Mobile Software Application Development

Tutorial

By

Caesar Ogole

April 2006

About the Tutorial:
In this tutorial, you will learn how to build a cross-platform mobile software application that runs on the modern phone Operating Systems such as the Symbian and the Palm OS, typically found in Nokia, Motorolla, Sony Ericsson, LG, and other popular cell phones. You will understand how to set up the development environment, write, debug, test (using multiple simulators), and deploy the applications in the actual phones. It is assumed that you have basic knowledge of the principles of (mobile) Software development.

I developed this rather simple tutorial from part of a class exercise for the course “Mobile Software, 2006”. I am compelled to release this intellectual property for public use because I know somebody out there needs it badly. Moreover, it would make little sense for me to keep it to myself yet someone out there could reap big from it before the dynamic technologies become obsolete. Please, make good use of it and be sure that the least I can ask from you is to merely include my name as the reference for part of your work.
1. Introduction

In the “2nd Working IEEE/IFIP Conference on Software Architecture (WICSA), 2001”, Bosch J et al, in their submission On the Notion of Variability in Software Product Lines defines software variability as the “ability of a software artifact to be changed or customized to be used on multiple contexts”. This concept covers virtually all kinds of software including mobile software. In this paper, I present, systematically, one of the ways of writing/creating and deploying cross-platform mobile applications using selected (and the latest) mobile software development tools and interfaces available on the market. In this demo, the functionality of the software will be as simple as one that just displays a “Hello World!” message on (screen of) a mobile device.

It is shown in this report that the program, whose source code is also appended here, runs on multiple mobile operating systems. This discussion is limited to the Symbian and Palm Operating Systems.

2. Development Environment (Tools and Devices)

2.1 J2ME: Short for Java 2 Platform Micro Edition. J2ME is Sun Microsystems' version of Java used for developing applications that can run on a consumer wireless device platform (like mobile phones, PDAs, TV set-top boxes, printers and a broad range of other embedded devices). J2ME allows developers to use Java and the J2ME wireless toolkit to create applications and programs for wireless and mobile devices. The Sun Java Wireless Toolkit (formerly known as J2ME Wireless Toolkit) is a set of tools for creating Java applications that run on devices compliant with the Java Technology for the Wireless Industry (JTWI, JSR 185) specification. It consists of build tools, utilities, and a device emulator. The J2ME technology consists of a virtual machine and a set of APIs suitable for tailored runtime environments for these devices. The APIs are defined through the Java Community Process (JCP). The J2ME technology has two primary kinds of components – configurations and profiles. As it is commonly known, Java is platform-neutral.

2.2 NetBeans Mobility Pack

This Integrated Development Environment (IDE) is used to write, test, and debug applications for the Java Micro Edition Platform technology-enabled mobile devices. The
NetBeans Mobility Pack integrates support for the Mobile Information Device Profile (MIDP) 2.0 and the Connected Limited Device Configuration (CLDC) 1.1.

As we will see later, the Netbeans IDE provides a mechanism for integrating third-party emulators for a robust testing and deploying environment. There exists the NetBeans Mobility Pack for CDC as well. This paper will not discuss this.

Furthermore, the NetBeans Mobility Pack offers support for development of applications for multiple platforms by adding and executing device-specific code as configurations within a single application.

Other features of the IDE Pack include Integrated Obfuscation and Optimization support, which adds security, improves performance, and optimizes the size of the application. (Details on when and how to use some of these features will be demonstrated in the subsequent sections).

These (and more) NetBeans tools can downloaded freely from http://www.netbeans.org

The Sun Java Wireless Toolkit is available for download at http://java.sun.com

2.3 Symbian OS Emulator: Carbide.j

As Nokia puts it, “Carbide.j (formerly Nokia Developer's Suite for J2ME) is a software development tool for Java™ Platform, Micro Edition (Java™ ME) developers that enhances the development and verification of applications for Nokia devices. It provides tools for creating Mobile Information Device Profile (MIDP) and Personal Profile (PP) applications and deployment packages, signing applications, and deploying applications to devices. It is also an essential tool for managing, configuring, and running emulators for various Nokia platform and device SDKs.”

This tool can be downloaded from http://www.forum.nokia.com

In this project, Carbide.j has been integrated with Netbeans. Carbide.j can be run either as a stand-alone application or within an IDE. Although Carbide.j provides many development features, we will use Carbide.j here only for creating packages that are deployable in Symbian and Palm OS-run devices. (Details will follow in the next sections). Another reason for having Carbide.j installed here is that it comes along with emulators for Symbian OS Devices.

A screen shot of the Carbide.j embedded within NetBeans 4.1 is shown in the Figure 1.
Figure 1

2.4 Palm OS Emulator (POSE)
Freely downloadable from the palm website, http://www.palmos.com/dev/tools/emulator/, the Palm OS Emulator is software that emulates the hardware of various models of Palm powered™ handhelds. By supporting writing, testing and debugging of applications, the emulator allows a developer to create "virtual" handhelds on Windows, Mac OS, or Unix computers.
As we will see, the POSE accepts applications in the prc format.
A POSE interface looks like this:
Figure 2
3 Writing, Debugging and Testing a MIDlet, and Deploying Applications

Having setup the development environment as outlined in Section 2, we can now write a simple application program that runs on a mobile device. A MIDlet is an application written for MIDP. MIDlet applications are subclasses of the javax.microedition.midlet.MIDlet class that is defined by MIDP.

Here is the source code for a “Hello World!” like program. It makes use of the major classes in the J2ME package.

```java
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class HelloMidlet extends MIDlet implements CommandListener
{
    public HelloMidlet()
    {
    }
    private void initialize()
    {
        Display.getDisplay(this).setCurrent(get_helloTextBox());
    }
    public void commandAction(Command command, Displayable displayable)
    {
        if (displayable == helloTextBox)
        {
            if (command == exitCommand)
            {
                Display.getDisplay(this).setCurrent(null);
                destroyApp(true);
                notifyDestroyed();
            }
        }
    }
    private TextBox get_helloTextBox()
    {
        if (helloTextBox == null)
        {
            helloTextBox = new TextBox(null, "Hello World!", 120, 0x0);
            helloTextBox.addCommand(get_exitCommand());
            helloTextBox.setCommandListener(this);
        }
        return helloTextBox;
    }
    private Command get_exitCommand()
    {
        if (exitCommand == null)
        {
            exitCommand = new Command("Exit", Command.EXIT, 1);
        }
        return exitCommand;
    }
    TextBox helloTextBox;
    Command exitCommand;
    public void startApp ()
    {
        initialize();
    }
    public void pauseApp ()
    {
    }
    public void destroyApp (boolean unconditional)
    {
    }
}
```

Mobile Software Variability, Ogole Caesar, 2006
3.1 Compilation:

Within the NetBeans IDE,

- The MIDlet source code (on previous page), in a file named HelloMidlet.java, is written and compiled into HelloMidlet.class. The file could also be compiled under the J2SE (with extra compile command options). Under Netbeans IDE, pressing the F5 (or F6) builds the .class file(s) and runs the MIDlet with the default device emulator that comes as part of the Sun Java Wireless Toolkit. A sample run for this application is shown in the figure below:

![Figure 3](image)

Successful compilation means there is no error.*

*At this level, you know what I mean.
3.2 Testing (using compliant device emulators) and deployment

It is possible to add other devices emulators through the Tools>Java Platform Manager> Add Platform of the Netbeans IDE. While this exercise is possible for emulators, some device emulators input applications in other formats. For this reason, let’s look at the Nokia’s Symbian OS and Palm OS one at a time.

3.2.1 Symbian OS

.jar and .jad files are generated (in a single run) using the Carbide.j utility embedded within the IDE: Tools>Carbide.j>New Application Package

In order to run the application on the emulator, a desired (Nokia) device emulator is selected and run with the .jad file from the interface: Tools>Carbide.j>Start Emulators

![Start Emulators](image)

Figure 4
A sample run with after a press on the “Emulate” button yields the output in Figure 5(a). Launch the application by pressing the OK soft key (Figure 5(b)).
Deployment

Case 1: Deploying to a device

Preliminary Settings:

- First establish the **Bluetooth communication** between the mobile device and the PC using, for example, the Nokia PC Suite (installed together with Carbide.j) and **Get Connected**.

![Image of Nokia PC Suite interface]

**Figure 6: Interface to the Nokia PC Suite**

- Then, use the Carbide.j’s **Deploy Facility**:

  A **JAD, JAR** is selected using the **browse button**. Alternatively, the file location is typed directly into the text field. To deploy an application package consisting of a JAD file and a JAR file, both files must be found in the local file system. Particularly, JAD file’s MIDlet-Jar-URL attribute must point to the correct location where the JAR file can be found. If the JAR is not found, an error message is displayed.
Having selected the application file(s), connect the device. A list of connected and compatible devices is displayed. Carbide.j 1.0 supports connections using Bluetooth, IrDA, USB and RS-232 serial cable. Click **Deploy** to deploy the files. Deployment status is shown in the message area.

**Case 2: Deploying to a Server**

The JAD and JAR files are sent to the specified location (directory) in the server through an established FTP channel. To enable FTP deployment, enter the following information: *FTP host*, *FTP port* to be used (normally 21), *username* for logging into the FTP server, *password* to use when logging into the FTP server and the *directory* where files will be uploaded. When all information is set, click **Deploy** to deploy the files. Deployment status is shown in the message area. (See the **Server Tab** in Case 1)

3.2.2 Palm OS

- For Palm OS, the Midlet must first be converted to a PRC format. The **MIDP for Palm OS** comes with a converter tool to convert a MIDlet to an executable Palm application, with the extension PRC. There are several to launch the converter.
• Here, I issue the command: `java -jar Converter.jar` in the directory where the batch-file `Converter.bat` resides.

• Using Converter, a JAD file is converted into a PRC file which is by default resident in the same directory as the JAD file.

```
Figure 8(a)  Figure 8(b)
```

The message on successful conversion of the JAD file to PRC (Figure 8 c)

```
Figure 8(c)
```

Mobile Software Variability, Ogole Caesar, 2006
To run the application on the **Palm Os emulator, the PRC file** is simply dragged and dropped on to the running emulator. Our “Hello World!” runs as shown on the Figure on the next page (Figure 9).

**Install and Test**

Once the JAD/JAR file pair has been converted to a PRC file (as we have done), it can be installed on the Palm OS device using **HotSync**. Once installed, it is run it and components are selected to be tested.

![Figure 9: The POSE emulating the “Hello World!” Application](image)

*Figure 9: The POSE emulating the “Hello World!” Application*
4. How to protect the program from illegal copying

Cryptographic Solution: Sign Application Package

4.1 Signing a MIDlet allows the user to authenticate the sender of the application through the use of certificates and ensures the integrity of the application with public/private key security features. The signature of the JAR is created with the signer’s private key. The signature and public key certificate are added to the JAD as attributes. The device uses them to verify the signature and to complete the authentication using a root certificate bound to a protection domain on the device.

4.2 Implementation Symbian using Nokia’s Carbide.j utility

Under the NetBeans IDE:

Go to:

Tools> Sign Application Package tool to sign a MIDlet Application Package (Figure 10).

Procedure:

- After creating a MIDlet application package, you can sign it using a public/private key pair and an associated public key certificate. Each key pair and the associated public key certificate are identified with a name, an alias.
To have an application actually signed, an alias for the key pair you wish to use is selected. The press Sign... This will bring up a file dialog, from which you must select the MIDlet application package's JAD file (HelloWorld.jad, in this case). The private key and the public key certificate corresponding to the selected alias are used in signing the MIDlet. The tool will notify you when it has successfully signed the application package.

You can sign the application package with several keys. If you sign the application package with different keys, each signed application package is saved.

The default key is a sample dummy key, Nokia SDKs and devices don't have certificates that would allow you to use the default key for verifying MIDlets.

The screenshot below shows the message after the signing of the application (HelloWorld.jad) has been carried out successfully.
5. Remark

Finally, I have to comment on the efforts of writing a program for different mobile operating systems:

- The biggest trouble with writing cross-platform applications is the tooling. Setting up the development environment is really such an intimidating exercise. Moreover, much as I tried to use a so-called Integrated Environment, I have to do several configurations on the various components in order to build a nearly unit environment. Nonetheless, NetBeans has its own strenght. Experienced developers like my fellow Sun Campus Ambassador (Teera Kanokkanjanarat, http://blogs.sun.com/branajam/entry/nb_community_docs_spotlight_teera) have a better insight:

"First of all, I like the simplicity of development with Netbeans IDE. I've had experience working with Eclipse and IntelliJ IDEA. But by far, I found myself most productive developing with Netbeans. I also like NetBeans Visual tools and add-on modules very much. I use Matisse GUI Builder, visual XML/XSD, and WSDL tools on regular basis. The last thing to note is the Ruby support in Netbeans 6. It's simply one of the best Ruby IDE out there."

- Not any tool (programming language, for example) can be used in developing applications that run on multiple platforms. This means that some of the well-known user-friendly languages are of no use. This further makes application development not very nice!

- Nevertheless, work becomes somewhat easier when everything is set up. The fact that there exist emulators that very closely “emulate” real devices means that applications can be fine-tuned before the actual deployment. This saves more time and other resources.

- Reusability could lessen the difficulty encountered in the development.
There you go! You have the framework for building any application beyond the “Hello Container Boys!”.

Write to me: cogole@gmail.com but I’m too busy to respond to your questions, sorry: you google!}

Mobile Software Variability, Ogole Caesar, 2006