Bluetooth for Java

BRUCE HOPKINS AND RANJITH ANTONY

Apress™

Bluetooth for Java
Copyright © 2003 by Bruce Hopkins and Ranjith Antony

ISBN (pbk): 1-59059-078-3

Printed and bound in the United States of America 12345678910

Technical Reviewer: Andrew Stringer
Editorial Directors: Dan Appleman, Gary Cornell, Simon Hayes, Karen Watterson, John Zukowski
Assistant Publisher: Grace Wong
Project Manager and Development Editor: Tracy Brown Collins
Copy Editor: Ami Knox
Compositor: Impressions Book and Journal Services, Inc.
Artist and Cover Designer: Kurt Krames
Indexer: Valerie Robbins
Production Manager: Kari Brooks
Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States, phone 1-800-SPRINGER, email `orders@springer-ny.com`, or visit `http://www.springer-ny.com`.
Outside the United States, fax +49 6221 345229, email `orders@springer.de`, or visit `http://www.springer.de`.

For information on translations, please contact Apress directly at 2560 9th Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax: 510-549-5939, email `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Downloads section.

CHAPTER 3

Before You Get Started

OKAY, NOW THAT YOU KNOW the ins and outs of Bluetooth, you're probably eager to find out how to integrate Bluetooth with Java. Well, this chapter is all about doing just that. But wait! Before you learn about how to use Bluetooth and Java, you need to know when it is not a good idea to use the two technologies together.

When NOT to Use Bluetooth and Java

You should not use Bluetooth with Java for the following purposes:

- Signal strength indicator
- Voice applications
- Distance measuring

The next few sections explain why to avoid those scenarios.

Signal Strength Indicator

Let's say that you have two Bluetooth units, and you want to know what the signal strength is between them. A good example is when you want to use the services of a network access point. A signal strength indicator would let you know if you were within range. Well, Java is not the ideal language for that sort of application because that kind of information is not exposed to the level where a JVM would have access to it. The JVM will let you know if you are within range or not within range; there is no middle ground. In this scenario, you're better off using a native language for your device such as C or C++.

Voice Applications

Now, you've already read Chapter 2, and you realize that Bluetooth is a really great technology because you have the ability to transmit voice and data information wirelessly to other Bluetooth devices. Suddenly, you get ideas bubbling in your head about how great it would be to create a speech-to-text application on your Bluetooth-enabled phone. Unfortunately, Java (especially J2ME) is not well suited to this arena just yet. Performance is a key factor in voice-based applications, and once again, in this case, you're better off using a native language such as C. However, this application may be feasible to do in Java if the Java Real-Time Technology can be incorporated.

Distance Measuring

The best wireless technology for accurately measuring distance is light waves and not radio signals. Light waves are direct, and the calculations can be pretty simple because the speed of light (in various mediums) is pretty well documented. Using radio signals to measure distance can be quite tricky, and one of the best ways to do that is to use triangulation, like GPS transceivers do. Whether you are using Java or C, Bluetooth might be a viable technology for triangulation, but definitely not for calculating or measuring accurate distances.

NOTE *The key word here is accurate. You can definitely use Bluetooth for proximity measurement (i.e., where in the building is Bruce Hopkins?). In fact, the Ericsson BlipNet does exactly that! See Chapter 11 for more information on the Ericsson BlipNet.*

So, to put it succinctly, you can only do what is possible using the constraints of the Bluetooth technology and what the JVM exposes to you. If the JVM only gives you access to the RFCOMM layer for communication, then you're stuck with it. If the OBEX layer is not exposed to the JVM, then don't expect to be able to send objects. To increase application portability, your Java Bluetooth vendor should implement the Java Bluetooth specification created through the JCP.

Understanding the JCP

The JCP is the Java Community Process, and it is the formal procedure to get an idea from a simple concept incorporated into the Java standard. This process allows developers and industry experts to shape the future of the Java standard. Popular APIs like Java USB, Java Real-Time, Java Printing, Java New I/O, J2ME MIDP 1.0, J2ME MIDP 2.0, JDBC 3.0, EJB 2.0, and even JDK 1.4 all went through the Java Community Process. If you want to add some new functionality to the Java language, or if you want to suggest a new API, or if you think that some new classes should have a package name of `java.*` or `javax.*`, then you need to go through the JCP.

The Role of the JSR-82

A JSR is a Java Specification Request in the Java Community Process. The JSR-82 is the formal JCP name for the Java APIs for Bluetooth. When a proposed JSR is approved, an Expert Group is formed by the specification lead. The specification lead for the JSR-82 was Motorola, and together with the JSR-82 Expert Group, they created the official Java Bluetooth APIs. The following companies participated in the JSR-82 Expert Group:

- Extended Systems
- IBM
- Mitsubishi
- Newbury Networks
- Nokia
- Parthus Technologies
- Research in Motion (RIM)
- Rococo Software
- Sharp Electronics
- Sony Ericsson
- Smart Fusion
- Smart Network Devices
- Sun Microsystems
- Symbian

- Telecordia
- Vaultus
- Zucotto

The JSR-82 Expert Group also had three individual experts: Peter Dawson, Steven Knudsen, and Brad Threatt.

What Is the RI and TCK?

According to the Java Community process, the specification lead company is responsible for creating a Reference Implementation (RI) and also a Technology Compatibility Kit (TCK). The Reference Implementation is basically a proof of concept to prove that the specification can be implemented. Other companies are free to implement the JSR-82, and in order to certify that their vendor kit is compliant to the JSR-82 standard, that vendor's product must pass the TCK.

The JSR-82 specification actually has two Reference Implementations and Technology Compatibility Kits. Why did they do this? Recall in Chapter 2 that the Bluetooth SIG has adopted some preexisting protocols in the Bluetooth specification, namely OBEX. The OBEX protocol was used with infrared technology for object transmissions long before Bluetooth was even invented. The designers of the Java Bluetooth specification decided not to tie OBEX to Bluetooth when creating the Java Bluetooth standard. Therefore, the JSR-82 actually consists of two independent packages:

- `javax.bluetooth` (the 13 classes and interfaces that are needed to perform wireless communication with the Bluetooth protocol)
- `javax.obex` (the 8 classes that are needed to send objects between devices, independent of the transport mechanism between them)

So, to answer your next question, yes, you can use OBEX without Bluetooth. Bluetooth is simply one of many transports with which OBEX can operate.

The classes and interfaces that comprise the Java Bluetooth specification are briefly described in Tables 3-1 and 3-2. These classes and their methods are covered as needed in the following chapters, and their APIs are listed in detail in Appendix A and Appendix B.

Table 3-1. Classes in the javax.bluetooth Package

CLASS NAME	**DESCRIPTION**
`DiscoveryListener`	The `DiscoveryListener` interface allows an application to receive device discovery and service discovery events.
`L2CAPConnection`	The `L2CAPConnection` interface represents a connection-oriented L2CAP channel.
`L2CAPConnectionNotifier`	The `L2CAPConnectionNotifier` interface provides an L2CAP connection notifier.
`ServiceRecord`	The `ServiceRecord` interface describes characteristics of a Bluetooth service.
`DataElement`	The `DataElement` class defines the various data types that a Bluetooth service attribute value may have.
`DeviceClass`	The `DeviceClass` class represents the class of device (CoD) record as defined by the Bluetooth specification.
`DiscoveryAgent`	The `DiscoveryAgent` class provides methods to perform device and service discovery.
`LocalDevice`	The `LocalDevice` class represents the local Bluetooth device.
`RemoteDevice`	The `RemoteDevice` class represents a remote Bluetooth device.
`UUID`	The `UUID` class defines universally unique identifiers.
`BluetoothConnectionException`	This `BluetoothConnectionException` is thrown when a Bluetooth connection (L2CAP, RFCOMM, or OBEX) cannot be established successfully.
`BluetoothStateException`	The `BluetoothStateException` is thrown when a request is made to the Bluetooth system that the system cannot support in its present state.
`ServiceRegistrationException`	The `ServiceRegistrationException` is thrown when there is a failure to add a service record to the local Service Discovery Database (SDDB) or to modify an existing service record in the SDDB.

Table 3-2. Classes in the javax.obex Package

CLASS NAME	DESCRIPTION
`Authenticator`	This interface provides a way to respond to authentication challenge and authentication response headers.
`ClientSession`	The `ClientSession` interface provides methods for OBEX requests.
`HeaderSet`	The `HeaderSet` interface defines the methods that set and get the values of OBEX headers.
`Operation`	The `Operation` interface provides ways to manipulate a single OBEX PUT or GET operation.
`SessionNotifier`	The `SessionNotifier` interface defines a connection notifier for server-side OBEX connections.
`PasswordAuthentication`	This class holds user name and password combinations.
`ResponseCodes`	The `ResponseCodes` class contains the list of valid response codes a server may send to a client.
`ServerRequestHandler`	The `ServerRequestHandler` class defines an event listener that will respond to OBEX requests made to the server.

The Benefits of the Java Bluetooth API

There are two key advantages to using the official Java Bluetooth API versus a C-based (or native) API:

- API is independent of the stack and radio
- Standardized Bluetooth API

API Is Independent of Stack and Radio

So what makes the official Java Bluetooth API better than a C/C++ Bluetooth API? One of the principle reasons is that the JSR-82 API is independent of the stack and the Bluetooth hardware. That gives you the ability to write applications without any knowledge of the underlying Bluetooth hardware or stack. And that's essentially what Java gives you today. If you write standard Java code (without any native methods), you can run your code on basically any hardware platform and on any OS with little or no modification. Whether it's an appli-cation, applet, midlet, servlet, or EJB, you can code your application on one platform and deploy to another platform.

The Only Standardized Bluetooth API

If you have a C/C++-based Bluetooth SDK, then you are basically at the mercy of the vendor. There is no standard for a C/C++-based Bluetooth SDK, so each vendor is free to name functions and methods to whatever they choose. Vendor A may have five profiles in its SDK, and Vendor B may only have three. If you want to change Bluetooth hardware or stack libraries, then you'll need to rewrite your Bluetooth application and/or change its functionality. Because the JSR-82 is the official Java API for Bluetooth, all vendors who implement the standard must include a core set of layers and profiles in their Bluetooth SDK.

A JSR-82–compliant Bluetooth stack must include the following layers:

- Host Controller Interface (HCI)
- Logical Link Control and Adaptation Protocol (L2CAP)
- Service Discovery Protocol (SDP)
- RFCOMM

These profiles are also required:

- Generic Access Profile
- Service Discovery Application Profile
- Serial Port Profile
- Generic Object Exchange Profile

> **CROSS-REFERENCE** *See "The Bluetooth Protocol Stack" and "Profiles" in Chapter 2 for details on the Bluetooth protocol stack and profiles just in case you forgot.*

The first thing that may come to your mind is, "Hey, wait a minute, doesn't the Bluetooth specification contain more profiles than that? Why did they implement only a few profiles in Java?" Well, here are two major reasons:

First of all, the JSR-82 team wanted to get the Java Bluetooth specification in the hands of developers as quickly as possible. Recall in Chapter 2 that Bluetooth profiles are designed to be functional enough where higher profiles extend the functionality of the lower, or base, profiles. Refer to Figure 2-9, which shows a diagram of the relationship of the profiles of the Bluetooth specification.

Secondly, by implementing the base profiles (Generic Access Profile, Service Discovery Application Profile, Serial Port Profile, and Generic Object Exchange Profile), the SDK vendor or the application developer is free to implement the higher profiles of the Bluetooth specification.

What You Need to Get Started

We know that this question has been on your mind for a while. Well, here's a list of what you'll need:

- Bluetooth devices (at least two)
- Bluetooth host (at least one)
- Bluetooth stack
- Java Bluetooth API

Now let's cover all these components in detail and describe how they all work together.

Bluetooth Devices

Bluetooth devices were covered in Chapter 2, but just in case you forgot, take another look at Figures 2-1, 2-2, and 2-3. Remember, Bluetooth devices are simply radios, so getting a single device is just like getting a single walkie talkie; it's pretty useless. If your Bluetooth device is point-to-point capable, then that means it can only talk to a single Bluetooth device at a time. If it is multipoint capable, then it can talk to up to seven devices at a time. The Bluetooth device is also known as the *controller*.

Bluetooth Host

The Bluetooth *host* is the computer that is physically connected to the Bluetooth device. For the most part, this is your desktop PC, laptop, PDA, or smart phone. Usually, the connection is USB, RS-232, or UART.

Now, you are definitely going to need two Bluetooth devices, but you can get away with having only one Bluetooth host. How does this work? Well, if you have a PC that has two serial ports or two USB ports (or both), then you can connect both of your Bluetooth devices to your PC's ports. In order for this to work, you need to start two instances of your JVM; each JVM will have its own Bluetooth device.

The Bluetooth host must meet the minimum requirements for the CLDC, so you need at least 512k total memory for the JVM.

Bluetooth Stack

A Bluetooth stack is required in order for a Bluetooth host (the PC) to properly communicate to the Bluetooth device (the controller). If you go back to Figure 2-6, which shows a diagram of the Bluetooth stack, the bottom layer of the stack is the Host Controller Interface! See, it does make sense. The Host Controller Interface is literally the software required to interface the Bluetooth host and the Bluetooth device (the controller).

Since this book is all about Java and Bluetooth, you might think that the Bluetooth stack needs to be written completely in the Java language. Well, not exactly. Some Bluetooth vendors have implemented a completely all-Java stack, while others have implemented a Java interface (i.e., JNI or other means) to a native stack. Either way, you need to access the stack through Java code, whether or not the stack is in Java.

Java Bluetooth API

Finally, you're going to need a set of libraries to interface with your stack. For the most part, a company will sell you a Java Bluetooth API and Bluetooth stack together in a kit. Just be sure to ask them what Bluetooth devices their kit supports.

Another question to ask your Java Bluetooth kit vendor is if their product is JSR-82 compliant. Currently, JSR-82 can only be implemented on the J2ME platform. JSR-82 cannot be implemented on the J2SE platform because the J2SE does not support the Generic Connection Framework. Hopefully, the Generic Connection Framework will be implemented by JDK 1.5.

NOTE *The official JSR to implement the GCF in the JDK is JSR-197.*

Does this mean that it is impossible to do Java and Bluetooth development on the J2SE platform? No, it simply means that whatever Java Bluetooth kit that you obtain for J2SE will not be compliant with JSR-82 until the Generic Connection Framework is implemented in J2SE. The major ramification of this problem is that your J2ME and J2SE code may be drastically different from each other, even if you are doing the same thing.

Java Bluetooth Vendor SDKs

So, who's offering Java Bluetooth SDKs, and which are JSR-82 compliant? Fortunately, there is a plethora of Java Bluetooth SDKs to fit the needs that your application requires. Vendor support is available for Java Bluetooth development on a wide range of operating systems and JVM platforms. Table 3-3 displays various attributes of many Java Bluetooth SDKs.

*Table 3-3. Java Bluetooth SDK Vendors**

COMPANY NAME	JSR-82 JAVAX.BLUETOOTH SUPPORT	JSR-82 JAVAX.OBEX SUPPORT	SUPPORTED JAVA PLATFORMS	SUPPORTED OPERATING SYSTEMS
Atinav	Yes	Yes	J2ME, J2SE	Win-32, Linux, Pocket PC
BlueGiga	No	No	Waba JVM	uClinux
Ericsson	No	No	J2SE	Win-32, Linux
Esmertec	Yes	No	J2ME	Win-32, Palm OS, Pocket PC, many others
Harald	No	No	J2SE	Win-32, Linux, others
Possio	Yes	Yes	J2ME	Win-32, Linux
Rococo	Yes	Yes	J2ME, J2SE	Win-32, Linux, Palm OS, Pocket PC
Smart Network Devices	Yes	No	J2ME	HyNetOS
SuperWaba	No	No	Waba JVM	Palm OS
Zucotto	No	No	J2ME, J2SE	Win-32

* The information in this table is subject to change, so check the companion Web site `http://www.javabluetooth.com` for up-to-date information. Palm OS is a registered trademark of Palm, Inc.

Summary

This chapter has only skimmed the surface of how to integrate Java Bluetooth. You learned about the advantages of using Java versus C for application development. You also learned about JSR-82 as well as what it takes to get things up and running.

In the next chapter, we'll focus more on integrating Java and Bluetooth, as well as introduce some example code.