDL file that produces the SOAP envelope needed to access the Web service from the Web service client.
3. Test the Web service using Apache Tomcat.
4. Create the Domino Web service client as an agent in Domino Designer and Eclipse.
5. Create a form in Domino Designer.
6. Test the Web service client in a Web browser on a Domino test server.

Create the project

First, create a project to contain the Web service client code and the Java Archive (JAR) libraries it uses. In the Eclipse workbench, create a new dynamic Web project:

1. From the File menu, choose New → Project to open the New Project wizard.
2. In the New Project wizard's 'Select a wizard' dialog (see **Figure 2**), expand the Web folder and select Dynamic Web Project. Click Next to continue.

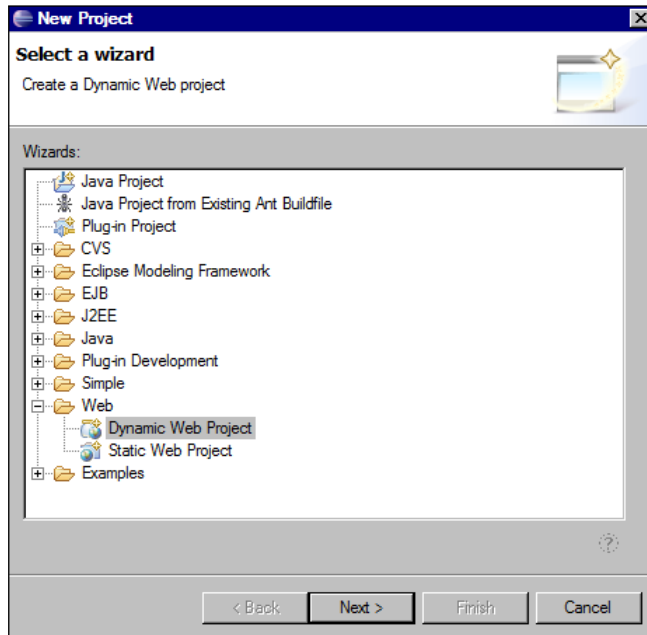


Figure 2 Opening the Dynamic Web Project wizard

3. In the next dialog (see **Figure 3**), provide a project name (DominoWebServiceClient in the example) and ensure that the Apache Tomcat v5.5 server acts as the runtime target. Click Finish.
4. When the License Agreement dialog box for the Sun Developer Network appears (see **Figure 4**), click the I Agree button to cache the Web application resource and create the project. If you don't click I Agree, Eclipse will create the project but you can't use the Apache Tomcat server to test the WSDL file.

Add the appropriate Java libraries

Next, add the Notes.jar library to the project's class path. Make sure you're in the Java perspective (the default) and follow these steps:

1. In the Package Explorer view of the Eclipse workbench, right-click the project name and choose Properties from the context menu (see **Figure 5**).
2. In the Properties for DominoWebServiceClient dialog box that opens, choose Java Build Path from the list on the left and select the Libraries tab (see **Figure 6**).

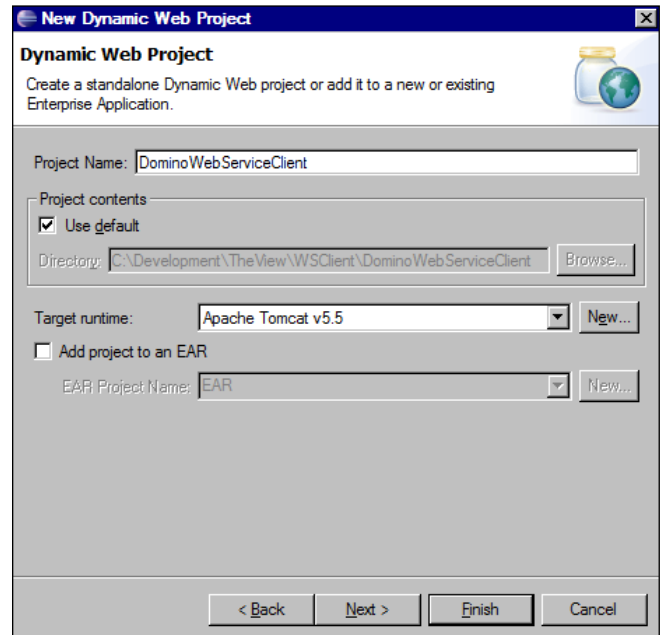


Figure 3 Naming the project and selecting the test server

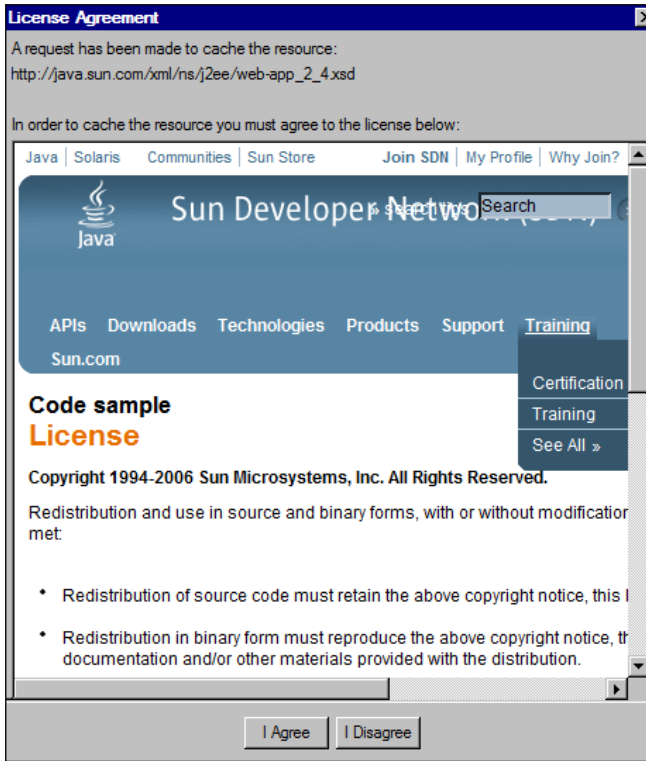


Figure 4 You must agree to the license agreement to proceed.

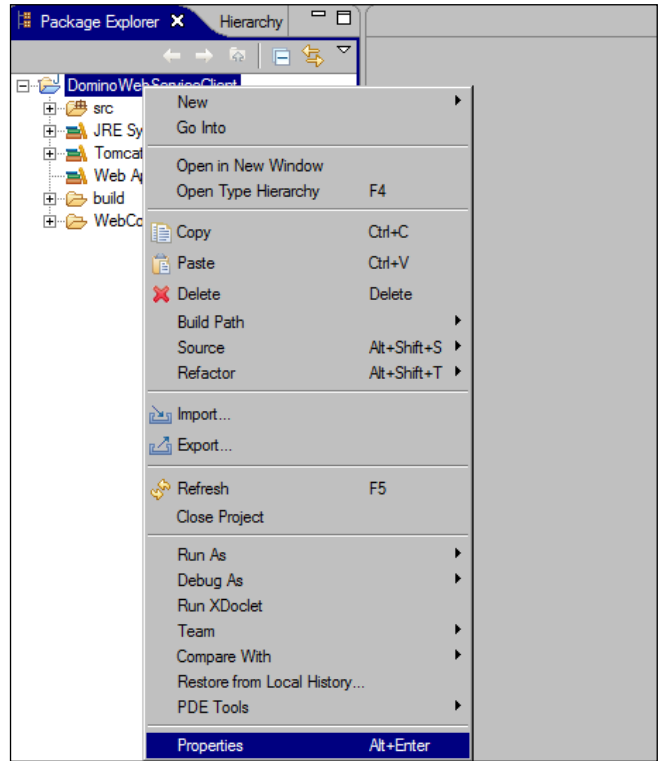


Figure 5 Opening the properties for the DominoWebServiceClient project

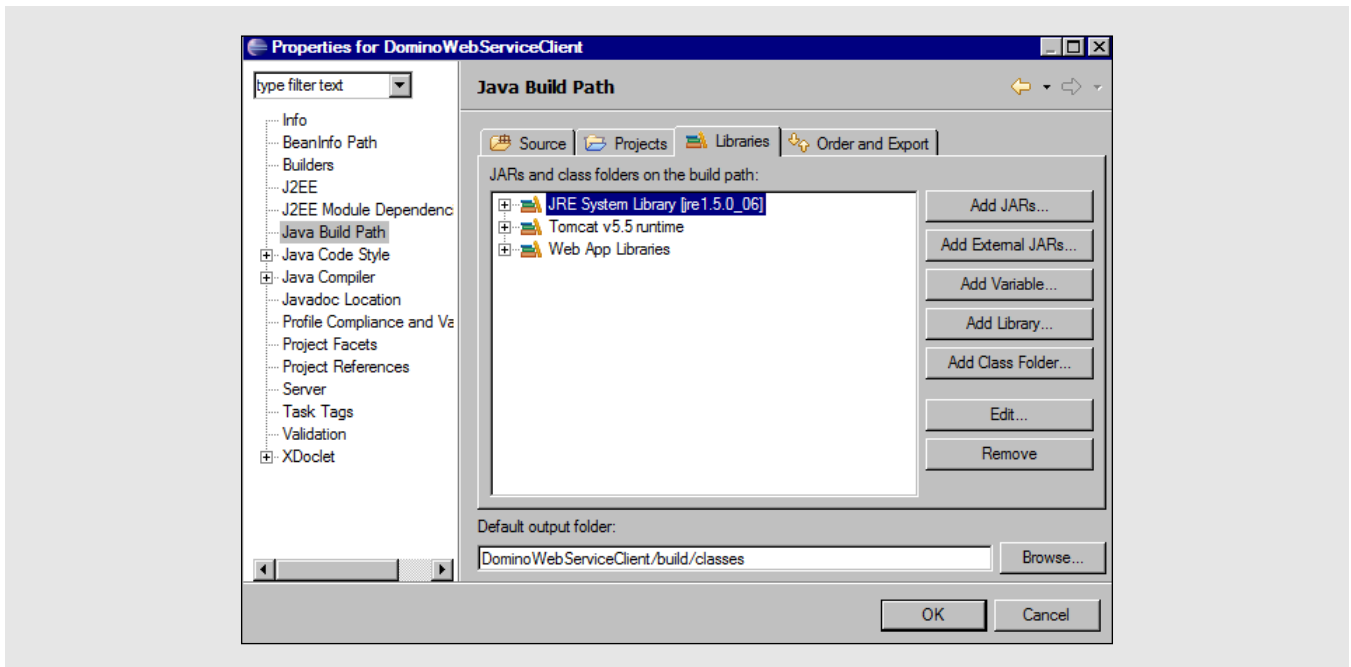


Figure 6 Navigating to the Libraries tab of the Java Build Path in Domino 6.x

3. On the Libraries tab, click the Add External JARs button.
4. In the JAR Selection dialog box (see **Figure 7**), navigate to the notes folder that contains the Notes.jar file. In the Domino 6.x platform, the JAR file is located in the directory Notesinstalldir\Notes.jar. (In the Domino 7.x platform, go to the Notesinstalldir\jvm\lib\ext\Notes.jar directory.) Click Open to add the file to the class path.
5. The Notes.jar file should now appear in the listing on the Libraries tab. Click OK to return to the Eclipse workbench.

Create the WSDL file and SOAP XML structure

The WTP Project plug-in for Eclipse includes a Web Service wizard that creates a WSDL file from a Java bean. To use this tool, you must create a Java bean that duplicates the method structure of the Web service the example client will call — that is, a Web service that accepts a String value and returns a String value.

Create the Java bean

To create the bean that generates the WSDL file, create a new class in the DominoWebServiceClient project:

1. In the Package Explorer view, expand the project, right-click the src folder, and select New → Class to open the New Java Class wizard.
2. In the wizard's opening Java Class dialog (see **Figure 8**), provide a package name and a class name.
3. Click Finish to create the class.

Eclipse automatically creates some basic class code for the new Java bean, which opens in the Java editor in the middle of the Eclipse workbench. You must modify the code to handle the string values. The boldface lines in **Figure 9** show the code I added for handling the strings. Save the code. It now represents a simple bean that contains a single method, `getInetAddress()`, that accepts a String value and returns a String value.

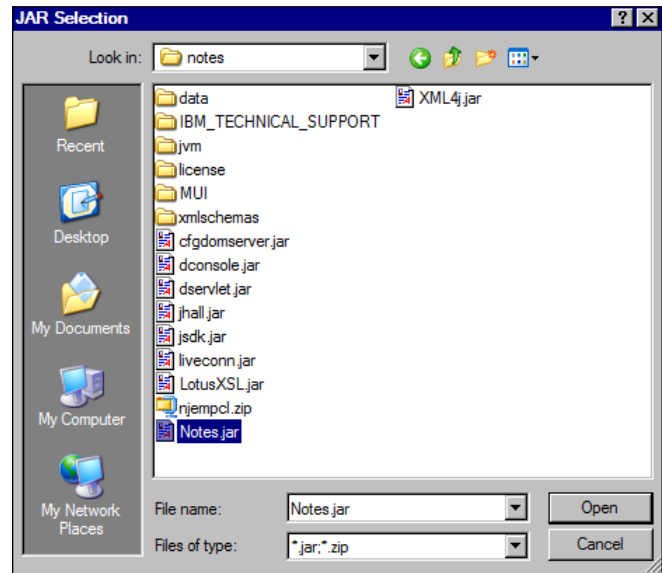


Figure 7 Selecting the Notes.jar file to add to the project's build path

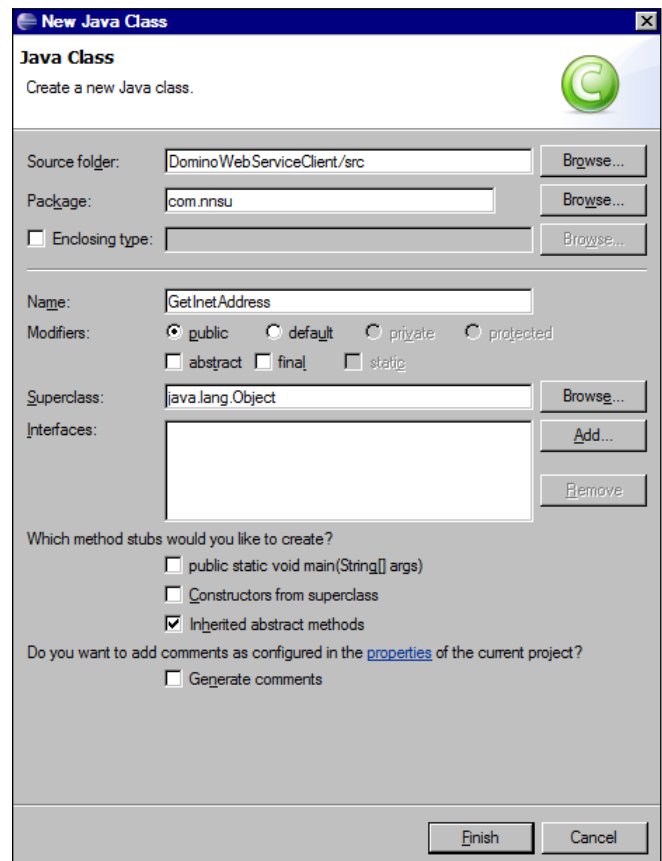


Figure 8 Naming the Java bean and package

Configure the bean as a Web service

To configure the new bean in the Eclipse workbench as a Web service, follow these steps:

1. In the Package Explorer view, expand DominoWebServiceClient → src → com.nnsu.
2. Right-click the Java bean (GetInetAddress.java) and choose Web Services → ‘Create Web service,’ as shown in **Figure 10**. The Web Service wizard opens.
3. In the wizard’s opening Web Services dialog, change the settings so the dialog appears as shown

```
package com.nnsu;
public class GetInetAddress {

    String inetAddress;

    public String getInetAddress(String perName) {
        inetAddress = perName;
        return inetAddress;
    }
}
```

Figure 9 The code for the GetInetAddress Java bean

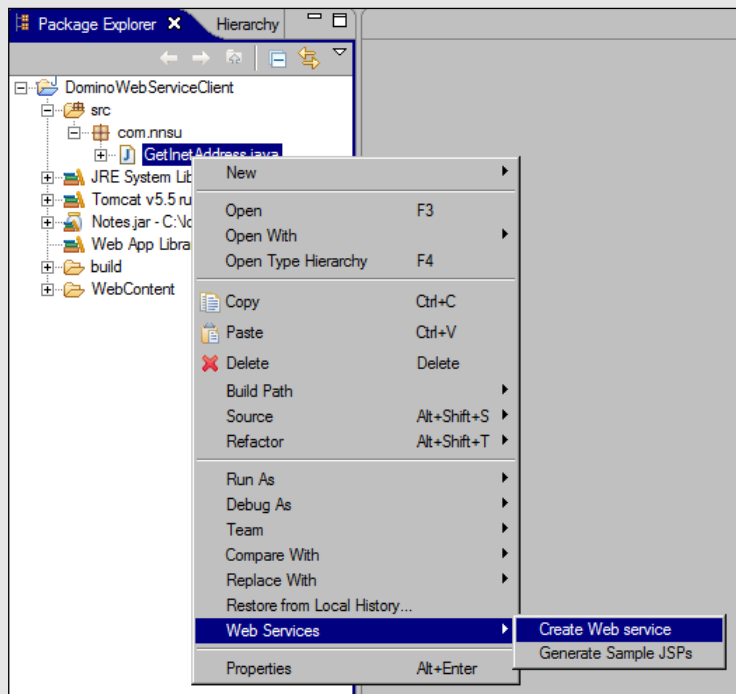


Figure 10 Opening the Web Service wizard

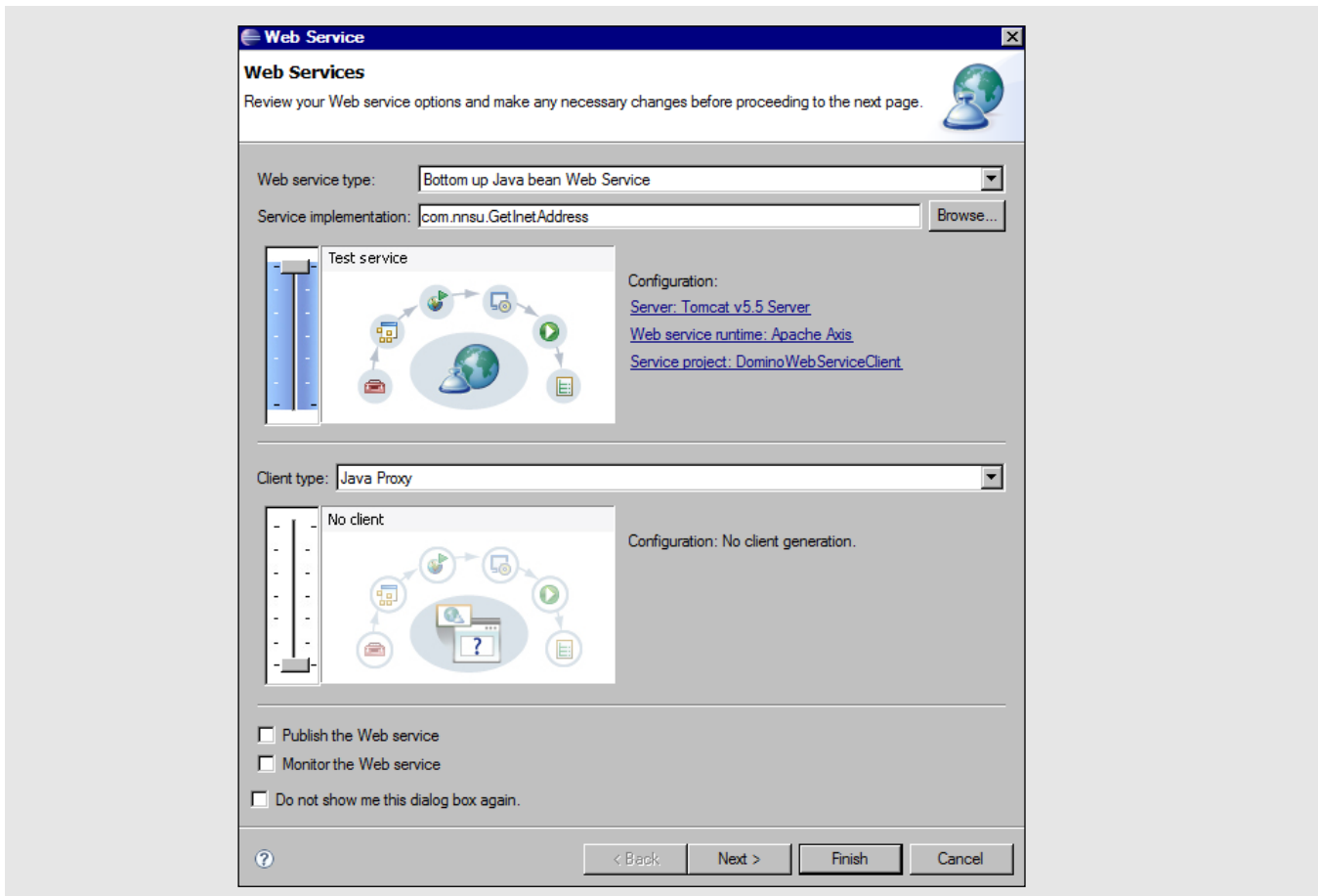


Figure 11 Setting the Web service options

in **Figure 11** to create the Web service and execute it in the Web services testing environment. Click Next to continue.

4. In the Web Service Java Bean Identity dialog (see **Figure 12**), under 'Style and use,' click the document/literal radio button. Click Next.

Note!

For a useful discussion on styles, see Russell Butek's IBM developerWorks article, "Which style of WSDL should I use?"⁵

⁵ <http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/>

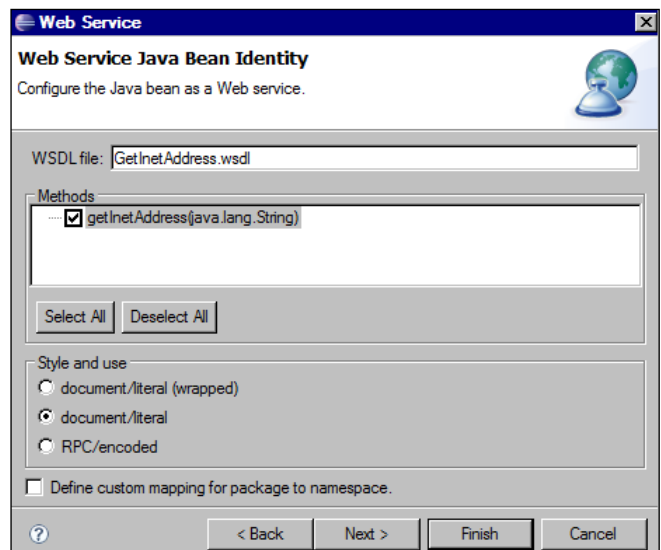


Figure 12 Configuring the WSDL style and use

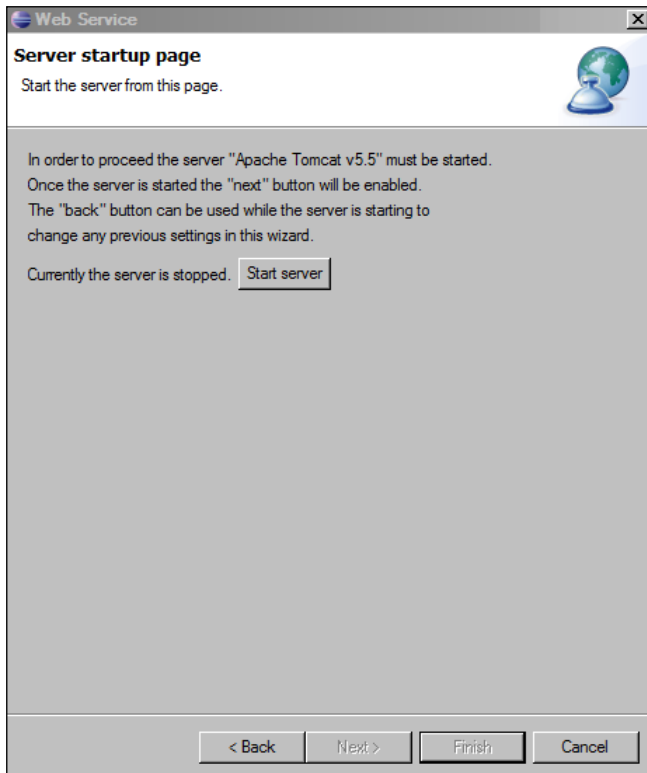


Figure 13 Starting the test server

5. In the 'Server startup page' dialog (**Figure 13**), click the 'Start server' button to start the Tomcat test server and load the Web service test client. When the server starts, click Next.
6. In the Test Web Service Page, leave the default test facility as Web Services Explorer. Click Finish.

The wizard finishes by loading Web Services Explorer in the middle of the Eclipse workbench.

Test the Web service

Now you can test the Web service generated by the Web Service wizard. In the Web Services Explorer view, follow these steps to test the Web service:

1. In the Navigator on the left, expand the GetInetAddressSoapBinding entry and select the getInetAddress method. The Invoke a WSDL Operation panel opens in the Actions section on the right (see **Figure 14**).
2. Type a name into the perName field. (I entered "Paul" in the example.)

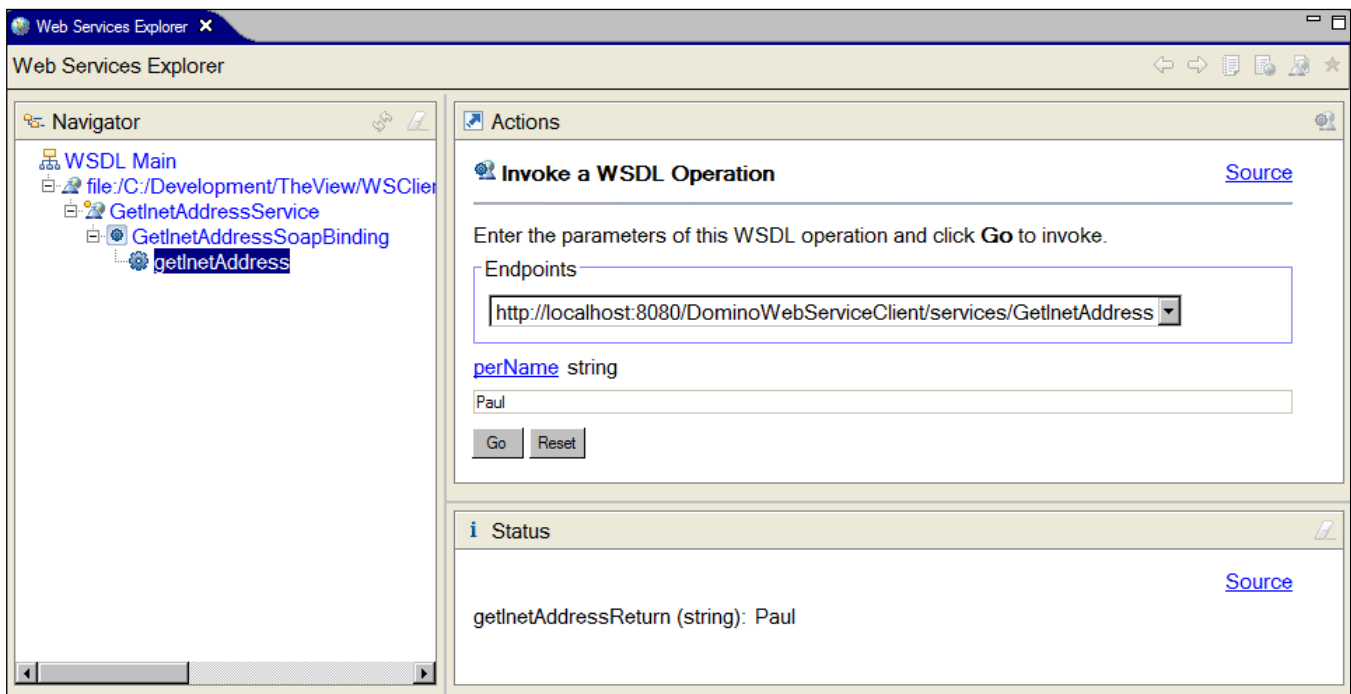


Figure 14 Testing the Web service

- Click the Go button and watch the Web service return the supplied value in the Status section.
- In the top-right corner of the Status section, click the Source link. The SOAP XML request and response produced by the Web service automatically appear, as shown in **Figure 15**.

Keep the Web Services Explorer open on your workstation.

While this basic test of the Web service simply returned the same string it was provided, it generated two important elements that you need to create a Web service client:

- The SOAP XML structure necessary for communicating with the Web service.** You now have the exact SOAP XML structure that the server-based Web service needs as a request, as

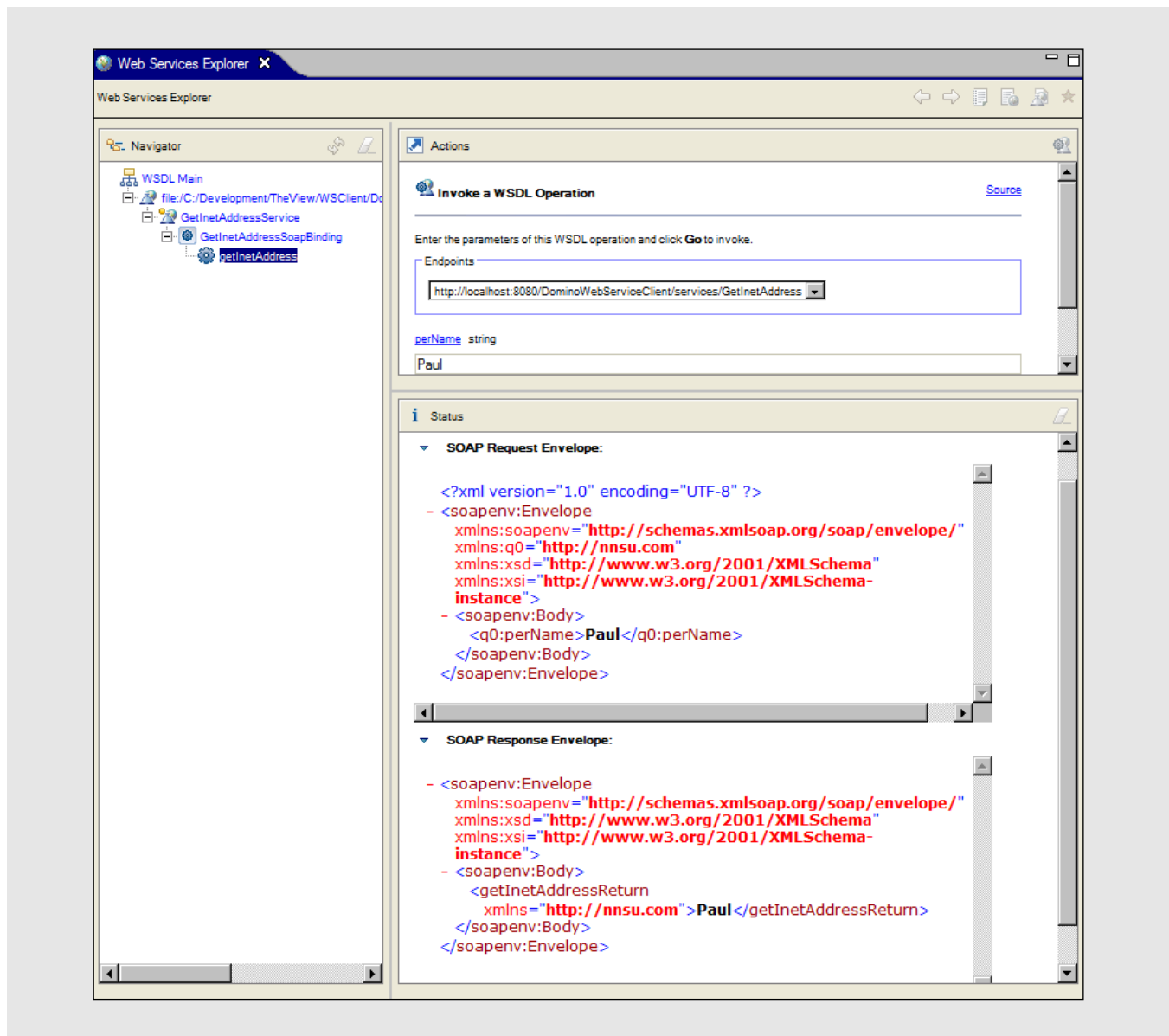


Figure 15 Viewing the SOAP XML source code for the GetInetAddress operation

well as the exact SOAP XML structure of a response that the Web service returns for successful requests.

- **A valid WSDL file to describe the Web service.** Look in the Package Explorer view in the WebContent folder. Open the new folder named “wsdl” to find a WSDL file with the same name as the Java bean (GetInetAddress.wsdl).

If you close Web Services Explorer at this point, you can come back to it later and test the GetInetAddress.wsdl file.

Testing the Web service from the Eclipse workspace

While the Tomcat server is still running, go to the Package Explorer view, expand DominoWebServiceClient → WebContent → wsdl, right-click the GetInetAddress.wsdl, and choose Web Services → Test with Web Services Explorer, as shown in **Figure 16**. The Invoke a WSDL Operation panel opens in the Actions section, as in **Figure 14**, and you can follow steps 2 – 4 to test the Web service.

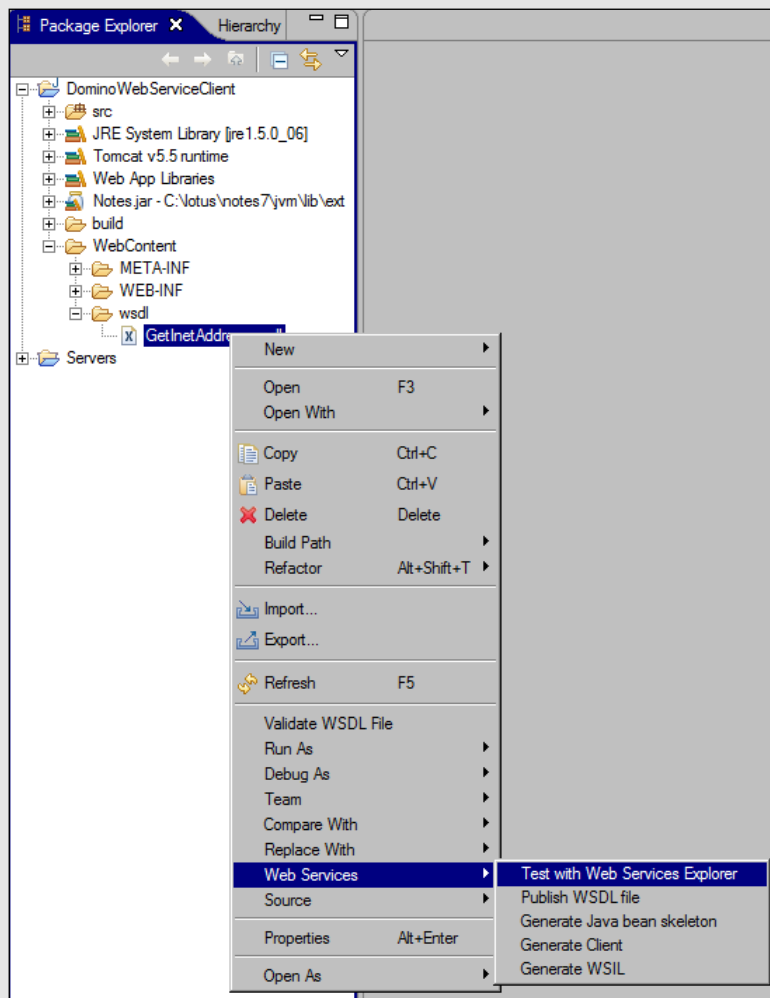


Figure 16 Testing the WSDL file from the Package Explorer view

Note!

Restarting Web Services Explorer accesses the Web service on the running server. The Tomcat server does not stop running until you shut down Eclipse or manually shut down the server. If you do shut down the server or Eclipse, you would need to run the project on the server again before following the testing procedure.

When you're satisfied with the WSDL file and you've generated the SOAP XML structure necessary for communicating with the Web service, you can start creating the Domino agent for the Web service client.

Tip!

You can test any Web service this way if you have access to the Web service's WSDL file.

Create the Web service client agent

The Web service client agent must perform the following tasks:

- Build the SOAP XML request.
- Send the request to the Web service URL.
- Parse the SOAP XML response and process the results.

Use Domino Designer to create the agent and incorporate the SOAP XML code from Eclipse into the agent code. Follow these steps in Domino Designer:

1. Create a Java agent in the normal Domino fashion and name it "GetInetAddressWebServiceClient."
2. Set the Agent properties, as shown in **Figure 17**. The agent triggers on the event of an action menu selection. Use the default runtime security level, '2. Allow restricted operations.' Close the Agent properties dialog.

In Eclipse, create a new Java class to use for coding the Web service client agent:

3. In the Package Explorer view, expand the DominoWebServiceClient project, right-click the

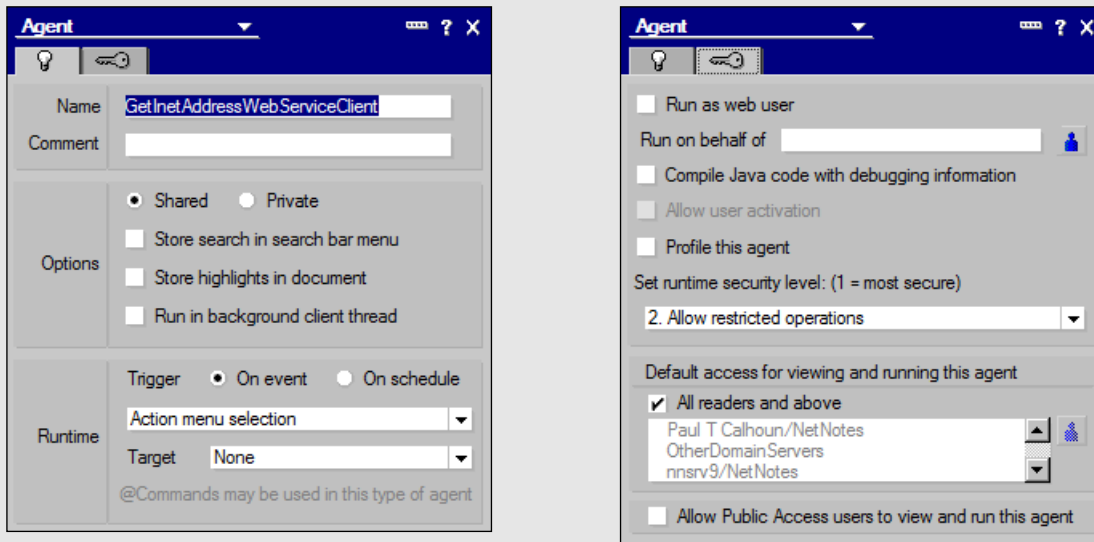


Figure 17 Setting properties for the GetInetAddressWebServiceClient agent in Domino

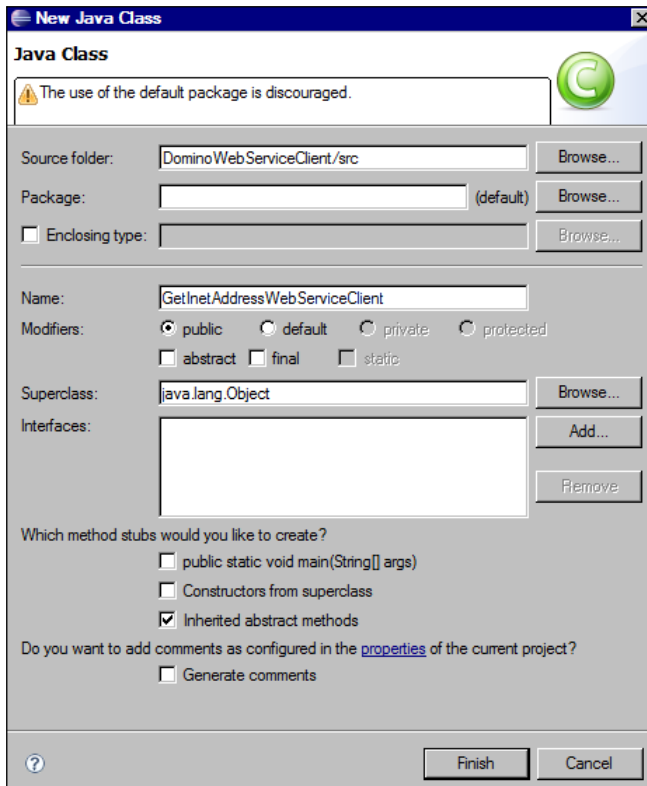


Figure 18 Naming the Java class in Eclipse used for coding the Web service client agent

src folder, and choose New → Class. The New Java Class wizard opens.

4. Provide a class name for the Web service. I entered “GetInetAddressWebServiceClient” in the example shown in **Figure 18**. Eclipse doesn’t include Domino agents in its packages by default, so you can leave the Package field blank this time. Click Finish to create the class. Eclipse automatically opens the Java template for the new class code in the Java editor.
5. In the Java editor in Eclipse, delete Eclipse’s Java template code and in its place, copy and paste the template code from the new Java agent you created in Domino Designer (see **Figure 19**). The commented line you see in the figure, which is there by default, is where you place the code for the agent functionality (the Web service code).

Now you’re ready to add the code that builds the SOAP request, passes the request to the Web service URL, parses the response, and processes the results. Continue in Eclipse:

6. Go back to the Web Services Explorer view of the SOAP XML request and response. (Follow the

```
import lotus.domino.*;

public class GetInetAddressWebServiceClient extends AgentBase {

    public void NotesMain() {

        try {
            Session session = getSession();
            AgentContext agentContext = session.getAgentContext();

            // (Your code goes here)

        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Figure 19 The Java agent’s template code from Domino Designer

instructions in the subsection “Testing the Web service from the Eclipse workspace.”)

7. Using the SOAP XML code as an example, write the code that will produce the correct SOAP request. This code is a combination of the XML from the Web Services Explorer and manually generated code that implements the functions for sending the request and processing the response. Be sure to save the code.
8. Copy the complete class code in Eclipse and paste it in the script area for the GetInetAddressWebServiceClient agent in Domino Designer. Save the code once more.

The code listing in **Figure 20** shows the complete agent code, along with my comments. The agent takes

the value in the NameLookup field and adds it to the SOAP XML request (see the embolded lines of code). I don’t provide a detailed walkthrough here since the code is thoroughly annotated. Note that all of the Java objects in the code are native to the Domino environment.

Now you can create the user interface (UI) for the browser users.

Create the Domino form

Recall that in the UI (a form), the browser users enter a name for lookup and submit it. A WebQuerySave event in the Submit button code invokes the agent you just created.

```
//Start code
import lotus.domino.*;
import java.io.*;
import java.net.*;
import org.w3c.dom.*;

public class GetInetAddressWebServiceClient extends AgentBase {

public void NotesMain() {

    try {
        Session session = getSession();
        AgentContext agentContext = session.getAgentContext();
        Database db = agentContext.getCurrentDatabase();
        lotus.domino.Document doc = db.createDocument();
        //Read the submitted Web form and get the lookup name value from the NameLookup field
        lotus.domino.Document rDoc = agentContext.getDocumentContext();
        String nlookup = rDoc.getItemValueString("NameLookup");

        // Create string buffers to hold the SOAP request, response, and returned value
        StringBuffer soapRequest = new StringBuffer();
        StringBuffer soapResponse = new StringBuffer();
        String retvalue;
        //Append the SOAP header to the SOAP request
        soapRequest.append(SoapHeader());

        //Append the lookup name value in the proper tag
```

Continues on next page

Figure 20 Complete, annotated code for the Web service client agent

```

XMLsoapRequest.append("<q0:perName>" + nlookup + "</q0:perName>");

//Append the SOAP footer to the SOAP request
soapRequest.append(SoapFooter());

    //Send the SOAP request to the Web service URL
    //Create a new URL object
    URL wsurl = new URL("http://nnsrv9.nnsu.com/wsclient.nsf/GetInetAddressWebService");

//Post the request to the URL using the HTTP Post method
    soapResponse.append(httpPost(wsurl,soapRequest.toString()));

//Store the response SOAP XML to a Domino Item object
    doc.replaceItemValue("xData",soapResponse.toString());
Item xData = doc.getFirstItem("xData");

//Create a new XML DOM document to hold the parsed XML node tree
org.w3c.dom.Document xdoc = xData.parseXML(false);
//Read the node tree to retrieve the Web service response
NodeList nl = xdoc.getElementsByTagName("getInetAddressReturn");
Node n = nl.item(0);
Node cn = n.getFirstChild();
retvalue = cn.getNodeValue();

//Create a print writer to output the data to the Web browser
PrintWriter webout = getAgentOutput();
//Write the output to the Web browser as HTML
webout.println("Content-type:text/html");
webout.println("");
webout.println("<!DOCTYPE HTML PUBLIC '-//W3C//DTD HTML 4.01 Transitional//EN'>");
webout.println("<html><head><meta http-equiv='Content-Type' content='text/html; charset=ISO-8859-1'>");
webout.println("<title>Web Service Output Page</title></head><body>");
webout.println("<h1>Internet Address for "+ nlookup+"</h1>");
webout.println("<br /><br />");
webout.println("<p>" + retvalue + "</p>");

//Complete the HTML output
webout.println("</body></html>");
} catch(NotesException e) {
    e.printStackTrace();
} catch(Exception e) {
    e.printStackTrace();
}
}
}

public String httpPost(URL url, String request) throws Exception {

```

*Continues on next page***Figure 20** (continued)

```

// This method handles normal HTTP Post requests
// Open the connection and post the request

    HttpURLConnection conn;
    Writer out;
    //Create the connection
    conn = (HttpURLConnection) url.openConnection();
    //Set the connection properties
    conn.setDoOutput(true);
    conn.setDoInput(true);
    conn.setUseCaches(false);
    conn.setRequestMethod("POST");
    conn.setRequestProperty("SOAPAction", "com.nnsu.hello");

    //Create an output stream writer to send the request data
    out = new OutputStreamWriter(conn.getOutputStream(), "UTF8");
    out.write(request, 0, request.length());
    out.flush();
    out.close(); // ESSENTIAL!

    /* Read the response back from the Web service*/
    //Create the input stream reader
    InputStream ins = conn.getInputStream();
    BufferedReader in = new BufferedReader(new InputStreamReader(ins));
    String inputLine = "";
    StringBuffer response = new StringBuffer();
    //Read the input stream
    while ((inputLine = in.readLine()) != null){
        response.append(inputLine);
    }
    //Close the input stream
    in.close();
    //Disconnect from the URL
    conn.disconnect();
    //Return the results
    return (response.toString());

} // End HTTP Post

public StringBuffer SoapHeader() {
    StringBuffer sh = new StringBuffer();
    sh.append("<?xml version='1.0' encoding='UTF-8' ?>");
    sh.append("<soapenv:Envelope xmlns:soapenv='http://schemas.xmlsoap.org/soap/envelope/'
xmlns:q0='http://nnsu.com' xmlns:xsd='http://www.w3.org/2001/XMLSchema'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'>");
    sh.append("<soapenv:Body>");
    return sh;
}

```

*Continues on next page***Figure 20** (continued)

```

    }

    public StringBuffer SoapFooter() {
        StringBuffer sf = new StringBuffer();
        sf.append("</soapenv:Body>");
        sf.append("</soapenv:Envelope>");
        return sf;
    }
}
// End of code

```

Figure 20 (continued)

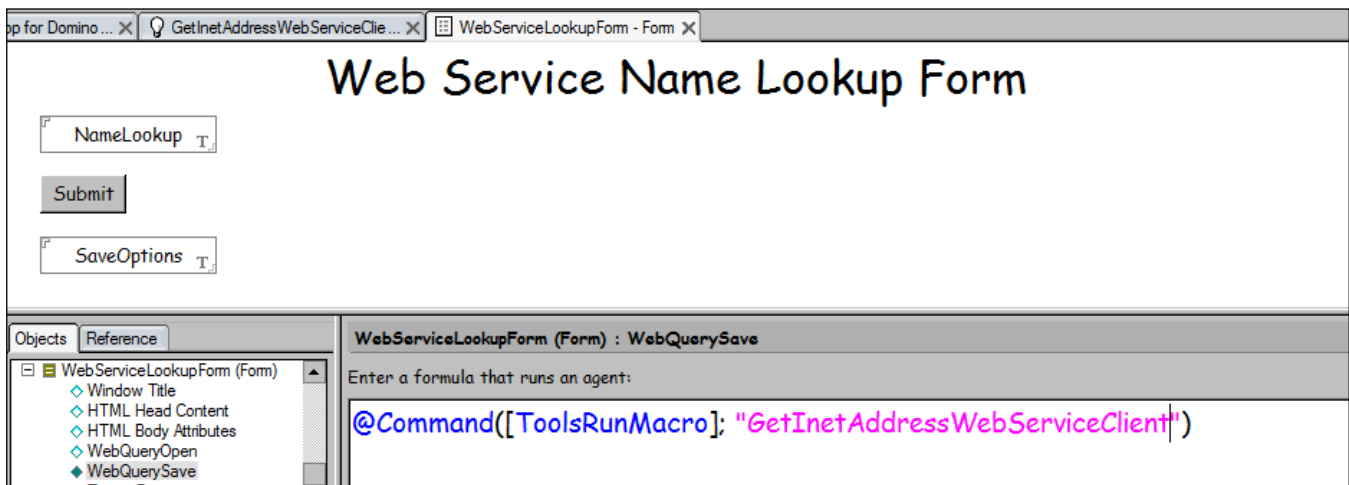


Figure 21 Posting the form to the GetInetAddressWebServiceClient agent via the WebQuerySave event

In Domino Designer, create a form in the normal fashion named “WebServiceLookupForm.” The form should look similar to the one in **Figure 21**, including a single text input field (NameLookup), a Submit button, and a hidden SaveOptions field. Since this form is simply a vehicle to submit the user’s request to the Web service, be sure to set the SaveOptions field default value to 0. Otherwise, the Domino server would save each user’s input in a document made with this form.

Then write the formula code for the WebQuerySave event. As shown in the script area in **Figure 21**, the formula should post this form to

the GetInetAddressWebServiceClient agent running on the Domino server.

Test the Web service client

This example case involves running two Web agents concurrently. You’re testing a Domino Web service client (a Web agent, GetInetAddressWebServiceClient) against a Domino Web service (another Web agent, GetInetAddressWebService). To avoid processing conflicts, you’ll need to change a setting on the Domino Web server to proceed:

1. Open the Server Configuration document and select the tabs Internet Protocols → Domino Web Engine.

- Go to the lower left-hand corner, the Web Agents section, and set the 'Run web agents concurrently?' property to Enabled, as shown in **Figure 22**.
- Restart the HTTP task that is running on the server.

If you don't perform these steps, the Domino Web server could hang while the second agent waits for the first agent to complete.

To test the Web service client, open the WebService-LookupForm form in a browser, enter a valid name (one that you know is in the Domino Directory on the test server), and click Submit. **Figure 23** shows the form loaded in a Web browser (Mozilla Firefox, in this case) and the results that the Web service agent generated and returned to the client via the GetInetAddressWebServiceClient agent.

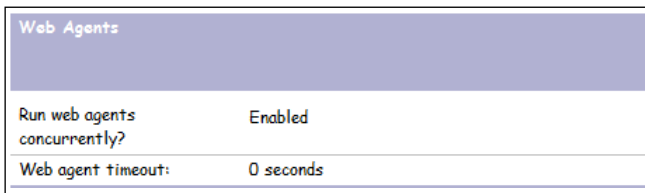


Figure 22 Enabling the Domino Web server to run Web agents concurrently

Developing the Web service client for Domino 7.x

Domino 7 has the new capability to serve Web services. One of the by-products of creating a Domino Web service is the generation of a native WSDL file in Domino. Because Domino 7.x produces the WSDL file for you, writing the client code is simpler for Domino 7.x than it is for the Domino 6.x environment. The primary difference is where the WSDL comes from. For Domino 7.x, you can import the Domino-produced WSDL file into a dynamic Web project in Eclipse and modify it so that you can test it with Web Services Explorer.

To keep the example simple, you can use the same DominoWebServiceClient project from the Domino 6.x example since we're only testing the WSDL file in the Web service test client. If you create a new project, be sure to add the appropriate Notes.jar file from the Notesinstalldir\jvm\lib\ext\Notes.jar directory to the class path.

Complete the following steps to create a Web service client agent on a Domino 7.x server:

- Import the WSDL file for the GetInetAddress Web service (installed from the download

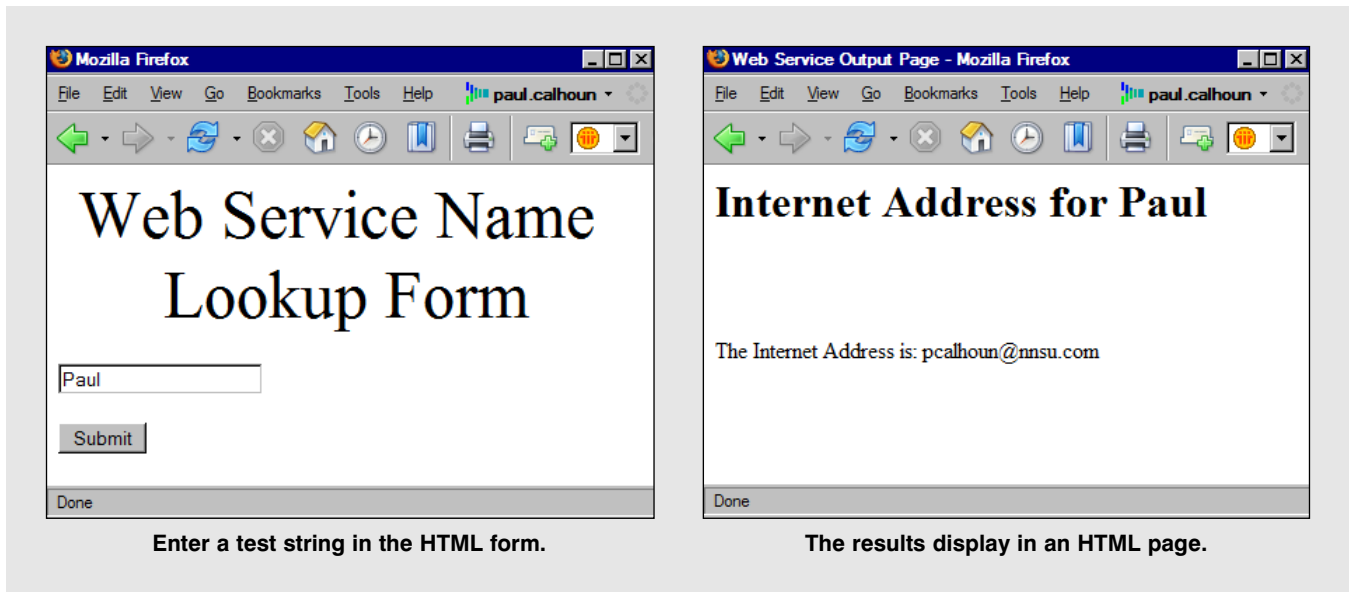


Figure 23 Testing the Web service client from Mozilla Firefox

files) from Domino Designer to Eclipse and modify it.

2. Test the Web service in Eclipse using Web Services Explorer.
3. In Domino Designer 7.x, create the Domino Web service client as an agent, incorporating the SOAP XML request and response structures from Eclipse.
4. Create the Domino form for the UI.
5. Test the new Web service client (the agent and form) in a Web browser on a Domino 7.x test server.

Detailed instructions for these steps follow.

Import the WSDL file to Eclipse

To export the WSDL file from Domino Designer 7.x:

1. Open the WebServiceClient database (wsclient.nsf).
2. In the Design pane, expand Shared Code → Web Services and click GetInetAddress. The code for the GetInetAddress Web service displays in the script area.
3. Click Export WSDL in the action bar and save the WSDL file to a temporary location on your hard drive (see **Figure 24**). Note that to reflect the 7.x platform and distinguish the file from the one previously created, I've named the WSDL file "D7GetInetAddress.wsdl" in this example.

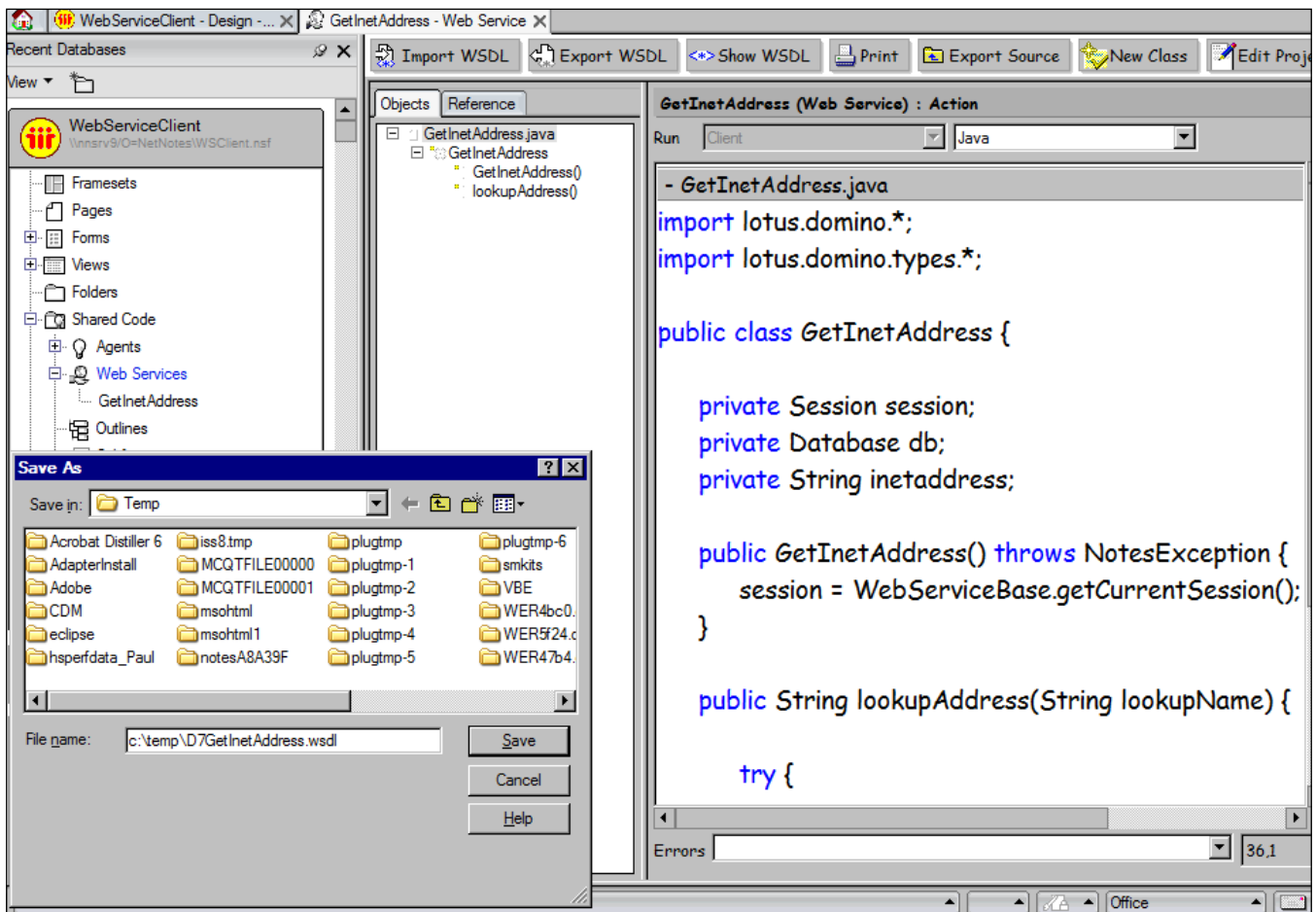


Figure 24 Exporting the WSDL file from Domino Designer 7 and saving it to the Temp directory

- In Eclipse's Package Explorer view, expand DominoWebServiceClient → WebContent.
- Right-click the wsdl folder and choose Import to open the Import wizard.
- In the Select dialog, choose 'File system' as the import source and click Next.
- In the 'File system' dialog (see **Figure 25**), click Browse to search for the folder where you saved the WSDL file exported from Domino Designer. Select the folder, find the WSDL file, checkmark it, and click Finish.

The wizard finishes by importing the WSDL file into your Eclipse workspace. Before you can test the Web service that this WSDL file references on the Domino server, you need to point to the correct location.

Modify the WSDL file

In Eclipse, open the WSDL file from the Package Explorer view and update the `<wsdlsoap:address>` tag, which is located near the bottom of the file:

```
<wsdlsoap:address location="http://localhost"/>
```

Edit this tag to point the location attribute to the Web service on the Domino server. Use the syntax `http://DomServerHostName/Database.nsf/WebServiceName`, as in this example:

```
<wsdlsoap:address
location="http://nnsrv9.nnsu.com/
WSClient.nsf/GetInetAddress"/>
```

Adjust the host name to reflect your own server environment. Then, save and close the WSDL file.

Test the Web service

To test the Web service from Web Services Explorer, follow these steps:

- In the Package Explorer view, expand DominoWebServiceClient → WebContent → wsdl.

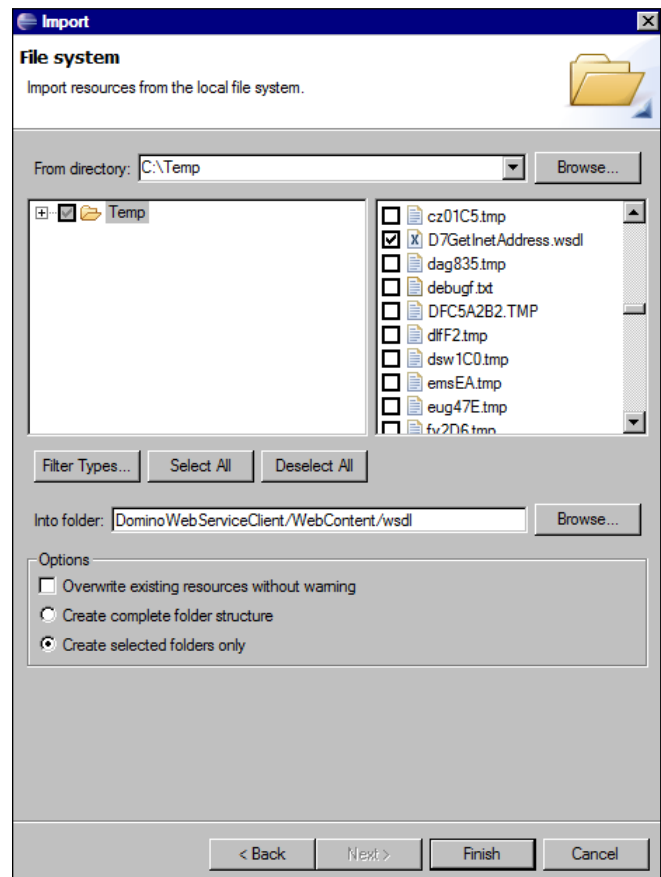


Figure 25 Selecting the WSDL file to import into Eclipse

- Right-click `D7GetInetAddress.wsdl` and select Web Services → Test with Web Services Explorer.
- In Web Services Explorer, expand the DominoSoapBinding entry in the Navigator and select `lookupAddress`, the method name for the Web service, as shown in **Figure 26**.
- In the `lookupName` field under Invoke a WSDL Operation, type a test string (a valid name in the Domino Directory) and click Go. You should see the return string below in the Status section.
- Click the Source link in the Status section to examine the SOAP XML structure that the Web service automatically creates. As in the Domino 6.x example, you use this code as part of the client agent. (See the commented code in the download for details.)

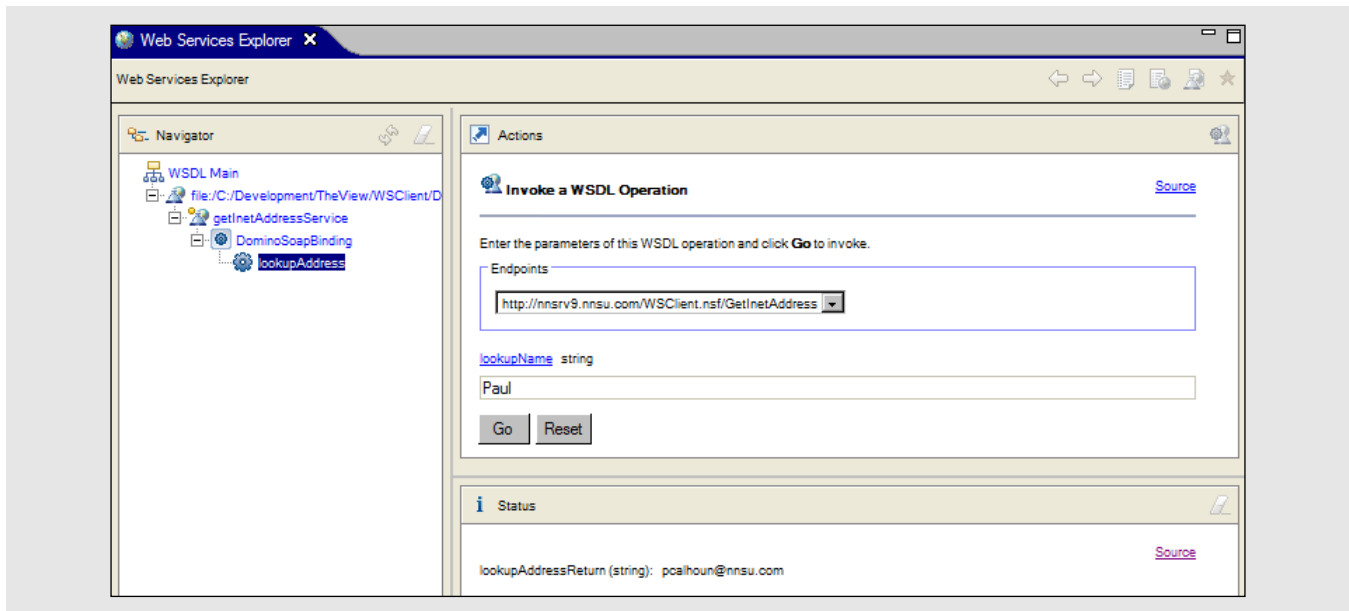


Figure 26 Testing the Web service from Eclipse's Web Services Explorer

Create the Web service client agent

Create a new Java agent in Domino Designer named “D7GetInetAddressWebServiceClient” to act as the Domino 7 Web service client agent with the same properties used for the Domino 6.x agent (refer to **Figure 17**). Follow the instructions for creating a Java class of the same name in Eclipse, copying and pasting the agent code into Eclipse's Java editor, adding the SOAP XML for requests and responses, and then copying the class code from Eclipse into the agent script area in Domino Designer. Now you have everything you need to create the Web service client agent.

I won't repeat all of the code details here. The complete Domino 7.x version of the Java code for the Web service client agent is in the download and is well commented. Functionally and procedurally, it's the same as the code for the 6.x agent: It builds the SOAP request, passes the request to the Web service URL, parses the SOAP XML response, and processes the results. The main difference is that for the 7.x environment, the client agent calls the `LookupAddress` method instead of the `GetInetAddress` method. Note that I created both methods in the Java classes in Eclipse.

Create the form for the UI

Create a form identical to the one you used to test the Domino 6.x Web service. Modify the `WebQuerySave` event to post the form to the new Domino 7 Web service client agent. In the download database, for example, the `D7WebServiceLookupForm` form invokes the `D7GetInetAddressWebServiceClient` agent.

Test the Web service client

Test the Web service client in the same manner as described for 6.x users. Submit the form to the Web service with a known valid name from the Domino Directory and look for the correct response.

Conclusion

Now you've seen how to use the Eclipse open-source tools to create Web service clients for Domino 6.x and 7.x environments. The primary benefit of this solution — aside from the fact that it's free — is that it does not require any third-party SOAP, XML or Java utilities. Everything you need is already in Domino 6.x or 7.x and Eclipse with the WTP Platform plug-in.

Resources

Articles

Michael Thomas Mohen, “Creating and testing Web services with the new design element in Domino 7” (*THE VIEW*, March/April 2006).

Jochen Finkbeiner, “Consume Web services in Lotus Notes applications, Part 1: Preparing the Java proxy” (*THE VIEW*, July/August 2006).

Jochen Finkbeiner and Bernd Straub, “Consume Web services in Lotus Notes applications, Part 2: Deployment” (*THE VIEW*, September/October 2006).

“Which style of WSDL should I use?,” Russell Butek:
<http://www-128.ibm.com/developerworks/webservices/library/ws-whichwsdl/>

Eclipse Web Tools Platform (WTP) Project

Download for the complete Eclipse WTP Project plug-in:
<http://download.eclipse.org/webtools/downloads/drops/R1.5/R-1.5.0-200606281455/>