Mobile database access

Skill Level: Introductory

Naveen Balani
Freelance Writer

09 Mar 2004

This tutorial shows you how to build a database application using J2ME Record Management System. You will also learn how to craft a MIDlet that performs the necessary logic of creating and accessing the DB2 database application and deploys it to a J2ME environment.

Section 1. Before you start

About this tutorial

This tutorial shows you how to build a database application using Java 2 Micro Edition (J2ME) Record Management System (RMS). It also demonstrates how to craft a MIDlet that performs the necessary logic of creating and accessing a sample database application and deploys it to a J2ME environment.

Products such as DB2® Everyplace use RMS as the underlying record store for storing application data and then synchronizing it with the remote database store.

Prerequisites

To fully appreciate the information in this tutorial, you should have a good working knowledge of J2ME. Before you start working, make sure you have the following downloads:

- The J2ME Wireless Toolkit 2.0 (http://java.sun.com/products/j2mewtoolkit/download-2_0.html).
- The Java SDK 1.4.1 (Java Software Developers Kit) (http://java.sun.com/j2se/index.jsp).

- The sample code that accompanies this article.

The installation of all of these components is discussed in Installing software.

Section 2. Installing software

The J2ME Wireless Toolkit

The J2ME Wireless Toolkit 2.0 provides a development and emulation environment for executing MIDP- and CLDC (Connected Limited Device Configuration) -based 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 toolkit's files into a folder called C:\WTK20.

The sample code package

Download wi-mob-j2me.zip (see the Downloads section) and unzip to it any location; for my example, I'll just use C:\. The J2meMob.java file in the C:\j2memob\src directory contains the complete source code for the application.

Section 3. MIDlet basics

What are MIDlets?

A *MIDlet* is an application written for the Mobile Information Device Profile (MIDP), which is part of the Java Runtime Environment (JRE) for mobile information devices such as phones and PDAs. MIDP provides the core application functionality required by mobile applications, including the user interface, network connectivity, local data storage, and application life cycle management. It is packaged as a standardized Java Runtime Environment 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. A MIDlet can be in one of three states: *paused, active,* or *destroyed.*

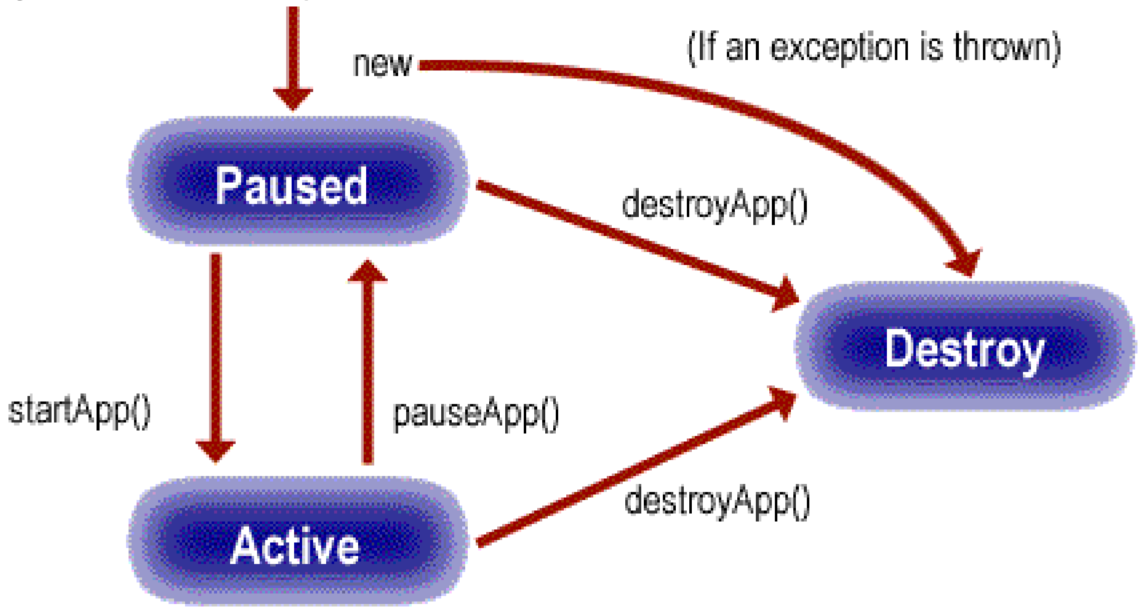
When first created and initialized, a MIDlet is in the paused state. If an exception occurs in the MIDlet's constructor, the MIDlet enters the destroyed state and is discarded.

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. MIDlet life cycle



The sample application shows all of these processes in action, and is covered in Life cycle methods.

J2ME record management system

The J2ME *record management system (RMS)* provides a mechanism through which MIDlets can persistently store data and retrieve it later. In a record-oriented approach, J2ME RMS comprises multiple record stores.

The record store is created in platform-dependent locations, such as nonvolatile device memory, which are not directly exposed to the MIDlets. The RMS classes call into the platform-specific native code that uses the standard OS data manager functions to perform the actual database operations. Each record in a record store is an array of bytes and has a unique integer identifier.

The `javax.microedition.rms.RecordStore` class represents an RMS record store. It provides various methods with which to manage as well to insert, update, and delete records in a record store.

Section 4. The MIDlet application: Code review

Introduction

This tutorial's sample application is a front end that interacts with an application database.

The database application represents a simple address book application where the user can search for an address in the database based on specified criteria. To simplify matters, the user can search by first name only, and only one record is retrieved by each query. (If there are two records with the same first name, the application returns the first one it encounters; more complex functionality is left as an exercise for you.)

Defining user and database interfaces

Take a look at the beginning of the J2meMob.java file included with this tutorial. The line numbers I'll use in this section are only for reference in my discussion; these numbers do not match up with those in the actual source code, given that this isn't a complete listing.

```
Line 1: public class J2meMob extends MIDlet implements CommandListener {

Line 2: Form mainForm = new Form ("J2MEMobApp");

Line 3: TextField symbolField = new TextField ("Search By Name", "", 5,
        TextField.ANY); StringItem resultFName = new StringItem ("", "");

Line 4:  RecordStore recStore;

Line 5:  private static final Command okCmd = new Command("OK",
         Command.OK, 1);


Line 6:  public J2meMob () {

Line 7:  createDatabase();
Line 8:   mainForm.append (symbolField);

Line 9:  mainForm.addCommand (okCmd);
         mainForm.setCommandListener (this);};
```

In the listing, Line 1 defines the `J2meMob` class, which extends the `MIDlet` class and implements `CommandListener` for capturing events.

Lines 2 through 10 define UI elements, RecordStore class for interacting with Record Management System, and control elements for capturing user input. The next section discusses the `createDatabase()` method in more detail.

The createDatabase() method

The `createDatabase()` method calls two subsequent methods. The `connect()` method uses the `RecordStore.openRecordStore("Address", true )` method to create an address record store, which represents the address database. The code is listed below.

```
   recStore = RecordStore.openRecordStore("Address", true );
```

The `populateData()` method populates the address record store with sample data and uses the `RecordStore.addrecord()` method to commit the sequence of data represented as a sequence of bytes, as shown below.

```
            int recordID = 0;
            ByteArrayOutputStream bytstream = new ByteArrayOutputStream();
        DataOutputStream writer = new DataOutputStream(bytstream);
        writer.writeUTF("Leo");
        writer.writeUTF("Fernandes");
        writer.writeUTF("1500 Dec Road,UC,CA 94545");
        writer.writeUTF("5107776666");
        writer.flush();

        byte[] rec = bytstream.toByteArray();
        recordID = recStore.addRecord(rec,0,rec.length);
        writer.flush();
        bytstream.reset();

        //Second Record

        writer.writeUTF("Raj");
        writer.writeUTF("Malhotra");
        writer.writeUTF("1501 Dec Road,UC,CA 94545");
        writer.writeUTF("5107775454");
        writer.flush();

        rec = bytstream.toByteArray();
        recordID = recStore.addRecord(rec,0,rec.length);
        System.out.println("recordID"+recordID);
        writer.close();
        bytstream.close();
```

Life cycle methods

Lines 11 through 13 provide the MIDlet life cycle methods that I discussed in The MIDlet life cycle. Every MIDlet class needs to provide these methods.

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

Line 12: public void pauseApp () { }

Line 13: public void destroyApp (boolean unconditional) {

 }
```

Retrieving information from the database

Lines 14 through 21 retrieve the information that the user wants from the address table.

```
Line 14 : public void commandAction(Command c, Displayable d) {

Line 15 :if(c == okCmd){
         String symbol = symbolField.getString();


Line 16 : Vector results = null;

Line 17 : results = fetchData(symbol);

Line 18 :   if(results.size() > 0) {
          resultFName.setLabel("First Name");
          resultFName.setText(rs.getString(2));...

Line 19 : }

Line 20 : private ResultSet fetchData(String data){

Line 21 :  ByteArrayInputStream stream;
           DataInputStream reader;
           recStore = RecordStore.openRecordStore("Address", true );
           String fname;

           for (int i = 1; i < = recStore.getNumRecords()
                                 || records.size() > 0; i++) {
                 byte[] rec = new byte[recStore.getRecordSize(i)];
                 rec = recStore.getRecord(i);
                 stream = new ByteArrayInputStream(rec);
                    reader = new DataInputStream(stream);
                    fname = reader.readUTF();

                    if(fname.equals(data)){

                       records.addElement(fname);
                    records.addElement(reader.readUTF());
                    records.addElement(reader.readUTF());
                    records.addElement(reader.readUTF());
                    }
```

The `fetchData()` method at line 17 searches the address record store based on the first name entered by the user and returns the result set in a `Vector`. The `fetchData()` method gets each record based on the unique record identifier, loops through each of the address records, and compares it with the first name entered by the user to find a match. At line 18, I loop through the `Vector` and display the result set values to the user.

Now that you've seen how the code works, see it in action as the application runs on an emulator.

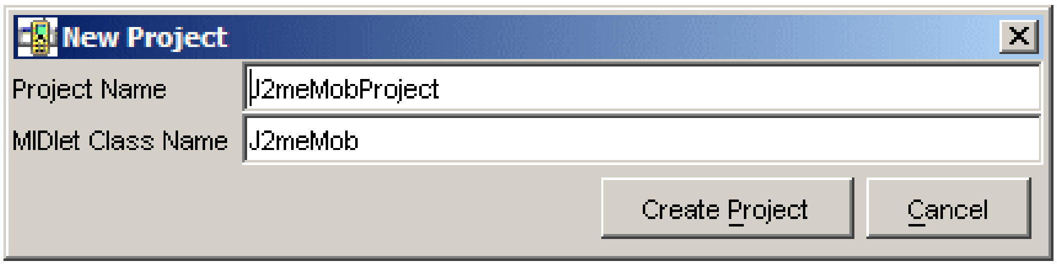
Section 5. The code in action

Creating the MIDlet

Before you can run the application, you need to create a new MIDP project in the J2ME toolkit. To create the new MIDP project:

1. Navigate to **Start -> Programs -> J2ME Wireless Toolkit 2.0 -> Ktoolbar.** The toolkit window opens.
2. Create a new project using Ktoolbar. Enter **J2meMobProject** as the project name and **J2meMob** as the MIDlet class name. Click on **Create Project.**

Figure 2. Opening the sample application



Project Name: J2meMobProject
MIDlet Class Name: J2meMob

3. The next window gives you the opportunity to specify the project settings. Click **OK** to accept the defaults.

4. Copy the source file, J2meMob.java, from the C:\j2memob\src directory to the C:\wtk20\apps\J2meMobProject\src directory. (Remember, the J2ME Wireless Toolkit is installed in the C:\wtk20 path.)

5. Click **Build** on the Ktoolbar. The following message displays:

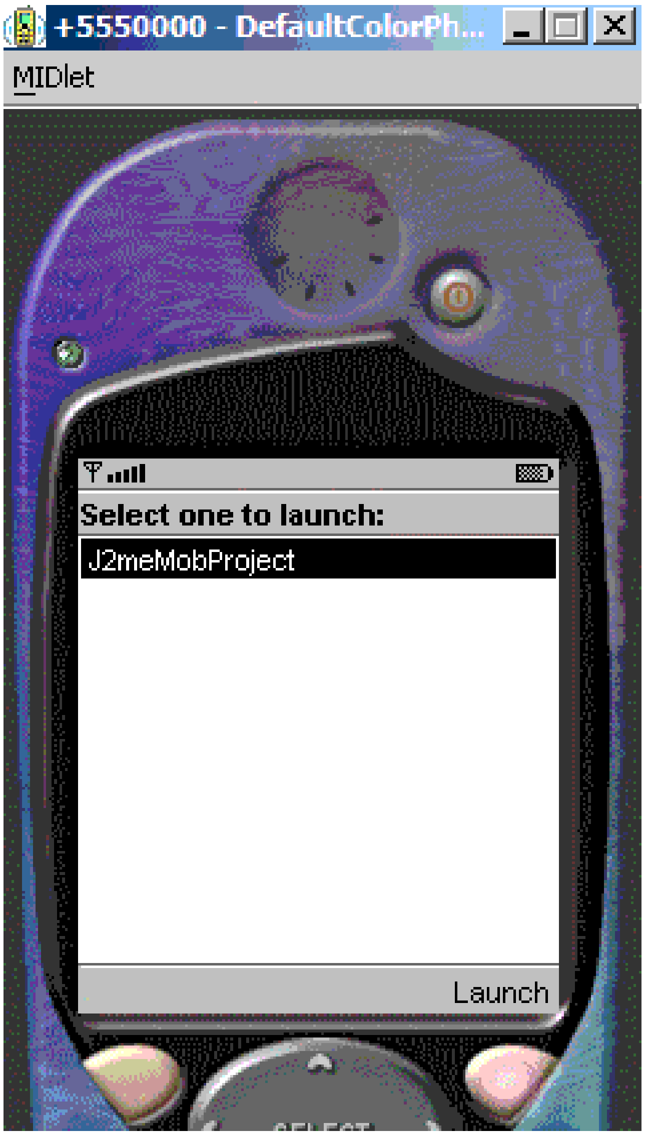
```
Project settings saved
Building "J2meMobProject"
Build complete
```

Congratulations. You have successfully built your `J2meMob.java` MIDlet.

Launching the application

To run the sample J2meMobProject application, click **Run** on the Ktoolbar. The default emulator shown in Figure 3 appears.

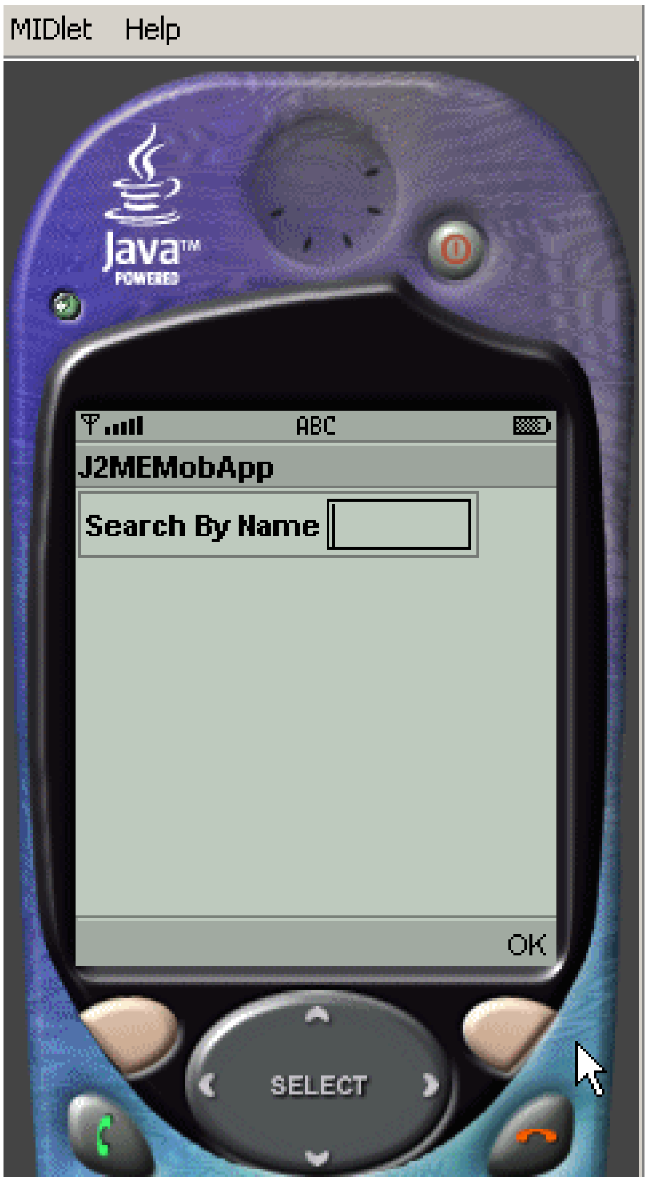
Figure 3. Default emulator



Running the application

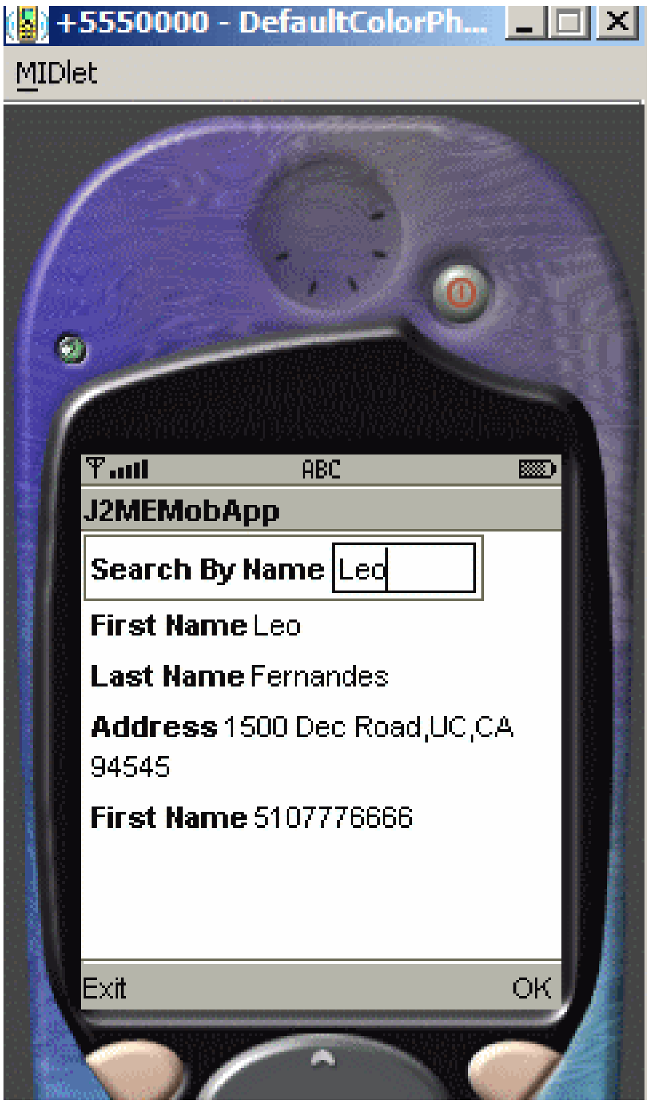
When the application is launched you should see the following screen:

Figure 4. Application ready for data input



Enter the name Leo and click **OK.** You should see address information displayed, as shown in Figure 5.

Figure 5. The selected address data



J2MEMobApp

Search By Name Leo

First Name Leo

Last Name Fernandes

Address 1500 Dec Road,UC,CA 94545

First Name 5107776666

Section 6. Summary

Summary

This tutorial offered direction on how to build a mobile database application using J2ME RMS. With what you've learned here, you could also build applications that synchronize local mobile data with a remote database store using products like DB2 Everyplace. Based on business needs, any complex mobile application can be developed, which you can work with and later synchronize with any remote data store to keep data available throughout the enterprise.

Resources

Learn

- Stay current with developerWorks technical events and Webcasts.
- Get more information on wireless solutions in the developerWorks Wireless technology zone.
- Visit Java.sun.com for the latest version of these Java products:
  - J2ME Wireless Toolkit 2.0
  - Java SDK 1.4.1 (Java Software Developers Kit)
- 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.

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 whitepapers for IBM *developerWorks*, covering topics such as Web services, SOA, CICS®, JMS, Axis, MQSeries®, WebSphere® Studio, Java wireless devices, DB2 Everyplace for Palm, .Net, Java on EPOC, and wireless data synchronization. You can reach him at naveenbalani@rediffmail.com.