Introduction to GPS, Part 3

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.

1. Introduction.. 2
2. Global positioning applications...................................... 6
3. The "Write once, run anywhere" promise 8
4. Java programming environments.................................... 11
5. Sample Application: MIDlet... 15
6. Sample application: Java in the Palm OS.......................... 20
7. Summary 26

Section 1. Introduction

Should I take this tutorial?

This tutorial is designed for developers and product managers considering the Java programming language as an option for use in Global Positioning System (GPS) applications. Developers with an interest in creating Java applications for PDA and embedded environments learn the challenges of embedded Java development in resource-limited environments, including available tools and virtual machine options. The broader topic of small-scale Java capabilities, limitations, and packaging options offers valuable market perspective for product managers.

This is the third tutorial in a three-part series on developing GPS applications with Java technology. It's recommended that you read through the first two tutorials, *Introduction to GPS, Part 1* and *Introduction to GPS, Part 2*, where fundamental GPS principles are introduced and discussed. The prerequisite tutorials demonstrate basic Java programming skills necessary for handling GPS data. The sample applications presented in this tutorial focus on porting these GPS applications to a Java PDA environment.

What is this tutorial about?

This tutorial examines the options, obstacles, and opportunities associated with Java-based GPS applications on portable devices such as PDAs and cell phones. This tutorial explores two basic programming models for portable GPS applications and examines the tools and environments required to build them. The first variety of application deviates slightly from the theme of the prior tutorials as it is characterized by the lack of a local GPS receiver unit. This application variant examines informational applications, focused on the *use* of GPS data, rather than the collection of it. The second variety of application explores the Java programming options available for the Palm OS-based GPS applications. Consistent with the experiences from the first two tutorials in this series, communicating with a GPS receiver via the Java programming language has its challenges. Various development options are examined during the construction of a Palm OS-based Java application that extracts GPS data directly from a Garmin unit via a serial connection. The tutorial consists of the following sections:

- Global positioning applications on page 6 : This section discusses two models of GPS applications including those that make use of GPS data versus those that collect GPS data.
- The "Write once, run anywhere" promise on page 8 : Java technology has proven itself in the application server environment and the browser client. This section explores the question of whether Java technology has lived up to the hype and promise in the wireless device market.
- Java programming environments on page 11 : This section offers a brief survey of the development tools and environments available to Java developers. This section also revisits the I/O requirements of data collection devices and the options available in

the Java environment.

- Sample Application: MIDlet on page 15 : This section details the creation of a MIDlet application, suitable for a cell phone or PDA, using the WebSphere Studio Device Developer environment. It demonstrates an HTTP connection for the purpose of using GPS data in a wireless application setting. The basis of the application rests upon a data retrieval model, supported by a remote Web/application server infrastructure.
- Sample application: Java in the Palm OS on page 20 : This section examines a Java Palm OS application utilizing the SuperWaba environment. The choice of this particular environment is due to its ability to "hide" the Palm OS specifics into a simple-to-follow Java application.

Source code roadmap

This tutorial's sample applications demonstrate a variety of Java- and GPS-related techniques. Examples of a simplistic MIDlet user interface, HTTP connections, serial connections, and Java/Palm OS user interface invocations are present. The following code snippets are found in this tutorial:

- The imports of a MIDlet, including the appropriate packages for network connectivity required for the sample application's GPS lookup function, are examined.
- User interface elements of the MIDlet are explained.
- The `commandAction` method, which handles events initiated by the user of the MIDlet, is presented and explained.
- An HTTP network connection is initiated and processed when the "Go" button is pressed in the MIDlet.
- The configuration of Mobile Creator is important to the successful building of the second sample application. A screenshot depicts the settings used to construct the sample application.
- The Mobile Creator build process is presented, including all of the steps that Mobile Creator hides from the SuperWaba developer, making the experience of getting started that much more palatable.
- The user interface elements of the SuperWaba application are presented and explained.
- The `AddString` method, which is the primary means of user feedback in the `ibmdwgpspalm` application, is examined.
- The `onEvent` method, which reacts to user input and drives the core functionality of the application, is presented.
- The `ProcessPort` method, which handles incoming data from the GPS unit, is examined.

Some helpful terms

- Byte stream: Data sent from one device to another can be characterized as a stream of individual bytes.
- DataInputStream: An implementation of the DataInput interface and an extension of the InputStream class. The unique characteristic of the DataInputStream is its ability to read Java native data types directly from a stream.
- Query: A very generic term used to represent a request or question originating from a "client" application and responded to or answered by a server or service.
- HTTP: Hypertext Transport Protocol
- eCOS: The embedded Configurable Operating System, an open-source, real-time operating system managed by RedHat, the distributor of a popular release of Linux.
- Eclipse: A comprehensive, open-source development environment.
- J2ME: Java 2 Micro Edition

Tools

This tutorial introduces a variety of software tools as viable alternatives for embedded Java development in an effort to effectively paint the landscape of tool offerings. Although there are many available Java development environments to choose from, the tools chosen for use in this tutorial are the Websphere Studio Device Developer, based on the Open Source Eclipse environment, and SuperWaba in conjunction with Mobile Creator. Eclipse itself encompasses many technologies and is a project unto itself. For the purposes of this tutorial, it is presented as a useful tool for managing various build targets -- a unique aspect of wireless cross-platform development. Mobile Creator and SuperWaba combine to be (arguably) the easiest Java development environment for the Palm OS.

- Websphere Studio Device Developer: This toolset, downloadable in evaluation form at *http://www.embedded.oti.com/wdd/*, is the development environment and toolset for the first sample application presented in this tutorial.
- SuperWaba: This development environment, toolset, and virtual machine are among the most popular Palm OS and WinCE Java development resources. SuperWaba is employed for the second sample application and can be downloaded at *http://www.superwaba.org*.
- Mobile Creator: This simple IDE simplifies SuperWaba development for the intermediate embedded Java developer. With a few entries in the preferences panel, the developer need not venture into the world of "DOS Prompt" commands and batch files for compilation. Mobile Creator is employed for the second sample application and can be downloaded at *http://www.tauschke.com/main.htm*.
- The Java Development Kit: This tool is the underlying compilation mechanism for the second sample application and can be downloaded at *http://java.sun.com*.
- Palm OS Emulator: The tutorial's sample applications are most conveniently tested on the Palm OS emulator available for download from *http://www.palmos.com/dev/tools/emulator/*.
- GPS Unit: The tutorial's second sample application demonstrates communications with a consumer GPS unit capable of the NMEA data protocol. For more information on the NMEA protocol, please refer to the *Introduction to GPS, Part 1* tutorial here on

developerWorks.

- Tutorial Sample Code: The complete code to the tutorial's sample applications can be found at: *http://www.palm-communications.com/ibmdw* .

About the author

After his college basketball career came to an end without a multiyear contract to play for the L.A. Lakers, Frank Ableson shifted his focus to computer software design. He enjoys solving complex problems, particularly in the areas of communications and hardware interfacing. When not working, he can be found spending time with his wife Nikki and their children. You can reach Frank at *frank@cfgsolutions.com*.

Section 2. Global positioning applications

Global Positioning System: Application opportunities

The previous tutorials in this series on Java and GPS introduced the basics of serial communication from the Java environment as well as some fundamental GPS concepts such as waypoints and routes. This tutorial explores the viability of GPS applications, built in Java language, for PDAs and other "embedded" environments. GPS offers significant technical information; terms such as location, velocity, and estimated-time-of-arrival (ETA) to a specific location represent commonly encountered technical points when discussing GPS. While technology for its own sake can make headlines and even move major markets for a spell (remember the dot com Nasdaq bubble?), without marketable products and viable applications, even the best technologies and "blue-sky" optimism wane before long. The goal of this tutorial is further education on the tools available for Java/GPS applications, the underlying technologies that make them work, and the practical compromises that might be necessary to accomplish the goal of market viability.

The next panel examines the task of using GPS data after the technical task of retrieving it is complete.

Use the data, or simply collect it?

The previous tutorials focused primarily on the collection and recognition of GPS data from a local receiver. Collecting the data is, of course, only a portion of the solution. Proper rendering or transmission of that data is also of significant importance when building GPS applications. Common uses of GPS data include mapping, locating points of interest, and navigation. One of the more popular and common applications involves mapping current location against a full-color display representing local highways. However, not all devices are equipped to store and display such images. Perhaps another application model is required, particularly for resource-limited devices such as PDAs and cell phones.

One viable application model combines the network connection available with modern cell phones, the rich Java programatic environment available in the phone, and relevant GPS data from either a built-in GPS receiver or a previously loaded store of GPS position information. The phone may not have significant storage capabilities when it comes to maps and images; however, it can certainly store waypoints, which are no more resource-consuming than a telephone number. The Java environment also provides network communication capabilities. The phone's ability to "punch-out" to an Internet host and exchange information via HTTP presents a different, yet viable, application model.

The next panel introduces the other focus of this tutorial -- PDA and embedded Java programming tools and techniques for extracting GPS data in the Java environment.

Java technology and the PDA platforms

One thing that makes Java development alluring is its cross-platform nature. Unlike other languages, like C, Java byte codes run on any platform with a valid Java Virtual Machine (JVM). Unfortunately, this feature's practical benefits fade as the device becomes smaller and the features require lower-level access to the native platform. This lower-level access can be characterized by:

- Communications
- Output technologies (video, sound, print)
- Input technologies (keyboard, pen/stylus, voice)

There are a multitude of PDA offerings in the marketplace today. Some have features that rival low-end laptops, but the majority of PDAs and cell phones are still resource-limited, and the expectation is that this trend will continue for some time. Until power consumption and price-point challenges are overcome to allow high-resolution graphics, memory capacity, and affordability (to name a few features), GPS developers must hit the market where it currently exists.

The next section explores Java technology and the profiles that define functionality and conformance levels

Section 3. The "Write once, run anywhere" promise

The promise

The promise of Java software development is, in short, "write once, run any (and every) where." Originally thought to be the dragonslayer of client development woes, Java technology has proven itself to be the most comfortable and productive on the "back end," or "server side." For example, in the application server market (think Jboss, WebLogic, WebSphere, iPlanet, Oracle 9i, SeeBeyond), a servlet or bean can be developed on one platform, such as Solaris, and confidently deployed to another platform, such as AIX. This is true because the activities of the server side application consist of:

- Database interaction
- Fundamental "CompSci" actions such as searching and sorting
- Network communications (HTTP, Sockets, e-mail, FTP, etc.)
- Text generation (HTML, XML)

Notice that these activities essentially need three things to operate:

- CPU cycles
- Memory
- Network stack

These resources do not have the same device-specific limitations and constraints found in client-side applications discussed in the previous section such as display, sound, and data input mechanisms. It is precisely these resource challenges and the quest to categorize what variety of application should be expected to run on a particular device that lead to the topic of the next panel -- *configurations* and *profiles*.

Profile proliferation

JVMs can be found on everything from smart cards and chips up to the largest of environments, such as the OS/390 from IBM. Can a smart card be expected to drive a windowed terminal via the Abstract Windowing Toolkit? Of course not, as there is no windowed terminal to drive. Nor can the JVM on a large supercomputer be outfitted with the hooks and connectivity required for something such as I2C communication, found at the board level of embedded designs. Obviously, each of the platforms encompassed in this diverse spectrum has differing capabilities and priorities. This diversity in JVM functionality requires a means to manage device design, programming, and application expectations.

The answer to this challenge is effective *communication*. In this context, these expectations are set forth in *configurations* and *profiles*. Profiles are established through the collaboration of various working consortiums, consisting of manufacturers, software vendors, and other parties with a vested interest in the shaping of a given profile. There are many profiles available in the wireless marketplace including the IrDA.org's specifications, Bluetooth's application profiles, and of course, the profiles of primary interest to Java developers: the profiles governing the features and functions of a particular Java environment.

The next panel introduces the Java 2 Micro Edition (J2ME) configurations and profiles with relevance to mobile Java development and, as such, of importance to GPS application development.

MIDP, CDC, CDLC

Sun Microsystems has introduced a specific release and specification of Java technology addressing the mobile community, namely J2ME. This offering is a complement to the Java 2 Standard Edition (J2SE) and Java 2 Enterprise Edition (J2EE) specifications. The J2ME defines base configurations known as:

- Connected Device Configuration (CDC): The CDC is seen as the first step "down" from the full J2SE specification. It is designed for devices with ample resources such as memory and processor capabilities. It also defines networking functionality for conforming devices and applications. This configuration is meant for applications such as kiosks and similar dedicated-function applications.
- Connected Limited Device Configuration (CDLC). The CLDC is aimed at devices with significant restrictions on memory, power, and connectivity. CLDC devices include cell phones, pagers, and low-powered PDAs.

Profiles map to the base configurations.

- The Mobile Information Device Profile (MIDP) presents a customized collection of Java classes, stream-lined and optimized for CDLC devices such as cell phones. Core Java packages are present; however, certain classes are omitted or trimmed for the lean environment. For example, the BufferedInputStream class normally found in the `java.io` package is omitted in the MIDP version.
- The Personal Basis Profile (PBP) assumes the presence of a graphical user interface; however, it does not offer support for the Abstract Windowing Toolkit found in J2SE.

The next panel examines the bare minimum requirements for Java GPS applications on a MIDP/CLDC-compliant device as well as a Palm OS device in particular, which straddles the configurations in terms of capability and available offerings.

Minimum requirements

The minimum requirements for this tutorial's PDA/Gadget GPS applications fall into two categories:

Section 4. Java programming environments

Cross-platform strategies

Nearly all embedded Java development occurs in a cross-platform manner, meaning that the actual compilation and linking phases of the development cycle occur on a host environment that differs from the target deployment platform. For example, when developing the code for the sample applications, all development occurs on a Windows or Linux machine, as opposed to a cell phone or Palm OS device. This strategy of course is nothing new to developers with experience in the wireless or "handheld" space; this practice has been the norm for decades now.

However, Java development takes the cross-platform concept to another tangential level. Not only does the development platform differ from the deployment platform, but the "finished product" can also vary significantly between various implementations. Take the Palm OS platform, for example. The native application files for the Palm are "PRC" files. Java programs compile to "class" files, which in turn require a virtual machine to interpret them. Some Palm OS/Java environments consist of a "post process" of converting ordinary Java classes into Palm OS native, executable PRC files. Other environments package the required Java classes into a Palm OS database file (PRB) for use by a Palm-resident interpreter, acting the part of the JVM.

The next four panels explore a few of the available Palm/Java development environments.

Commercial offerings

Despite its large open source following, there are indeed commercial Java products available from organizations eager to capitalize on the demand for quality, high-performance software based on Java technologies. With Nokia and other manufacturers planning to push millions of Java-enabled devices onto the market over the next few years, Java vendors are vying for pole position. Each offering attempts to capitalize upon a distinct selling feature that sets their product apart from the competition. Until hardware capabilities in small spaces make these optimizations unnecessary, these companies will continue to argue that their brand of bread is sliced better than the next. Following is a small sampling of the major players, along with their unique selling propositions:

- The Jbed offering boasts a variety of features including on-device code compiler for dynamic compilation, native code compilation target, and a Windows-based IDE for application development. Jbed also makes a real-time version of its environment available for various platforms. (See Tutorial resources on page 26 for a link.)
- Simplicity for Palm OS Platform from Data Representations (see Tutorial resources on page 26 for a link) leverages IBM's J9 virtual machine for the underlying, run-time Java environment. Simplicity is a Rapid Application Design (RAD) tool for Java technology. The J9 virtual machine is available on numerous platforms and is a key

component of the Websphere Studio Device Developer.

- Websphere Device Developer and the Eclipse development environment offer a comprehensive, if not complex, development environment. Applications are compiled into an intermediate file format known as *JXE*. Once compilation is complete, a build step combines with a desired launch mechanism. In this manner, the same application can be launched in a variety of settings. For example, a MIDlet can be launched in a MIDlet simulator, as an application running in the Palm OS Emulator (POSE), or installed as a PRC onto a physical Palm OS device. The run-time Java environment employed is the IBM J9 Virtual Machine.

While each of these commercial offerings presents one or more unique selling features, there are Java development options available at no cost from Sun and other companies.

J2ME

Sun provides products for both the CDC and the CDLC configurations, including environments for PDAs such as Palm OS and WinCE. From a GPS perspective, the MID Profile does not support serial communications but does mandate HTTP connectivity. As such, the J2ME offering is a viable alternative for the first sample application, which relies solely on HTTP for retrieving relevant GPS data. John Muchow's *developerWorks* tutorial provides an excellent introduction to MIDlets and the use of the J2ME. See Tutorial resources on page 26 for more information.

The next panel examines a popular open source Java development environment and toolset known as SuperWaba.

SuperWaba

Perhaps the most popular Java option for PDA platforms is SuperWaba. This toolset provides a Java development environment that is outside the scope of traditional J2ME Java classes. SuperWaba provides its own virtual machine and custom class libraries. The code that is written is Java code; however, the core classes are part of the `waba` package instead of the `java` package. A standard Java compiler is used for compilation of projects into Java classes and packages. The core packages available for Waba include:

- `waba.fx` -- This package provides graphics and drawing classes.
- `waba.io` -- This package includes input/output classes, including the important SerialPort class. This is a key ingredient for GPS applications requiring direct access with a GPS receiver via an RS-232 connection.
- `waba.sys` -- This package provides access to useful items such as regional settings and system-level settings.
- `waba.ui` -- This package contains all of the user interface elements required for useful applications. These include controls such as buttons, labels, lists, and the like.

- `waba.util` -- This package provides useful data structures along with a `Date` object, which is undoubtedly important for virtually any application.

These classes provide all of the user interface and system level functions required for building useful applications. Additional classes and packages can be added by anyone, as the code is, in fact, pure Java code, compiled with the J2SE's JDK itself.

For the sake of completeness, the next panel introduces an industry buzzword that sounds like Java technology but is explicitly not Java technology.

BREW: Where multiple-language development percolates

Qualcomm's mark on the wireless economy is not soon forgotten. The owner of CDMA technology's intellectual and patent rights has an offering for the wireless development space. The initiative is called Binary Runtime Environment for Wireless (BREW). BREW supports multiple-language development, although Java language is not the first, native choice; that honor falls to C++. In a nutshell, BREW is a "light-weight fat client" for wireless devices. Its success is hinged upon the ability to effectively download entire applications and run them locally on a device without the necessity of a long-haul network connection.

BREW goes beyond the simple technology of what can be done and where, by addressing some of the economic aspects to wireless development. One of the largest impediments to wireless application, network, and device adoption is the lack of applications. Of course, this is a bit of a chicken-and-the-egg dilemma as it helps application developers to have a platform before investing in developing new or ported applications. In any event, BREW aims to assist in this adoption process by making it easier to achieve financial gain in at least two distinct manners:

1. BREW has taken carefully considered actions to ensure the validity of any downloaded application. This is achieved through the digital signature of applications with a trusted certificate from Verisign. Only to the degree that downloadable applications and content can be trusted, can mobile commerce take hold.
2. BREW reaches beyond technical specifications and delivers to carriers and other operators the tools necessary for distributing BREW applications via its BREW distribution system. This package includes download servers, transaction managers, and hooks for integration with operator billing systems.

The next panel summarizes this brief survey of development environments and describes the rationale behind the choices made for construction of this tutorial's two sample applications.

Java programming environment wrapup

The above sections just scratch the surface of the available Java development environments for PDAs and other "micro" environments.

Two distinct tools were chosen for use in developing the two sample applications. These tools were chosen based on their relevant strengths to accomplish the chosen task and their broader relevance to the marketplace and Java development community.

The first application, a simple MIDlet demonstrating an HTTP connection to retrieve arbitrary GPS data from a remote application/Web server, could be constructed with any number of tools. The WebSphere Studio Device Developer was chosen as it provides the most comprehensive development environment. It may also be the most complex toolset available. The same application can be readily targeted to multiple environments. This flexibility and high degree of control over the build process speak to the strength of the Eclipse environment.

Unfortunately, the tools used in the sample applications of the earlier tutorials are not readily available for the Palm OS, as the Abstract Windowing Toolkit and `javax.comm` are not part of J2ME. An alternative approach is required. The classes available from WebSphere Studio Device Developer offer a wrapper class for the entire Palm OS system. While this "OS" class is very comprehensive and virtually any task required on the Palm can be achieved through the use of this class, it requires an intimate knowledge of the underlying Palm OS "C" APIs. The use of such a class flies in the face of the original goal of this tutorial -- to port the application from the prior tutorials into a PDA environment with as much ease as possible. Due to the simplistic nature of the application and the availability of an easy-to-use `SerialPort` class, the SuperWaba environment, in conjunction with the Mobile Creator IDE, was selected for use in the development of the second application.

The next section demonstrates the development of a MIDlet.

Section 5. Sample Application: MIDlet

Motivation/purpose of the sample application

This sample application demonstrates the use of the WebSphere Studio Device Developer environment to create a MIDP-compliant GPS application. The purpose of the application is to provide current (simulated) GPS information in the form of a query to a remote application or Web server. The back-end server responds with an XML message containing the names of sports venues within a 100-mile radius. This application represents one of a variety of applications for the GPS domain. To get started, be sure to obtain the WebSphere Studio Device Developer (see Tools on page 4). The next few panels walk through the process and the code snippets needed to construct this application.

Creating the project

Once the environment has been installed, a new project must be created. There are a few options available in the presented dialog box. Choose "J2ME for J9" as the type of project, and the "Create MIDlet Suite" as the specific target choice in the right-hand list box. Provide a name for the project. This sample application's name is `ibmdwmid`. Enter this name into every box except "package". Select Finish. The project's source and metadata files will now be created. When ready, the project appears in the tree-view window to the left. Note the Java imports automatically included. Additional imports are required. They are:

- `javax.microedition.io.*` -- This package includes important classes including the `Connector` class.
- `java.io.*` -- This package contains the InputStream class useful for reading data on the HTTP connection.
- `javax.microedition.lcdui.Command` -- This class includes the required functionality for presenting buttons for user input.
- `javax.microedition.lcdui.CommandListener` -- This class includes the required interface for reacting to user actions.
- `Javax.microedition.lcdui.Displayable` -- This is required for implementing the `commandAction` method of the CommandListener interface.

Note that the Eclipse editor displays the contents of any Java class in an outline/tree view. The currently selected Java class is shown by default. The next two panels examine the actual source code of the MIDlet.

User interface

The MIDlet contains only two buttons. Remember, a MIDlet is designed to run on an ultra-limited device, such as a cell phone. Here is the code to create the primary user interface:

```
public class ibmdwmid extends MIDlet implements CommandListener
{
private Command goCommand = null;
private Command exitCommand = null;
Form f = null;
HttpConnection url = null;
String targeturl = "http://gps.cfgsolutions.com/places.xml";

/**
* @see MIDlet#startApp()
*/

protected void startApp() throws MIDletStateChangeException
{
   Display d = Display.getDisplay(this);
   f = new Form("IBM developerWorks");
   goCommand = new Command("Go!",Command.SCREEN,1);
   exitCommand = new Command("Exit",Command.SCREEN,2);
   f.addCommand(goCommand);
   f.addCommand(exitCommand);
   f.setCommandListener(this);
   f.append("Ready.");
   d.setCurrent(f);
}
...
}
```

This code adds two buttons to the form and then sits idly by awaiting user input at one of the buttons. Button presses are handled by the `commandAction` method. Note that the `commandAction` method satisfies the requirement imposed by the `CommandListener` interface.

```
public void commandAction(Command c,Displayable s)
{
        if (c == exitCommand)
        {
              notifyDestroyed();
        }
        if (c == goCommand)
        {

        }
...
}
```

The next panel discusses the actual HTTP connection and the use of an InputStream for reading purposes.

URLs and streams

When the "Go!" button is selected, the application begins the actual connection and query to the remote site. The query is reduced to a simple HTTP GET operation:

```
f.append("Attempting to retrieve data");
try
{
        url = (HttpConnection) Connector.open(targeturl);
        InputStream is = url.openInputStream();
        while (is.available() > 0)
        {
                byte [] b = new byte[is.available()];
                is.read(b);
                f.append(new String (b));
        }
        url.close();
}
catch (Exception e)
{
        e.printStackTrace();
}
```

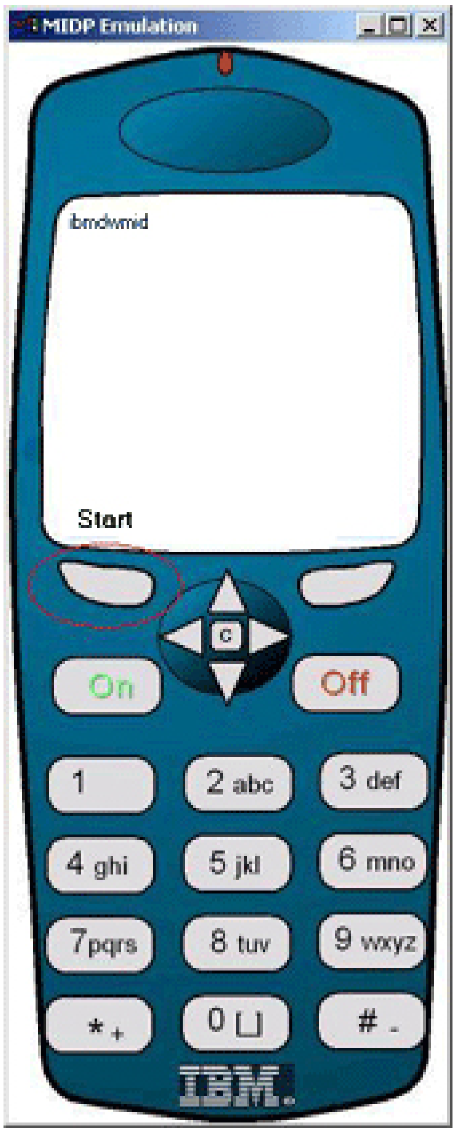
Note that the entire interaction with the remote host is accomplished within a try/catch block. Upon any error, control of the application passes to the simplistic catch block where it simply displays an error message to the standard error device.

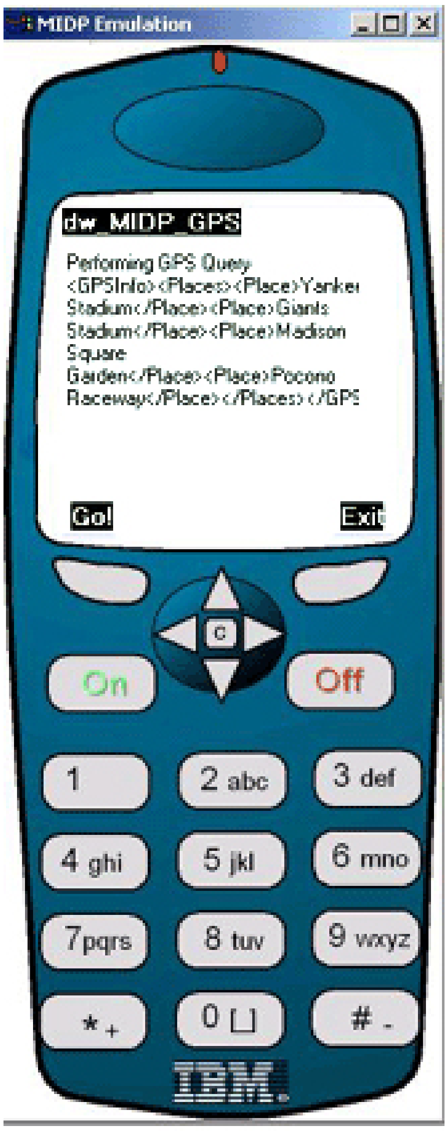
Once the connection is established, an InputStream is constructed to read data from the remote site. The MIDP classes do not provide for a BufferedInputStream, which is a desirable mechanism for PC-based Java communications.

The next panel demonstrates building and testing the application.

Building and testing

The Eclipse environment compiles each source file as it is saved. Any compilation errors show up in the "tasks" list and must be dealt with before the application can be tested. When compilation is clean, i.e., there are no tasks remaining in the list, it is time to build the actual application. Select the "Run" menu option and the "MIDlet Suite on MIDP Emulator" choice. This selection results in a dialog box. Simply select Finish. At this point, an emulated cell phone should appear with the `ibmdwmidp` application running. Here is a sequence of screen shots depicting the application flow:


MIDP Emulation
ibmdwmid
Start
C
On
Off
1
2 abc
3 def
4 ghi
5 jkl
6 mno
7pqrs
8 tuv
9 wxyz
* +
0 ␣
# -
IBM.



```
Performing GPS Query
<GPSInfo><Places><Place>Yanke
Stadium</Place><Place>Giants
Stadium</Place><Place>Madison
Square
Garden</Place><Place>Pocono
Raceway</Place></Places></GPS
```

Note that the buttons below the display control the action. The Start button is circled in the first image. This concludes the first sample application. The next section examines the use of SuperWaba for reading NMEA sentences from a Garmin GPS unit.

Section 6. Sample application: Java in the Palm OS

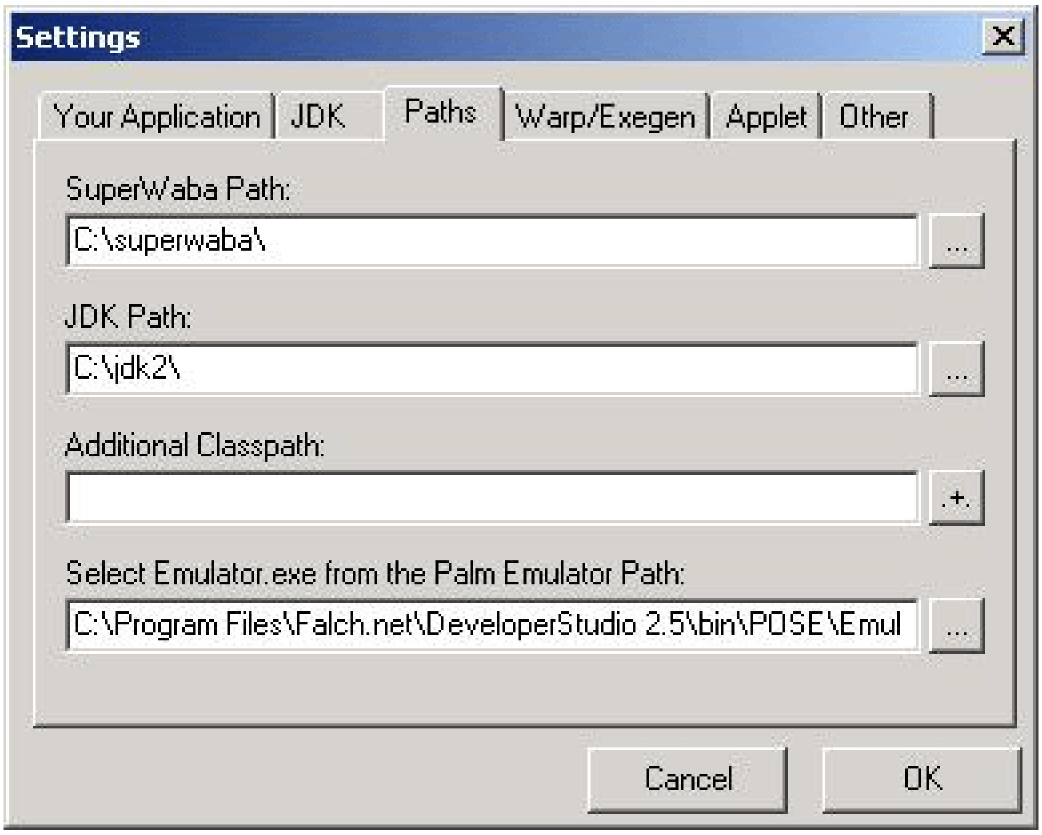
Motivation/purpose of sample

The second sample application is a port of an application from the first tutorial and is named `ibmdwgpspalm`. This application demonstrates a few basic user interface elements as well as the ever-important serial communications required for interacting with a serial GPS device. As mentioned earlier, this application employs the SuperWaba development toolset along with the Mobile Creator IDE. While the source code presented here is not a direct port of the earlier code base, the functionality is very similar. The earlier application utilized the Abstract Windowing Toolkit, a package that is not readily available for the PDA environment. Basic elements of the `waba.ui` package are responsible for the user interface elements of this application. Also, the lack of a suitable `javax.comm` implementation for Palm OS leads to the use of the `SerialPort` class from the `waba.io` package.

The next panel walks through the installation of the required toolsets.

Getting started

The three software components required for this application are the Mobile Creator, the SuperWaba SDK, and the latest JDK from Sun Microsystems. Tools on page 4 contains links for obtaining each of these tools. Once they have been installed, the Mobile Creator has a simple configuration setting dialog to fill out. This configuration tells Mobile Creator where to find the SuperWaba installation and the JDK's Java compiler, `javac.exe`. Here is a screenshot demonstrating a valid configuration:



SuperWaba Path: C:\superwaba\

JDK Path: C:\jdk2\

Additional Classpath:

Select Emulator.exe from the Palm Emulator Path: C:\Program Files\Falch.net\DeveloperStudio 2.5\bin\POSE\Emul

The next panel details the steps involved in the creation of a Mobile Creator/SuperWaba application.

Using Mobile Creator

The Mobile Creator hides all of the "moving parts" associated with the SuperWaba development cycle. In a nutshell, here are the steps involved in the creation of a Mobile Creator/SuperWaba application:

1. Editing of a textual source file -- in this case `ibmdwgpspalm.java`. This file contains valid Java source referencing introduced by the Waba packages earlier (`waba.io`, `waba.ui`, etc.)
2. The F-10 key initiates the build process. The source file is compiled via the JDK's Java compiler.
3. Assuming no errors were encountered, the Mobile Creator invokes two other tools -- warp and ExeGen. The warp tool packages the resulting Java class(es) into a Palm database file, `ibmdwgpspalm.prb`, in this case. ExeGen generates a Palm PRC file, `ibmdwgpspalm.prc`, containing the application's CreatorId and icons.
4. Mobile Creator finally initiates a session of the POSE and loads the application, ready to run and test.

The next panel highlights some of the application's user interface features and structure.

User interface elements, Part 1

A SuperWaba application is an extension of the `MainWindow SuperWaba` class. The constructor function `ibmdwgpspalm` initiates the user interface elements:

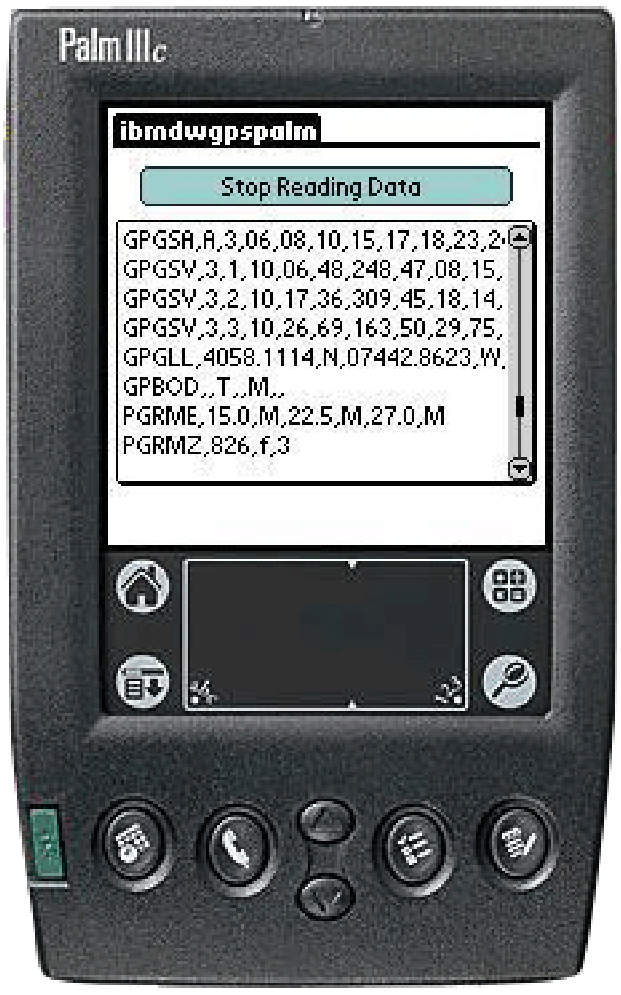
```
public class ibmdwgpspalm extends MainWindow {
   Button btnStart = null;
   ListBox lb = null;
   StringBuffer sb = new StringBuffer();
   SerialPort sp = null;
   boolean running = false;
   boolean eatchecksum = false;
   Timer savetimer = null;

   public ibmdwgpspalm()
   {
      setDoubleBuffer(true);
      setBorderStyle(TAB_ONLY_BORDER);
      setTitle("ibmdwgpspalm");
      add(btnStart = new Button("Start Reading Data"),10,20);
      add(lb = new ListBox(),LEFT+5,AFTER-10);
      lb.setRect(1,40,158,100);
      btnStart.setRect(10,20,140,15);
      Settings.setPalmOSStyle(true);
   }
```

User interface elements, Part 2

The user interface consists of a single button and listbox. The button controls the application, telling it to start or stop collecting data from the GPS unit. Valid NMEA sentences are displayed in the listbox. A help function named `AddString` places content into the listbox. This function ensures that the box's length stays manageable:

```
private void AddString(String s)
{
   if (lb.size() < 100)
   {
      lb.removeAll();
   }
   lb.add(s);
   lb.repaint();
}
```

Here is a screen shot of the application in action:



```
GPGSA,A,3,06,08,10,15,17,18,23,2
GPGSV,3,1,10,06,48,248,47,08,15,
GPGSV,3,2,10,17,36,309,45,18,14,
GPGSV,3,3,10,26,69,163,50,29,75,
GPGLL,4058.1114,N,07442.8623,W,
GPBOD,,T,,M,,
PGRME,15.0,M,22.5,M,27.0,M
PGRMZ,826,f,3
```

The next panel examines the `onEvent` method, reacting to user input.

Handling events

Once the button is tapped, the application moves into a mode for collecting data from the GPS unit. The following code is responsible for reacting to the button tap and initiating the communications session with the GPS, along with some other incidental housekeeping:

```
public void onEvent(Event event)
{
   switch (event.type)
   {
      case ControlEvent.PRESSED:
         if (event.target == btnStart)
         {
            if (!running)
            {
               sp = new SerialPort(0,4800);
               if (sp != null)
               {
                  sp.setReadTimeout(100);
                  sp.setFlowControl(false);
                  savetimer = addTimer(500);
                  running = true;
                  btnStart.setText("Stop Reading Data");
                  eatchecksum = false;
               }
               else
               {
                  popupModal(new MessageBox("ibmdwgpspalm","Failed to Start!"));
               }
            }
            else
            {
               sp.close();
               running = false;
               btnStart.setText("Start Reading Data");
               removeTimer(savetimer);
               sb.setLength(0);
            }
         }
      break;
      case ControlEvent.TIMER:
              ProcessPort();
      break;
   }
}
```

Things to note about this code include the manner in which the SerialPort instance, `sp`, is created and opened, as well as the `addTimer` method. `addTimer` causes the application to raise an event periodically -- in this case, every 500 milliseconds or one half second. The `ControlEvent.TIMER` event initiates the `ProcessPort` method responsible for actually interacting with the serial port. The SuperWaba SerialPort is polled at a frequency set by the parameter passed to `addTimer`. This functionality differs from the event-driven model of `javax.comm` employed in the earlier tutorials.

The next panel examines the `ProcessPort` method, responsible for retrieving information from the GPS unit and parsing it into NMEA sentences.

Interacting with the serial port

SuperWaba applications must poll the SerialPort, looking for newly received data. Due to this polling requirement, the `ProcessPort` method must continually gather data, looking for a valid NMEA sentence. A Java StringBuffer collects the incoming data as it is received. Each incoming character is examined in an effort to parse the data. A $ character marks the beginning of a sentence and a carriage-return, line-feed combination delimits the end of the sentence. The checksum portion of the sentence is silently ignored and not displayed. Here is the code to the `ProcessPort` method:

```
private void ProcessPort()
{
        if (sp == null)
        {
                AddString("port not open!");
                return;
        }
        int bytes = sp.readCheck();
        if (bytes > 0)
        {
                byte [] b = new byte[bytes];
                int bytesread = sp.readBytes(b,0,bytes);
               for (int i = 0;i<bytesread;i++)
                {
                  switch (b[i])
                  {
                   case '$':
                       sb.setLength(0);
                       break;
                   case '*':
                       eatchecksum = true;
                       break;
                   case 0x0d:
                       break;
                   case 0x0a:
                       eatchecksum = false;
                       AddString(sb.toString());
                       sb.setLength(0);
                       break;
                   default:
                       if (!eatchecksum)
                       {
                          sb.append((char) b[i]);
                       }
                       break;
                  }
             }
        }
        return;
}
```

Note the use of the `readCheck` method. This method indicates the presence of data available for reading. The data is accumulated into the StringBuffer variable, `sb`. When the full sentence is available, the `toString()` method of the StringBuffer returns a String representation of the NMEA sentence. This new string is passed to the `AddString` method for inclusion in the ListBox user interface element.

Section 7. Summary

Tutorial summary

This tutorial examined some of the challenges facing embedded and portable Java developers. The available toolset offerings cover a wide-spectrum of functionalities and techniques to get the job done; however, it is clear that the embedded Java developer does not share the same "write once" experience of the server side Java developer. From a GPS perspective, the basic theme of data collection holds true: The core requirement is a valid means to share data with a serial-based device. This requirement is not easily met with the J2ME offering, nor is it met with uniformity. Fortunately for the GPS enthusiast without a corporate budgetary backing, there is SuperWaba. This open source environment not only fits the bill, but it does so with traditional Java programming techniques, keeping things simple. One of the highlights of the second sample application is the fact that no Palm OS-specific knowledge is imparted in the code!

This tutorial also presented an alternative to the traditional data collection means of GPS applications by introducing a network based, MIDP applet using the WebSphere Studio Device Developer environment. Though simplistic in nature, it is a powerful design pattern, suitable for numerous applications, GPS and otherwise.

Regardless of the path chosen, Java technology and GPS have tremendous promise, particularly as GPS capabilities become built-in and more common in consumer devices and as the Java crowd thins out to expose a consensus on how to interact on these limited resource devices.

Tutorial resources

- The *http://wireless.java.sun.com/* Web site is home to the J2ME release of the Java language from Sun Microsystems.
- *http://java.sun.com/j2me/docs* contains the documentation relevant to the breadth of capable device profiles.
- This site, *http://www.palm-communications.com/ibmdw* , contains all of the source code presented in this tutorial.
- John Muchow's tutorial on MIDlets provides a nice introduction to the topic of MIDP/CLDC and MIDlet development. The tutorial may be found at: *https://www6.software.ibm.com/developerworks/education/wi-midlet/*
- The *http://www.redhat.com/embedded/technologies/ecos* Web site is the primary source of information pertaining to the embedded configurable operating system (eCos).
- The *http://www.eclipse.org* Web site is the home of the Eclipse IDE.
- *http://www.oreillynet.com/wireless/* regularly presents wireless development content, including GPS topics.
- The PalmOS emulator may be found at: *http://www.palmos.com/dev/tools/emulator*

- The introductory, prerequisite tutorials may be found at *Introduction to GPS, part 1* and *Introduction to GPS, part2*.
- A series of tutorials on Building Palm OS Applications, which discuss using a PDA for serial communications applications, is available on the developerWorks Wireless zone at *http://www-106.ibm.com/developerworks/wireless/* .
- Elliotte Rusty Harold's book titled *Java I/O* (O'Reilly and Associates) is a valuable resource to Java developers needing information and "How-To" knowledge in the realm of Input and Output streams in Java.
- The *http://www.jboss.org* Web site boasts the world's most popular open source application server.
- The Jbed environment may be found at *http://www.jbed.com*.
- Simplicity, a Rapid Application Development tool for the Palm OS platform using Java Technology, is available from *Data Representations*.

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.