
Using kXML to access XML files on J2ME devices

Skill Level: Introductory

[Naveen Balani](#)
Author

23 Sep 2003

This tutorial will detail the use of kXML 2, a small XML pull parser specially designed for constrained environments, to access, parse, and display XML files for Java 2 Micro Edition-enabled devices. This tutorial will demonstrate how to build a mobile application that brings XML data to wireless J2ME devices, and will instruct the reader in how to craft a MIDlet that performs the necessary logic and deploys it to a J2ME environment.

Section 1. Before you start

About this tutorial

This tutorial details the use of kXML 2r2.1.8, a small XML pull parser specially designed for constrained environments, to access, parse, and display XML files for Java 2 Micro Edition (J2ME)-enabled devices. This kXML version is based on the common XML pull API and is BSD-licensed. This tutorial demonstrates how to build a mobile application that brings XML data to wireless J2ME devices, and instructs you on how to craft a MIDlet that performs the necessary logic and deploys it to a J2ME environment.

Prerequisites

To use the information in this tutorial, you should have a good working knowledge of J2ME and XML. Before you start working, you will need the following downloads:

- The [J2ME Wireless Toolkit 2.0](http://java.sun.com/products/j2mewtoolkit/download-2_0.html), which can be found at http://java.sun.com/products/j2mewtoolkit/download-2_0.html.
 - The [Java SDK 1.4.1](http://java.sun.com/j2se/1.4.1/download.html) (Java Software Developers Kit), available at <http://java.sun.com/j2se/1.4.1/download.html>.
 - The [kXML 2r2.1.8](http://kobjects.dyndns.org/kobjects/auto?self=$dikgikg2) ([http://kobjects.dyndns.org/kobjects/auto?self=\\$dikgikg2](http://kobjects.dyndns.org/kobjects/auto?self=$dikgikg2)).
 - The sample code (see the [Downloads](#) section) that accompanies this article.
-

Section 2. Getting started

Installing the software

The J2ME Wireless Toolkit 2.0 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 the Java SDK.

After you download the J2ME Wireless Toolkit, run the setup file and install the file in a folder called c:\WTK20.

The kXML library

kXML 2 (revision 2.1.8) is a small XML pull parser, specially designed for constrained environments such as Applets, Personal Java, or MIDP devices.

Download [kxml2-midp.zip](#) (from K Objects) to a folder called c:\ksoap.

As stated on xmlpull.org, the Common API for XML Pull Parsing (XmlPull) is an effort to define a simple and elegant pull parsing API that will provide a standardized method to do XML pull parsing from J2ME to J2EE. It is a minimal API, one that is easy to implement as a stand-alone API or on top of an existing parser. XmlPull allows both fast, high-level iteration (using `next()` method) and low-level tokenizing (using the `nextToken()` token). It is designed for easy building on top of SAX, XML pull parsers that use iterators with event objects, and even DOM implementations.

Section 3. MIDlets basics

What are MIDlets?

A MIDlet is an application written for MIDP, the Mobile Information Device Profile, part of the Java Runtime Environment (JRE) for such mobile information devices as phones and PDAs. MIDP provides such core application functionality required by mobile applications as the user interface, network connectivity, local data storage, and application life cycle management. It is packaged as a standardized JRE and set of Java APIs.

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 user interface -- similar to Java Swing.

The 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.

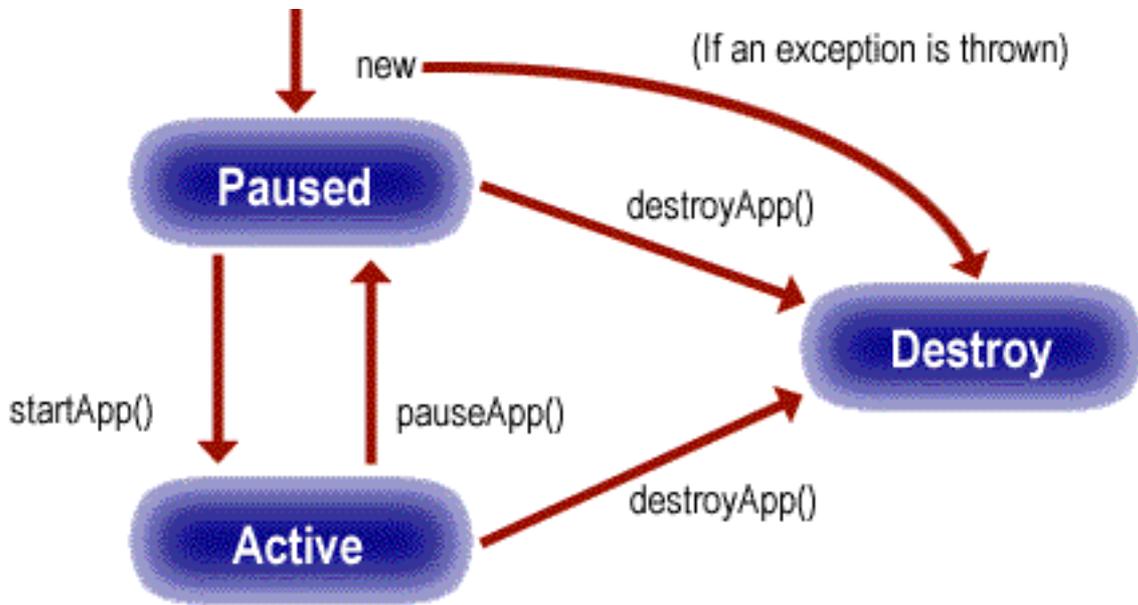
When you first create and initialize a MIDlet, it is in the paused state. If an exception occurs in the MIDlet's constructor, the MIDlet enters the destroyed state and is discarded.

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 always enters the destroyed state.

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

Figure 1. The MIDlet life cycle



Section 4. Analyzing and deploying the XML application

Analyzing the XML

My XML application is a sample XML file representing a book catalog. In a normal scenario, a Java servlet processes an incoming query from the J2ME device (such as the book's ISBN number), performs a lookup on the database, and generates an XML response to the client.

The client might be a browser, which renders the display by applying style sheets. Or, it might be a J2ME device, which can parse the data and display the contents.

In this tutorial, assume that the following XML file, the book catalog sample, is already available.

```

<?xml
version="1.0"?>
<catalog>
<title>
<name>EJB
2</name>
<description>EJB
2
  
```

```
Fundamentals</description>
<author>Jason</author>
<rating>4</rating>
<available>Yes</available>
</title>
<title>
<name>Applied
XML</name>
<description>Advanced
XML Parsing
Programming</description>
<author>Jason</author>
<rating>5</rating>
<available>Yes</available>
</title>
</catalog>
```

Deploying the XML application

To deploy the book catalog sample XML file, perform the following:

1. Extract the xml-j2me.zip file to any directory (for example, c:\. A directory c:\xmlj2me is created.
2. Copy the XML file from c:\xmlj2me\xml, and place the XML file in the Web directory of your application server.
3. In case of tomcat, if you assume that it's installed at c:\tomcat, then place the XML file in the examples Web directory (c:\tomcat\webapps\examples).
4. Start the application server.

Your XML file can be now accessed at <http://localhost:8080/examples/book.xml>.

Section 5. Creating the MIDlet

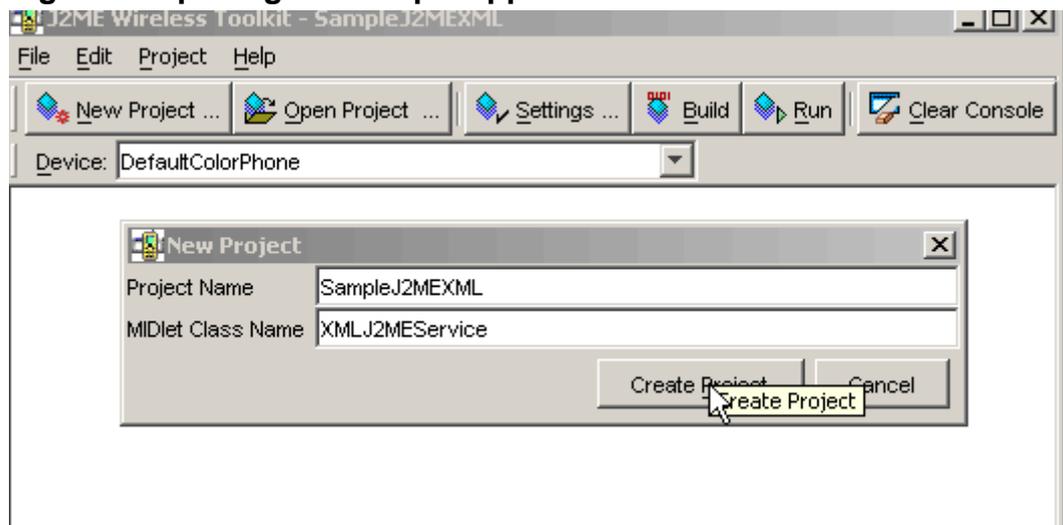
Creating the MIDlet

First, create a new MIDP project in the J2ME toolkit by:

1. Selecting **START > PROGRAMS > J2ME Wireless Toolkit 2.0 > Ktoolbar**. The toolkit window opens.

2. Creating a new project using Ktoolbar, as shown in the figure below. Enter **SampleJ2MEXML** as the project name and **XMLJ2MService** as the MIDlet class name. Click **Create Project**.

Figure 2. Opening the sample application



3. Click **OK** when the pop-up window appears.
4. Copy the source file XMLJ2MService.java from c:\xmlj2me\src to c:\wtk20\apps\SampleJ2MEXML\src; the J2ME Wireless Toolkit is installed in the c:\wtk20 path.
5. Click **Build** on the Ktoolbar. You receive the following message:

```
Project settings saved
Building "SampleJ2MEXML"
Build complete
```

Congratulations, you have successfully built your XMLJ2MService.java MIDlet.

Section 6. Analyzing the MIDlet code

Analyzing the first 10 lines

In this section, I'll analyze the `XMLJ2MIService.java` code in detail. I've listed line numbers in the code reproduced here to make it easier to follow; these line numbers do not match the line numbers of the actual source code.

```
Line 1: public class XMLJ2MIService extends MIDlet implements CommandListener {  
Line 2: Form mainForm = new Form ("SampleJ2MEXML");  
Line 3: static final String URL = "http://localhost:8080/examples/book.xml";  
Line 4: Vector bookVector = new Vector();  
Line 5: StringItem resultItem = new StringItem ("", "");  
Line 6: private final static Command xmlCommand = new Command("Get XML Data",  
    Command.OK,1);  
Line 7: public XMLJ2MIService () {  
Line 8:     mainForm.append (resultItem);  
Line 9:     mainForm.addCommand (xmlCommand);  
Line 10: mainForm.setCommandListener (this) };
```

In the listing, Line 1 defines the `XMLJ2MIService` class, which extends the `MIDlet` class and implements `CommandListener` for capturing events.

Lines 2 through 10 define UI elements, location of XML files, and control elements for capturing user input.

Analyzing lines 11-13

Lines 11, 12, and 13 provide the MIDlet life cycle methods, as I discussed earlier. Every MIDlet class needs to provide these life cycle methods.

```
Line 11: public void startApp () { Display.getDisplay (this).setCurrent (mainForm);
new ReadXML().start(); }

Line 12: public void pauseApp () { }

Line 13: public void destroyApp (boolean unconditional) { }
```

Line 12 executes the `run` method of `ReadXML` class, which parses the XML file. Normally it's a good performance practice to execute HTTP connections and I/O operations in separate threads rather than performing a `commandAction` method.

Analyzing lines 14-25

```
Line 14 : class ReadXML extends Thread {
Line 15 : public void run() {
Line 16 : HttpURLConnection httpConnection = (HttpURLConnection) Connector.open(URL);
Line 17 : KXmlParser parser = new KXmlParser();
Line 18 : parser.setInput(new InputStreamReader(httpConnection.openInputStream()));
Line 19 : parser.nextTag();
Line 20 : parser.require(XmlPullParser.START_TAG, null, "catalog");
Line 21 : while (parser.nextTag () != XmlPullParser.END_TAG)
Line 22 :     readXMLData(parser);
Line 23 : parser.require(XmlPullParser.END_TAG, null, "catalog");
Line 24 : parser.next();
Line 25 : parser.require(XmlPullParser.END_DOCUMENT, null, null);
```

Lines 14 to 18 open an HTTP connection to retrieve the XML file and initialize `KXmlParser`.

From Line 20, I start parsing the XML file; I start with the root element `catalog` and continue processing the XML file until I reach Line 23, where the root element `catalog` ends. I perform similar processing in `readXMLData(parser)`, where I parse the child elements of `catalog` and store it in the `Book` object for later display.

Analyzing lines 26-40

```
Line 26 : public void commandAction(Command c, Displayable d) {
Line 27 :
Line 28 : StringBuffer sb = new StringBuffer();
Line 29 : if (c == xmlCommand) {
Line 30 :     for(int i = 0 ; i < bookVector.size() ; i++){
Line 31 :         Book book = (Book) bookVector.elementAt(i);
Line 32 :         sb.append("\n");
Line 33 :         sb.append("Name : ");
Line 34 :         sb.append(book.getName());
Line 35 :         sb.append("\n");
Line 36 :         sb.append("Description : ");
Line 37 :         sb.append(book.getDescription());
Line 38 :         sb.append("\n"); }
Line 39 :     resultItem.setLabel("Book Information");
Line 40 :     resultItem.setText(sb.toString());
```

Lines 26 through 40 retrieve the `Book` objects from the vector and display it in the `resultItem` text field when the user clicks the **Get XML Data** command button.

Now that I'm finished analyzing the code, I'll show the code in action as I run the application.

Section 7. Running the XHTML application

Launching the application

To run the sample `SampleJ2MEXML` application, click **Run** on the Ktoolbar. The default emulator shown in the following figure appears.

Now launch the application. You should see the screen as shown in Figure 3.

Figure 3. Opening the sample application



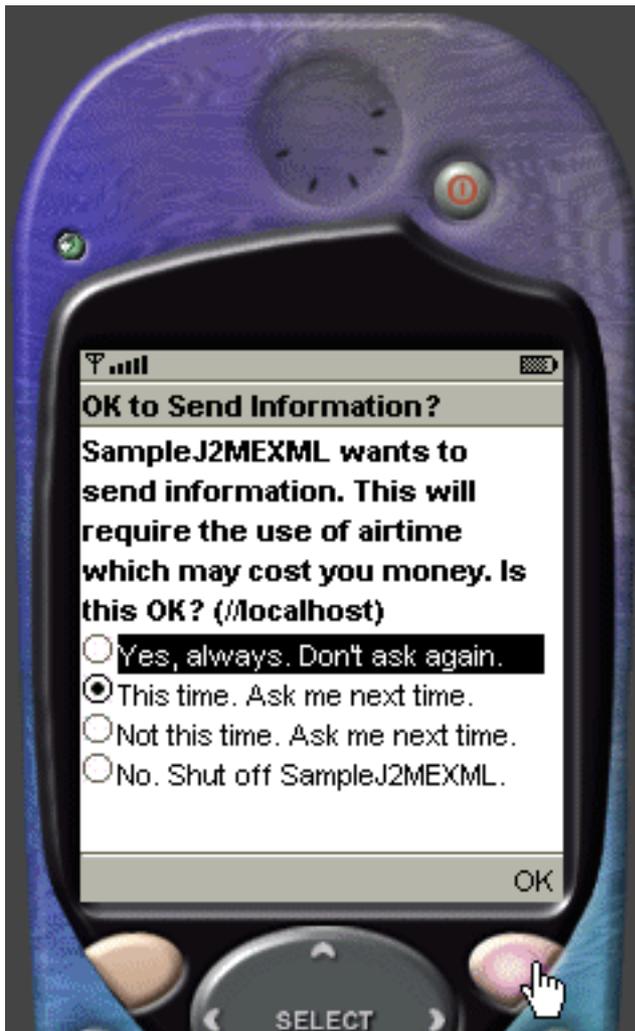
Running the application

On launching the application, you should receive the following message:

```
SampleJ2MEXML wants to send information. This will require the use of airtime  
which may cost you money. Is this OK? (//localhost)
```

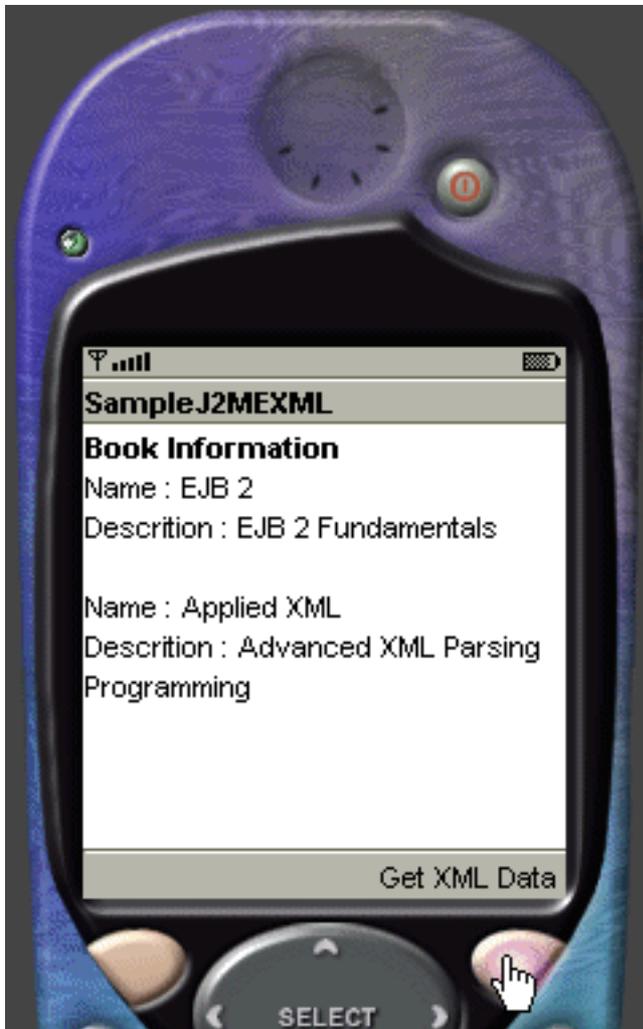
Click on default value to proceed.

Figure 4. Default message



Click **Get XML Data** and the following screen that displays the parsed XML data appears.

Figure 5. The parsed book catalog



Section 8. Summary and resources

Summary

This tutorial offered direction on ways to deliver complex XML-based applications to mobile devices. I've demonstrated how to retrieve and parse a sample XML application to a J2ME-compatible device using kXML 2r2.1.8, a small XML pull parser specially designed for constrained environments, based on the common XML pull API, and BSD-licensed. The tutorial also showed how to build a mobile application that brings XML data to wireless J2ME devices, as well as how to craft a

MIDlet that performs the necessary logic and deploy it to a J2ME environment.

Downloads

Description	Name	Size	Download method
Code sample	wi-xmlj2me.zip	3 KB	HTTP

[Information about download methods](#)

Resources

Learn

- Stay current with [developerWorks technical events and Webcasts](#).
- Visit [Java.sun.com](#) for the latest version of these Java products:
 - The [J2ME Wireless Toolkit 2.0](#).
 - The [Java SDK 1.4.1](#) (Java Software Developers Kit).
- You can find [kXML 2r2.1.8](#) here.
- The [Web Services Tool Kit for Mobile Devices](#) provides tools and run-time environments that allow development of applications that use Web Services on small mobile devices, gateway devices, and intelligent controllers.
- Find out more information about the [Common API for XML Pull Parsing](#).

Get products and technologies

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

Discuss

- Participate in [developerWorks blogs](#) and get involved in the developerWorks community.

About the author

Naveen Balani

Naveen Balani spends most of his time designing and developing J2EE-based products. He has written various articles and white papers for IBM *developerWorks*, covering topics such as Web services, JMS, AXIS, MQSeries, WebSphere Studio, Java wireless devices, DB2 Everyplace for Palm, Java on EPOC, and wireless data synchronization. You can reach him at naveenbalani@rediffmail.com.