Lab 2 – Mobile Web Services

In this lab we will create a J2ME calculator application. The calculator GUI will operate on the J2ME application but all the mathematical operations will be processed by a Web service. Although the theory behind the system may not make complete sense, the purpose of this lab is to understand how Web services can be utilized by mobile devices.

Prerequisites

Before you begin, you will need to have the following installed on your computer:

1) NetBeans IDE version 6.0 or later
2) Java Standard Development Kit 6.0 or later (if not bundled with the IDE)
3) Sun Java System Application Server 9.0 or later (if not bundled with the IDE)
4) Sun Java Wireless Toolkit for CLDC

Configuring the Web Server

If you have not registered an instance of the Sun Java System Application Server, you must do so before you can begin developing Java EE applications.

1) Choose Tools → Servers

   Note: If a default server instance has already been created then you can skip the remaining steps. The server instances are listed in the *Servers* list.

2) Click Add Server. Select Sun Java System Application Server and give a name to the instance. Click Next.

3) Specify the server information, the location of the local instance of the application server, and the domain to which you want to deploy.

4) Click Finish.

Configuring the Sun Java Wireless Toolkit

The Sun Java Wireless Toolkit allows us to create J2ME application for mobile devices. It also allows us to simulate those applications for testing. If the Sun Java Wireless Toolkit is not already installed and recognized within NetBeans then you will have to install it manually.

1) In NetBeans, go to Tools → Plugins.

2) Click on the *Available Plugins* tab. Find *Sun Java Wireless Toolkit for CLDC.*

3) Check the corresponding box and click install.

Creating the Web Service Project

1) Choose File → New Project. Select Web Application from the Web category.

2) Name the project *CalcWebApp*.

3) Click Finish.

By default, NetBeans sets the browser to open when the Java server is started. Disable this action as follows:

1) Right-click on the *CalcWebApp* node in the projects window and choose Properties.

2) Go to the Run category and uncheck Display Browser on Run. Click OK.

Creating the Web Service

1) Right-click on the *CalcWebApp* node in the projects window and choose New → Web Service…

2) Name the service CalcWS and specify any package name (i.e lab.calc).

Coding the Web Service

The calculator Web service will need to perform some basic mathematical functions. The method definitions that are required from the J2ME calculator are as follows:

1) add(double num1, double num2) : double
2) subtract(double num1, double num2) : double
3) multiply(double num1, double num2) : double
4) divide(double num1, double num2) : double
5) exp(double num1, double num2) : double

Note: Use the *Design* view in the NetBeans IDE to add operations to the Web service.

With the methods created you must now complete the implementation (business logic) of the operations. All the operations are straight forward should not take more that 5-10 minutes.

Deploy and Test the Web Service

Obviously, your Web service needs to be running in order to interact with it so start the service and Java server.

1) Right-click the CalcWebApp project node and select Run.

2) Right-click the CalcWS node in the CalcWebApp project and select Test Web Service.

   Note: You can close the browser and testing window but leave the server running. The Web service needs to be available when creating the Web service client as you will see in the following steps.

Creating the Client Project

1) Choose File → New Project. Select MIDP Application from the Mobility category.

2) Name the project *CalcClient*. Uncheck the box Create Hello MIDlet

3) Click Finish.

Creating the Web Service Client

1) Right-click on the *CalcClient* node in the projects window and choose New → Java ME Web Service Client from the MIDP category. Click Next.

2) Specify the location of the WSDL from the running CalcWebApp service. It should be http://localhost:8080/CalcWebApp/CalcWSService?WSDL if you used the same naming conventions as the lab.

   Note: You can get the actual URL of the WSDL file when you test your Web service. When you test your Web service in the browser there is a link that directs you to the WSDL file. You can copy and paste the URL from the browser.

3) Click Retrieve WSDL. If successful, the rest of the corresponding setup information will be automatically populated. If not successful, then the WSDL file that you specified was either incorrect or your Web service was not running.

4) Click Finish

You should now have an XML configuration file, a WSDL file and a client stub added to you project. You will use the generated stub from your J2ME application to invoke your Web service.

Creating the Client Interface

We want to create a simple GUI for the mobile client to enter simple mathematical expressions that can then be evaluated by the Web service.

1) Right-click on the CalcClient project node and select New → Visual MIDlet from the MIDP category.

2) Name it CalcScreen. Use the default package.

3) Click Finish.

The first thing that you will see is the Flow view of your Visual MIDlet. The Flow view shows the relationship between components of your MIDlet. The other views are the Source view and the Screen view. The next thing that we want to do is add the UI components that will allow us to enter mathematical equations and invoke the Web service.

1) Right-click anywhere on the Flow view and select New/Add → Form. Rename the form title to “Calculator” and the form instance name to “formCalc”.

2) Staying in the Flow view, click and drag the “started” field from the Mobile Device object to the “formCalc” object. This will create an arrow that indicates that when the application starts the first screen will be the Calculator form.

3) Double-click the formCalc object. This will take you to the screen view of the Calculator form.

Now we are going to add two text fields that allow the user to input two numbers to perform calculations on.

4) Right-click on the Calculator form in the Screen view and select New/Add → Text Field. Name the text field instance name “tfNum1” and name the label “Number 1:”.

5) Right-click on the Calculator form in the Screen view and select New/Add → Text Field. Name the text field instance name “tfNum2” and name the label “Number 2:”.

6) Right-click on the Calculator form in the Screen view and select New/Add → Text Field. Name the text field instance name “tfResult” and name the label “Result:”.

Now we are going to add some menu item commands the application.

7) Right-click on the Calculator form in the Screen view and select New/Add → Item Command. Name the text field instance name “icAdd” and name the label “Add”.

8) Right-click on the Calculator form in the Screen view and select New/Add → Item Command. Name the text field instance name “icSubtract” and name the label “Subtract”.

9) Right-click on the Calculator form in the Screen view and select New/Add → Item Command. Name the text field instance name “icMultiply” and name the label “Multiply”.

10) Right-click on the Calculator form in the Screen view and select New/Add → Item Command. Name the text field instance name “icDivide” and name the label “Devide”.

11) Right-click on the Calculator form in the Screen view and select New/Add → Item Command. Name the text field instance name “icExp” and name the label “Exponent”.

Run the J2ME MIDlet to get a feel for the application and what you have done so far. When you launch the CalcClient application you’ll first see the screen with the two number text fields and the result text field. You will also see the five menu items that will perform actions of the numbers by invoking the Web service.

Consuming the Web Service

All that is left is to invoke the Web service when a user selects one of the operations from the menu items. One problem with invoking Web services is that they can cause deadlock in applications if the service does not return. The solution to this is to invoke the service from a thread. The easiest way to do this is to create an inner class for invoking the Web service that implements the Runnable class. The code for this class can be found at the end of this lab. The class is named MathAction and the default contructor takes three parameters; those being a string action command (“add”, “subtract”), the first number, and the second number. Copy the inner class, MathAction.java, into the CalcClient class.

Go to the Source view of CalcClient and find the method named commandAction. This method is mandatory because the class implements the CommandListener. Notice that all

of the menu items that we have already added to the application have already been coded into the commandAction method.

1) For each of the command actions create a new thread that calls on the MathAction constructor. The example code is as follows:

```
Thread th = new Thread(new MathAction("add", "2", "4"));
th.start();
```

Note: Don't forget to replace the parameter values with the appropriate numbers and action string.

Deploy and Test the Client

1) Right-click on the CalcClient node and select Run.

Note: Make sure that the Web service is running or you will get a java.rmi.RemoteException error.

2) Verify that all operations working and that they are returning the correct values.

Conclusion

This lab demonstrated how to work with Web services from a mobile perspective. By now you should have a more practical understanding that Web services are capable of fulfilling demands from various platforms and environments. Although the Web service and application created in this lab were simplified, it is expected that you grasp the various possibilities for which Web services can be used.

MathAction.java

```
public class MathAction implements Runnable {

    String action;
    double num1;
    double num2;

    public MathAction(String action, double num1, double num2){
        this.action = action;
        this.num1 = num1;
        this.num2 = num2;
    }

    public void run(){

        try{
            double result = 0;
            CalcWSService_Stub stub = new CalcWSService_Stub();
            if(action.equals("add")){
                result = stub.add(num1, num2);
            }
            else if(action.endsWith("subtract")){
                result = stub.subtract(num1, num2);
            }
            else if(action.endsWith("multiply")){
                result = stub.multiply(num1, num2);
            }
            else if(action.endsWith("divide")){
                result = stub.divide(num1, num2);
            }
            else if(action.endsWith("exp")){
                result = stub.exp(num1, num2);
            }
            tfResult.setString(Double.toString(result));
        }catch(java.rmi.RemoteException re){
            System.out.println("Math service failed: " +
re.getMessage());
        }

    }

}
```