Deliver Web services to mobile apps

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. Tutorial tips

About this tutorial

This tutorial demonstrates how to access Web services using J2ME-enabled mobile devices and the kSOAP (SOAP for the kVM) library. First, you'll develop a sample stock price Web service using WSDK (IBM WebSphere SDK for Web Services for today's example), and then you'll access this service using a MIDlet deployed in a J2ME environment.

Prerequisites

You must have a knowledge of Web services and J2ME (Java 2 Platform, Micro Edition).

Software requirements

You will need the following software for this tutorial:

1. *IBM WebSphere SDK for Web Services*
2. *J2ME Wireless Toolkit 2.0*
3. *Java 2 Platform, Standard Edition 1.4*
4. *kSOAP*
5. *Sample code*

About the author

Naveen Balani spends most of his time designing and developing J2EE-based products. He has written various articles for IBM on the following topics: Web services, WebSphere MQ, WebSphere Studio, Java wireless devices, DB2 Everyplace for Palm, Java-Nokia, and wireless data synchronization. You can reach Naveen at *naveenbalani@rediffmail.com*.

Section 2. Getting started

Install the software

WSDK

The IBM WebSphere SDK for Web Services (WSDK) provides a runtime environment, simple development tools, and examples to design and execute Web service applications. The WSDK is based on open specifications, such as SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), and UDDI (Universal Description, Discovery, and Integration).

To install the WSDK software, download the *WSDK*, run the setup file, and install the file to a folder called c:\wsdk.

Install the software

The J2ME Wireless Toolkit 2.0 (WTK)

The J2ME WTK provides a development and emulation environment for executing MIDP- (Mobile Information Device Profile) and CLDC-based (Connected Limited Device Configuration) applications. J2ME 2.0 requires that you install JDK 1.4.

Download the *J2ME Wireless Toolkit*, run the setup file, and install the file in a folder called c:\J2mewtk.

Install the software

kSOAP library

The kSOAP (SOAP for the kVM) library provides lightweight APIs that let J2ME MIDP-based devices execute Web services. Download *ksoap-midp.zip* to a folder called c:\ksoap.

Section 3. Introduction to MIDlets

MIDLET basics

A MIDlet is an application written for MIDP. The MIDP specification supports HTTP client capabilities, which allows a MIDlet to access remote services through HTTP. MIDP provides user interface APIs for display purposes, giving applications direct control over the UI -- similar to Java Swing.

MIDlet life cycle

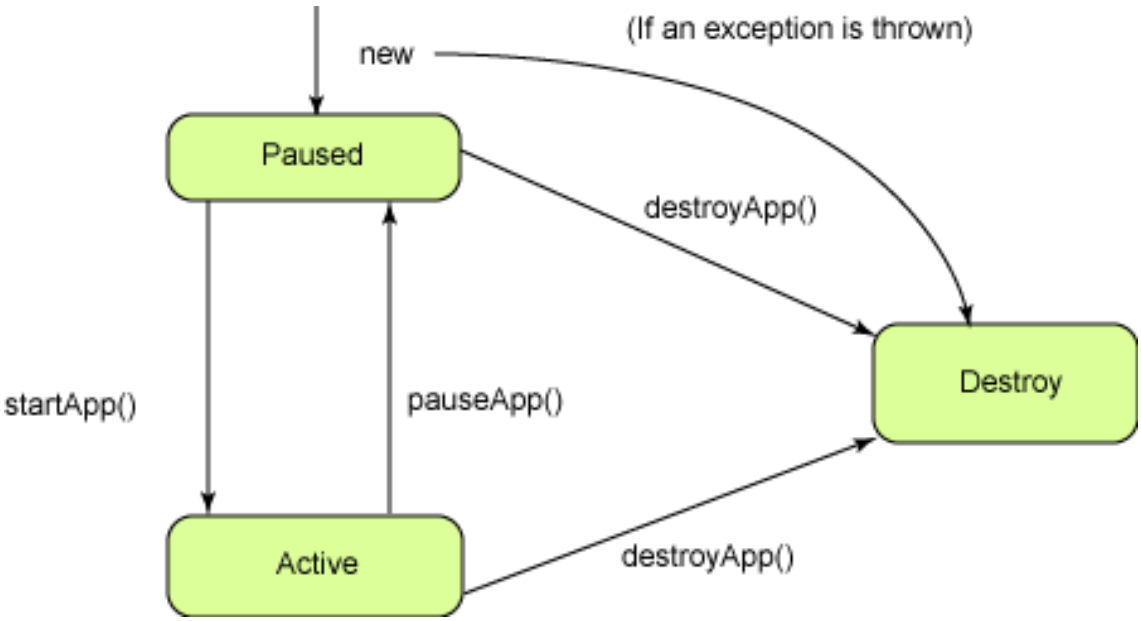
A MIDlet is managed by the Java Application Manager, which executes the MIDlet and controls its life cycle. The MIDlet can be in one of the following states: paused, active, or destroyed.

When you first create and initialize a MIDlet, it is in the paused state. If an exception occurs in the MIDlet's constructor, the MIDlet enters the destroyed state and is discarded.

The MIDlet enters the active state from the paused state when its `startApp()` method call is completed, and the MIDlet can function normally.

The MIDlet can enter the destroyed state upon completion of the `destroyApp(boolean condition)` method. This method releases all held resources and performs any necessary cleanup. If the condition argument is true, the MIDlet will always enter the destroyed state.

Figure 1 illustrates the various states of the MIDlet life cycle.


new
(If an exception is thrown)
Paused
destroyApp()
Destroy
startApp()
pauseApp()
Active
destroyApp()

Section 4. Build and deploy a Web service

Build the Price Web service

WSDK comes with a beta version of the Axis framework, a Java-based, open source implementation of SOAP 1.2 and the SOAP with attachments specification from Apache.

To set up and deploy the Axis framework on WSDK, complete the following steps:

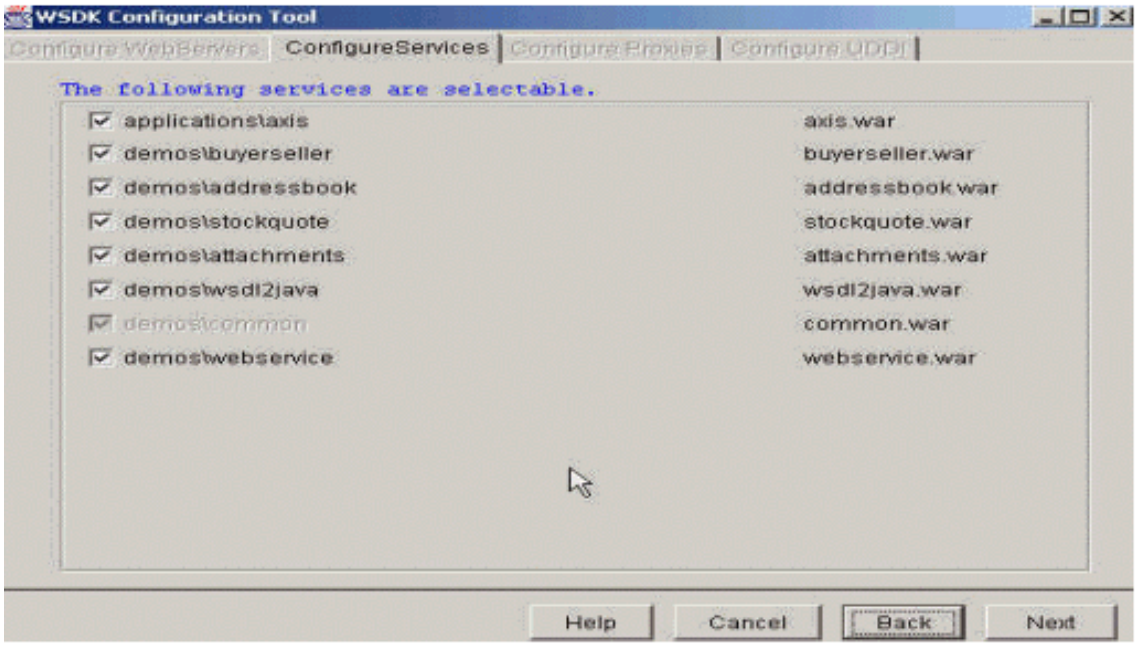
1. Download the *sample-j2me.zip* to the c: drive.
2. Copy the Axis folder from c:\j2web to c:\wsdk\services\applications; WSDK is installed in the c:\wsdk path. Also create an applications folder if one does not exist already.
3. The Axis directory contains the required structure for deploying the Axis framework in WSDK. It contains a `web.xml` file in the WEB-INF directory, which contains the mapping for registering the `org.apache.axis.transport.http.AxisServlet` servlet. Copy the required Axis framework classes from c:\wsdk\lib to the c:\wsdk\services\applications\axis\WEB-INF\lib folder.
4. The deployment folder in c:\wsdk\services\applications\axis\deployment directory contains the `PriceServiceDeploy.wsdd` file, which is the deployment descriptor for deploying our Web service, called `PriceService`. The source code is provided in the c:\j2web\src folder.
5. Finally, our Web service has just one method, `getPrice(string companyName)`, which returns a constant price as a string back to the client. Here is the source code:

```
public class PriceService { public float getPrice(String number)
{ //Return the price return 1500.50f; } }
```

Deploy the Axis framework on WSDK

To deploy the Axis framework on WSDK, follow these steps:

- Open an instance of the Windows command prompt. Change the directory to c:\wsdk\bin.
- Set the path variable to include c:\wsdk\bin and c:\wsdk\sdk\bin, and then run the command `wsdkconfig`.
- The `wsdkconfig` tools start up. Click the Next button to proceed to the Configure Services section, and select the Application\Axis services, as shown in Figure 2.



- Click Next twice, and then click Finish to deploy the Axis framework. You should receive the message: "The application server has been configured for WSDK."

Deploy the Price Web service

To deploy the Price Web service, follow these steps:

1. Start the WebSphere Application server associated with WSDK by navigating to START-> PROGRAMS-> IBM WebSphere SDK for Web Services-> Start App Server.
2. Change the directory to c:\wsdk\bin and execute the following command to deploy the Price Web service: `axisDeploy c:\j2web\axis\deployment\PriceServiceDeploy.wsdd -l http://localhost:80/axis/services/AdminService`, where `c:\j2web\axis\deployment\PriceServiceDeploy.wsdd` is the location of our deployment file and `http://localhost:80/axis/services/AdminService` is the URL where our Axis framework is deployed.
3. After executing the above command, you will receive this message:

```
"- Processing file
c:\j2web\axis\deployment\PriceServiceDeploy.wsdd"<Admin>Done
processing</Admin>
```

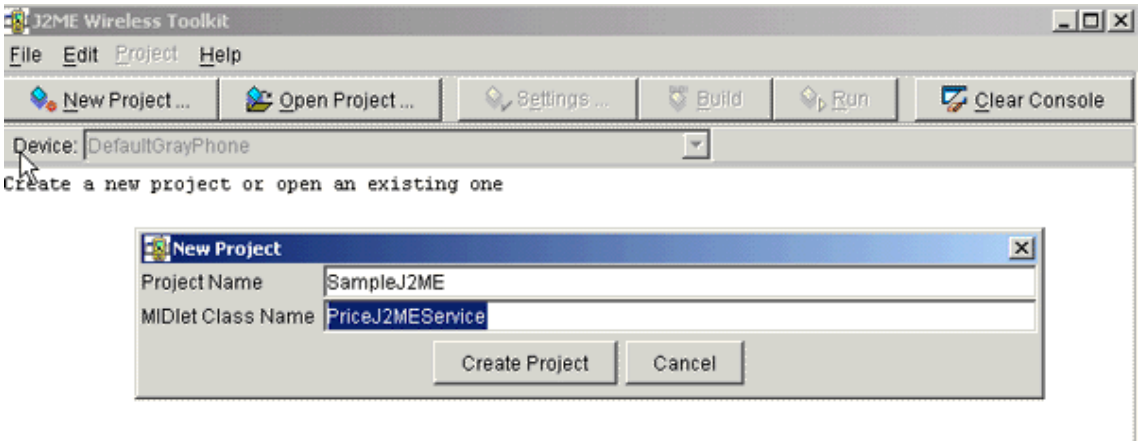
Finally, we have deployed our Price Web Service. Next, we'll access this service using the J2ME MIDP emulator.

Section 5. Create, deploy, and analyze the MIDlet

Create the MIDlet

First, we'll create a new MIDP project in the J2ME toolkit. Follow these steps:

- Navigate to START-> PROGRAMS-> J2ME Wireless Toolkit 2.0 Beta-> Ktoolbar. The toolkit window will open.
- Create a new project using Ktoolbar, as shown in Figure 3. Type `SampleJ2ME` as the project name and `PriceJ2MEService` as the MIDlet class name. Click on Create Project.



- Upon clicking Create Project, a pop-up window will appear. Click OK.
- Copy the source file `PriceJ2MEService.java` from c:\j2web\src to c:\J2mewtk\apps\SampleJ2ME\src; the J2ME Wireless Toolkit is installed in the c:\J2mewtk path. `PriceJ2MEService.java` is a MIDlet application that uses the kSOAP library for connecting to our Price Web service.
- Copy the kSOAP library, `ksoap-midp.zip`, from c:\ksoap (the org folder) to the c:\J2mewtk\apps\lib folder so that `PriceJ2MEService.java` can find the kSOAP library.
- Click on the Build button on the KtoolBar; you will receive the following message:

```
Building "SampleJ2ME" [Output directory for verified classes:
c:\winnt\temp\tmp2] [Output directory for verified classes:
c:\J2mewtk\apps\SampleJ2ME\classes] Build complete
```

Congratulations, you have successfully built your `PriceJ2MEService.java` MIDlet.

Analyze the MIDlet code

In this section, we'll analyze the `PriceJ2MEService.java` code in some detail. I've listed line numbers below for better understanding; however, they don't match the line numbers of the actual source code:

```
Line 1: public class PriceJ2MEService extends MIDlet implements
CommandListener { Line 2: Form mainForm = new Form
("PriceService"); Line 3: TextField symbolField = new TextField
("GETPrice", "IBM", 5, TextField.ANY); Line 4: StringItem
resultItem = new StringItem ("", ""); Line 5: Command getCommand
= new Command ("Get", Command.SCREEN, 1); Line 6: public
PriceJ2MEService () { Line 7: mainForm.append (symbolField);
Line 8 :mainForm.append (resultItem); Line 9:
mainForm.addCommand (getCommand); Line 10:
mainForm.setCommandListener (this);
```

Line 1 defines our `PriceJ2MEService` class, which extends the MIDlet class and implements `CommandListener` for capturing events.

Lines 2 through 10 define UI elements and control elements for capturing user input.

Analyze the MIDlet code

Lines 11, 12, and 13 below provide the MIDlet life cycle methods, as discussed earlier. Every MIDlet class should provide these life cycle methods.

```
Line 11: public void startApp () { Display.getDisplay
(this).setCurrent (mainForm); } Line 12: public void pauseApp ()
{ } Line 13: public void destroyApp (boolean unconditional) { }
```

Analyze the MIDlet code

Here are the final lines of the MIDlet code:

```
Line 14: public void commandAction (Command c, Displayable d) {
String symbol = symbolField.getString (); resultItem.setLabel
(symbol + ": "); Line 15: SoapObject rpc = new SoapObject
"urn:PriceService", "getPrice"); rpc.addProperty ("price",
symbol); resultItem.setText (""+new HttpTransport
("http://localhost:80/axis/services/PriceService",
"uurn:PriceService#getPrice").call (rpc));
```

In line 14, we define the `commandAction()` method, which captures user events. This method gets the string entered by the user and calls the Web service.

In line 15, we create a `SoapObject` constructor, passing to it the URN of the service name, `PriceService`, and the method name to invoke, i.e. `getPrice()`.

We then create an `HttpTransport` object, which calls the SOAP service, retrieves the value (a constant string 1,500 from our Price Web service), and displays it back to

the user.

Section 6. Run the application

Run the application

First, start the WSDK application server by navigating to START-> PROGRAMS-> IBM WebSphere SDK for Web Services-> Start App Server. Then click on the Run button on the KtoolBar. The default emulator shown in Figure 4 will appear.

Figure 4. The J2ME MIDlet emulator

Run the application

Now launch the application. You should see the screen shown in the figure below.

Type a value in the IBM text field, and then click on the Launch button to get the stock price.

You should receive the screen shown below, which displays the stock price.

Thus, we have successfully created and deployed our Web service, calling it from a J2ME MIDP-enabled device.

Section 7. Wrap-up

Apply your knowledge

In this tutorial, we developed and deployed our sample Price Web service using WSDK. We then accessed this service, built using a MIDlet, and deployed it in a J2ME environment. You can apply this knowledge further to deliver any realtime data using Web services on J2ME-enabled devices.

Resources

- Browse the official site for *kSOAP*. Here you will find the kSOAP source code, as well as compiled binaries for both UNIX and Windows.
- At *java.sun.com*, you will find the latest installation of the Java Standard Development Kit (SDK) and J2ME for all platforms.
- Read more Web services articles in the *IBM developerWorks Web Services Zone*.
- Part 2 of Naveen Balani's *"Web Architecture Using MVC Style"* provides a procedure to transform any MVC implementation into Web services.

Section 8. Feedback

Feedback

Please send us your feedback on this tutorial. We look forward to hearing from you!

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial Building tutorials with the Toot-O-Matic demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11. We'd love to know what you think about the tool.