Build a smart J2ME mobile application, Part 1

Skill Level: Introductory

Naveen Balani
Author

05 Apr 2005

This two-part tutorial shows you how to build a mobile database application using the J2ME Record Management System and how to synchronize it with a remote Cloudscape database. You also learn how to craft a MIDlet that performs the necessary logic to create and access the database application and deploy it to a J2ME environment.

Section 1. Before you start

About this tutorial

This tutorial is the first in a two-part series designed as a step-by-step guide to building a smart J2ME mobile application. In this tutorial you learn how to build a simple mobile application for order placement. The example application uses the Java 2 Platform, Micro Edition (J2ME) record management system (J2ME RMS) to store order information and a MIDlet to perform the necessary logic of creating and accessing the database application. You learn how to work with the J2ME RMS, craft a MIDlet, and deploy the resulting application to a J2ME environment.

In the next tutorial in this series, you will write your own two-way synchronization logic to synchronize the order information stored in the J2ME RMS with a remote Cloudscape database. The mobile user will then be able to track the status of his or her order by accessing the remote Cloudscape database.

Prerequisites

© Copyright IBM Corporation 2005. All rights reserved.

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

- The Java Software Developers Kit (SDK) 1.4.1.
- The J2ME Wireless Toolkit Version 2.2.
- The sample code that accompanies this article.

I'll walk you through the installation of the latter two components in the section on Installing the software .

Section 2. Installing the software

The J2ME Wireless Toolkit

The J2ME Wireless Toolkit 2.2 provides a development and emulation environment for executing MIDP and Connected Limited Device Configuration (CLDC)-based applications. J2ME 2.2 requires that you also install the Java SDK.

After you download the J2ME Wireless Toolkit, run the setup file and install the toolkit's files in a folder called C:\WTK22.

The sample code package

Download the source code and unpack it to any location you like; for the sake of this example I'll use C:\. The J2meMob.java file in C:\j2mesync\src contains the complete source code for the example application.

Section 3. MIDlets basics

What are MIDlets?

A *MIDlet* is an application written for the Mobile Information Device Profile (MIDP),

© Copyright IBM Corporation 2005. All rights reserved.

which is part of the Java Runtime Environment (JRE) for mobile information devices such as cellular phones and PDAs. MIDP provides the core application functions required by mobile applications, such 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 allow 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.

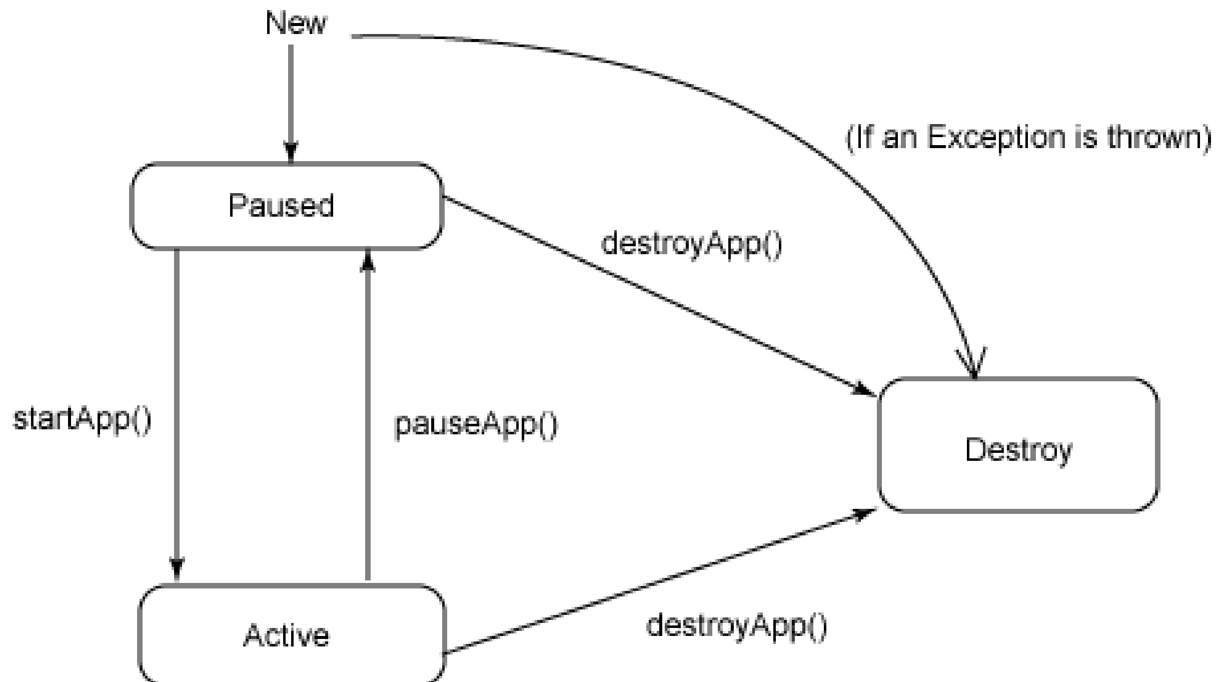
The MIDlet enters the active state from the paused state when its `startApp()` method call is completed, and the MIDlet can function normally.

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. The MIDlet life cycle

© Copyright IBM Corporation 2005. All rights reserved.



You'll see all of these processes at work in the example application in the tutorial section on Life cycle methods.

The 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 operating system 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 to manage as well to insert, update, and delete records in a record store.

Section 4. The MIDlet database application in code

© Copyright IBM Corporation 2005. All rights reserved.

Introduction

The example application is a front end that interacts with a database. The database application represents a simple order placer and tracking application where the mobile user can place orders for various clients using his or her mobile phone. The mobile user can track client orders using a search facility provided by the application. For simplicity the search criteria is limited to order *id* only. The following structure represents the application's order record structure.

- `userid` represents the mobile user *id*.
- `customerid` represents the *customer id* whose order is being placed.
- `productname` is the product name.
- `productquantity` is the quantity of product ordered.
- `orderstatus` represents the status of the order.
  Y represents that the order has been shipped.
  N represents that the order is pending.
- `syncstatus` represents the status of the order used during the synchronization process.
  N represents a new order (order has been inserted).
  U represents an updated order.
- `orderid` is the unique *order id* generated for each `Order` for tracking purposes.
- `orderdate` is the date when the `Order` was placed.

Defining user and database interfaces

Take a look at the initial code for the J2meMob.java application. The line numbers used 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 searchField = new TextField ("Search By Order Id", "", 25,
        TextField.ANY);
        TextField userId = new TextField ("User Id", "", 10, TextField.ANY);
        TextField customerId = new TextField ("Customer Id", "", 10, TextField.ANY);
        TextField productName = new TextField ("Product Name", "", 40,
        TextField.ANY);

Line 4:  RecordStore recStore;
```


© Copyright IBM Corporation 2005. All rights reserved.

```
Line 5: private Command searchOrderButton= new Command("Search Order",
        Command.OK, 1);;
        private Command searchResultsButton= new Command("Search Results",
        Command.OK, 1);;;
        private Command insertOrderButton = new Command("Insert New Order",
        Command.OK, 1);;
        private Command addOrderButton = new Command("Add Order", Command.OK, 1);;

Line 6: public J2meMob () {

Line 7: mainForm.addCommand (searchOrderButton);
        mainForm.addCommand (insertOrderButton);
        mainForm.append(errorMessage);
        mainForm.setCommandListener (this);
```

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

Lines 2 through 7 define the application's user interface elements, the `RecordStore` class for interacting with the RMS, and control elements for capturing user input.

Life cycle methods

Lines 8 through 10 provide the MIDlet life cycle methods discussed in the section on The MIDlet life cycle. Every MIDlet class must provide these methods.

```
Line 8: public void startApp () {  createDatabase();
 }
Line 9: public void pauseApp () { }
Line 10: public void destroyApp (boolean unconditional) {
 }
```

I'll discuss the `createDatabase()` method in more detail in the next section.

The createDatabase() method

The `createDatabase()` method calls the method `connect()`, which in turn uses the `RecordStore.openRecordStore("OrderDB", true )` method to create an `OrderDB` record store (if one has not already been created). The `OrderDB` record store represents the example application's order database. Here's the code for creating the `OrderDB` record store.

© Copyright IBM Corporation 2005. All rights reserved.

```
            try{
                  recStore = RecordStore.openRecordStore("OrderDB",false);
               }
              catch( RecordStoreNotFoundException re ){

              }

              if(recStore == null){
                 //Create New one
                  recStore = RecordStore.openRecordStore("OrderDB",true);

              }
```

Inserting order information

Lines 11 through 18 place the order by inserting the order information into the `OrderDB` record store. After the order is placed, the example application generates a unique *order id* and displays it back to the mobile user for tracking purposes. Here's the code for placing the order.

```
Line 11 : public void commandAction(Command c, Displayable d) {

Line 12 :if(c == addOrderButton){


Line 13 : Form orderIdForm = new Form("Order Information");


Line 14 : String orderId = insertRecord();

Line 15 : orderIdForm.append("Your order has been inserted, the order
          is  "+ orderId);}

Line 16 : display.setCurrent(orderIdForm);

Line 17 : private String insertRecord(){

Line 18 :  String orderId =null;
           int recordID = 0;
           ByteArrayOutputStream bytstream = new ByteArrayOutputStream();
           DataOutputStream writer = new DataOutputStream(bytstream);

           long timeStamp = System.currentTimeMillis();

            //Generate Unique Key.
            orderId = userId.getString() + String.valueOf(timeStamp);

            writer.writeUTF(orderId);
            writer.writeUTF(userId.getString());
            writer.writeUTF(customerId.getString());
            writer.writeUTF(productName.getString());
            writer.writeUTF(orderQty.getString());
            writer.writeUTF(ORDER_SHIPPED);
            writer.writeUTF(RECORD_ADDED);
            writer.writeLong(timeStamp);
            writer.flush();

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


© Copyright IBM Corporation 2005. All rights reserved.

```
                    System.out.println("orderId"+orderId);
                    writer.close();
                    bytstream.close();
                    }
```

Note that the `insertRecord()` method inserts the order information entered by the mobile user into the `OrderDB` record store and uses the `RecordStore.addrecord()` method to commit the sequence of data represented as sequence of bytes. This action is shown in Line 18 above.

Retrieving order information from the database

Lines 19 through 26 retrieve the order information based on the *order id*.

```
Line 19 : public void commandAction(Command c, Displayable d) {

Line 20 :if(c == searchResultsButton){


Line 21 : Vector results = null;

Line 22 : results = fetchData(searchField.getString());

Line 23 : if(results.size() > 0) {
          searchResultsForm.append(userId);
          searchResultsForm.append("\n");
          searchResultsForm.append(customerId) ...

Line 24 : }

Line 25 : private ResultSet fetchData(String data){

Line 26 :  ByteArrayInputStream stream;
           DataInputStream reader;
           String orderID;

           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);
                orderID = reader.readUTF();

                       if(orderID.equals(orderId)){

                        records.addElement(orderID);
                        //User Id
                        records.addElement(reader.readUTF());
                        //Customer Id
                        records.addElement(reader.readUTF());
                        //Product Name
                        records.addElement(reader.readUTF());
                        //productquantity
                        records.addElement(reader.readUTF());
                        //orderstatus
                        records.addElement(reader.readUTF());
                        //status
```


© Copyright IBM Corporation 2005. All rights reserved.

```
                        records.addElement(reader.readUTF());
                        //create date
                        records.addElement(reader.readUTF());
                       }
```

The `fetchData()` method at line 22 searches the `OrderDB` record store based on the user-entered *order id* and returns the result set in a Vector. Next, the `fetchData()` method gets each record based on its unique record identifier, loops through each of the `OrderDB` records, and compares it with the user-entered *order id* to find a match. At line 23, you loop through your `Vector` and display the values to the user.

Now that you've seen how the code works, you can see it in action by running your application in an emulator!

Section 5. See the application code in action

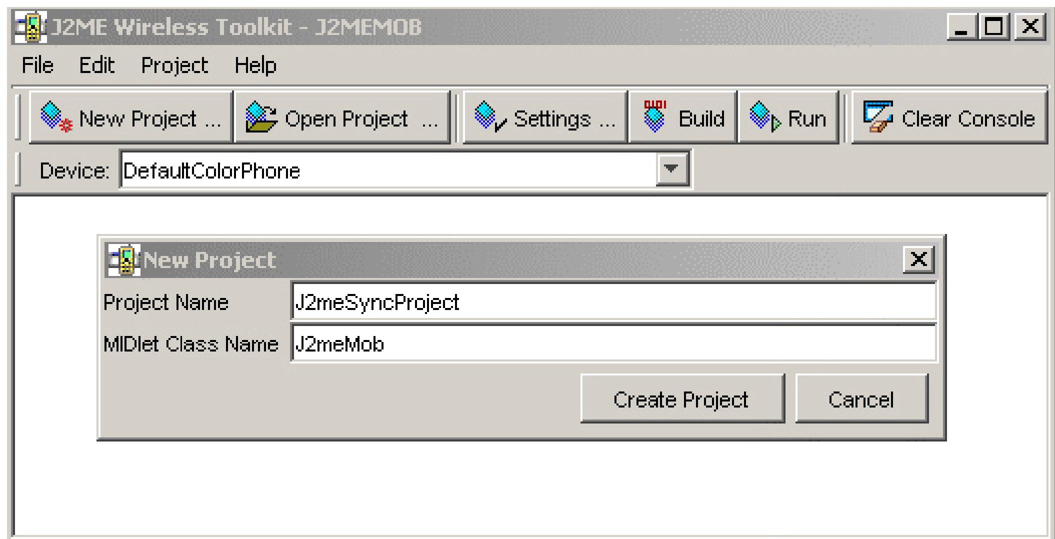
Creating the MIDlet

Before you can run the application, you need to create a new MIDP project in the J2ME toolkit. To do so, follow these steps in your J2ME Wireless Toolkit:

1. Select **Start > Programs > J2ME Wireless Toolkit 2.2 > Ktoolbar.** The toolkit window opens.

2. Create a new project using the Ktoolbar. Enter `J2meSyncProject` as the project name and `J2meMob` as the MIDlet class name. Click on **Create Project.**
   Figure 2. Opening the sample application, step 1

© Copyright IBM Corporation 2005. All rights reserved.



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

4. Copy the source file, J2meMob.java, from C:\j2mesync\src to C:\wtk22\apps\J2meSyncProject\src. (Remember, the J2ME Wireless Toolkit is installed in the C:\wtk22 path.)

5. Select **Build** on the Ktoolbar. You receive the following message:

```
Project settings saved
Building "J2meSyncProject"
Build complete
```

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

Launching the application

To run the example J2meSyncProject application, select **Run** on the Ktoolbar. The default emulator shown in Figure 3 should appear.

Figure 3. Opening the sample application, step 2

© Copyright IBM Corporation 2005. All rights reserved.

Running the application

Launch the application, select **Menu**, and you should be provided with menu options as shown in Figure 4.

Figure 4. Menu options for data input

© Copyright IBM Corporation 2005. All rights reserved.

Select **Insert New Order**, and you are presented with the screen shown in Figure 5. Click **Menu** and select **Add Order** to add the order information.

Figure 5. Insert Order has been selected

© Copyright IBM Corporation 2005. All rights reserved.

After an order has been entered, the user is given an *order id*, which is used for tracking his or her order. The application generates a unique *order id* based on concatenation of `userid` and the current timestamp, as shown in Figure 6.

Figure 6. An example order Id

© Copyright IBM Corporation 2005. All rights reserved.

Running the application

To track orders, select **Menu** and you should be provided with the menu options as shown in Figure 7.

Figure 7. Menu options to perform a search

© Copyright IBM Corporation 2005. All rights reserved.

Select **Search Order** and you are presented with the screen shown in Figure 8.

Figure 8. Insert an order Id

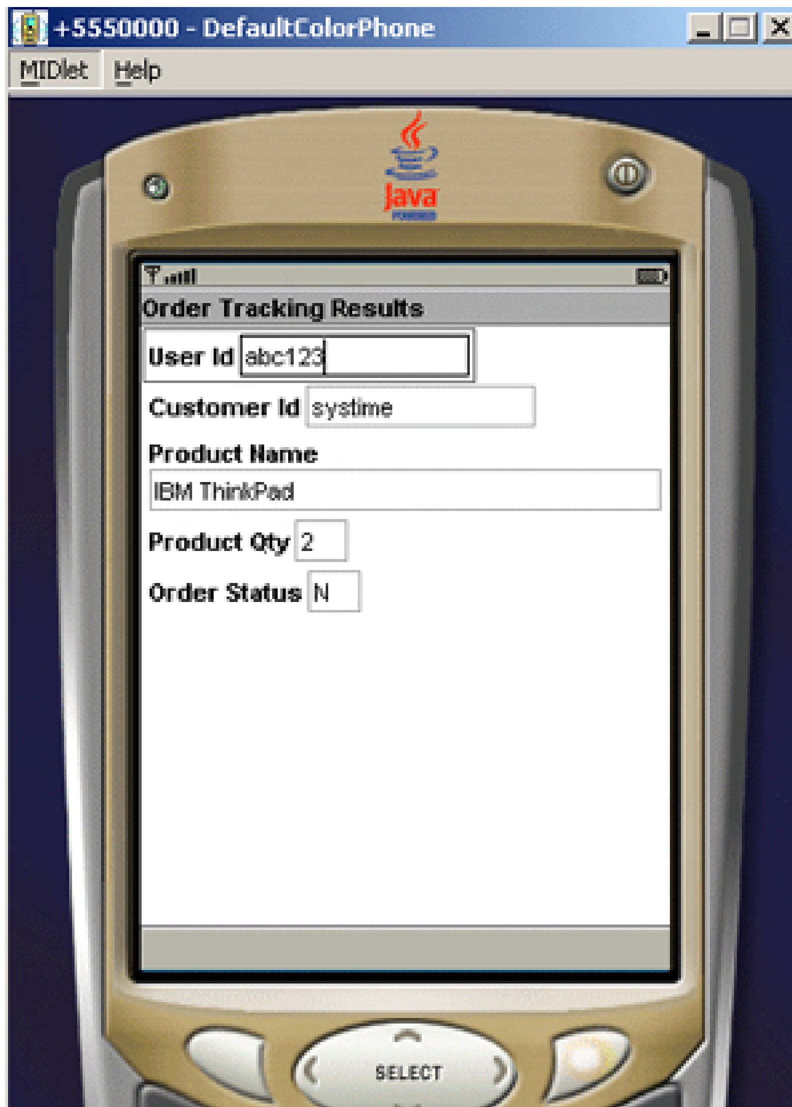
© Copyright IBM Corporation 2005. All rights reserved.

Enter the *order id* of the order for which you are searching. Click **Menu** and select **Search Results** to search the order information.

If the *order id* matches, the user is given his or her order information along with the status of the order, as shown in Figure 9. An order status of "N" indicates the order has not been shipped. After you have synchronized the order with the remote database (which you'll learn to do in Part 2 of this tutorial series), and it has been acted upon by the remote application, the order status is updated to "Y". Figure 9 shows the order information screen.

Figure 9. Search order results

© Copyright IBM Corporation 2005. All rights reserved.



Section 6. Summary

Summary

In this first part of a two-part tutorial series on building smart J2ME mobile applications, you have successfully used the J2ME record management system and a MIDlet to build a simple J2ME wireless application for mobile order placement and tracking. In the next part of the series, you'll learn to write your own synchronization

© Copyright IBM Corporation 2005. All rights reserved.

logic to synchronize your application's order information with a remote Cloudscape database.

© Copyright IBM Corporation 2005. All rights reserved.

Downloads

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

Information about download methods

© Copyright IBM Corporation 2005. All rights reserved.

Resources

Learn

- Visit java.sun.com for the latest version of these Java technology products:
  - The J2ME Wireless Toolkit 2.2.
  - The Java SDK 1.4.1 (Java Software Developers Kit).
- Learn more about wireless application development with J2ME through Naveen's two-part tutorial on "Accessing DB2 Everyplace using J2ME devices" (developerWorks, April 2004).
- You'll find tutorials about every aspect of the wireless frontier in the developerWorks Wireless technology zone.
- Stay current with developerWorks technical events and Webcasts.
- Get involved in the developerWorks community by participating in developerWorks blogs.

Get products and technologies

- Build your next development project with IBM trial software, available for download directly from developerWorks.

About the author

Naveen Balani

Naveen Balani spends most of his time designing and developing Java 2 Platform, Enterprise Edition (J2EE)-based frameworks and products. He has written various articles for IBM® developerWorks in the past, covering topics like ESB, SOA, JMS, web services Architectures, CICS, AXIS, J2ME, DB2® XML Extender, WebSphere® Studio, MQSeries, Java™ Wireless Devices, and DB2 Everyplace for Palm, Java-Nokia, Visual Studio, .Net, and wireless data synchronization. You can reach him at naveenbalani@rediffmail.com.

© Copyright IBM Corporation 2005. All rights reserved.