**[Login](https://java.net/people/login?original_uri=http://today.java.net/pub/a/today/2004/07/27/bluetooth.html)**[**Join Now**](http://java.net/people/new)[**Help**](http://java.net/projects/help/pages/Home)

**[Java.net logo](http://www.java.net/)**

*The Source for Java Technology Collaboration*

Top of Form

ForumsBlogsProjectsPeople



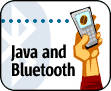
Bottom of Form

* [**Home**](http://www.java.net/)
* [**Projects**](http://www.java.net/projects/community)
* [**Forums**](http://www.java.net/forum)
* [**People**](http://beta.java.net/people)
* [**Java User Groups**](http://www.java.net/jugs/java-user-groups)
* [**JCP**](http://today.java.net/community/jcp)

**Getting Started with Java and Bluetooth**

July 27, 2004

[Bruce Hopkins](http://today.java.net/pub/au/220)

****

your PC. Or perhaps you've heard a lot about Bluetooth, but you aren't  
sure what exactly you can do with it. In either case, you've had some  
exposure to Bluetooth, and now you're ready to start flexing your programming  
muscles with the technology. Great! The purpose of this article is to give you  
a good introduction to the Bluetooth protocol, including an overview of its  
protocol layers and profiles. We'll also cover the the classes and methods  
of JSR-82, the official Java Bluetooth API. Finally, we'll wrap things  
up by describing what software that you'll need in order to get started.

**What is Bluetooth?**

What exactly is Bluetooth? Well, simply stated, Bluetooth is a wireless communication  
protocol. Since it's a communication protocol, you can use Bluetooth to communicate  
to other Bluetooth-enabled devices. In this sense, Bluetooth is like any other  
communication protocol that you use every day, such as HTTP, FTP, SMTP, or IMAP. Bluetooth  
has a client-server architecture; the one that initiates the connection is the  
client, and the one who receives the connection is the server. Bluetooth is a  
great protocol for wireless communication because it's capable of transmitting  
data at nearly 1MB/s, while consuming 1/100th of the power of Wi-Fi.

In order for Bluetooth devices to communicate properly, they all need to  
conform to the Bluetooth specification. The Bluetooth specification, like any  
other spec, defines the standard that a Bluetooth device should adhere to, as  
well as rules that need to be enforced when communicating. You can download  
the specification documents at the official [Bluetooth  
web site](http://www.java.net/external?url=http://www.bluetooth.com). The Bluetooth protocol stack and profiles together comprise the  
Bluetooth specification.

**The Bluetooth Protocol Stack**

The Bluetooth stack is the software or firmware component that has direct access  
to the Bluetooth device. It has control over things such as device settings, communication  
parameters, and power levels for the Bluetooth device. The stack itself consists  
of layers, and each layer of the stack has a specific task in the overall functionality  
of the Bluetooth device. Since Bluetooth device manufacturers are not required to use all of the layers in the stack, we're only going to cover the main ones that are implemented in almost every Bluetooth device.

* HCI is the Host Controller Interface. This layer is the interface between  
  the radio and the host computer.
* L2CAP stands for Logical Link Controller Adaptation Protocol. This layer  
  is the multiplexer of all data passing through the unit. Audio signals, however,  
  have direct access to the HCI.
* SDP is the Service Discovery Protocol. The SDP layer is used to find services  
  on remote Bluetooth devices.
* RFCOMM is widely known as the virtual serial port protocol.
* OBEX is the object exchange protocol.

**Bluetooth Profiles**

Bluetooth Profiles were created to allow different Bluetooth devices to interoperate.  
For instance, let's say that you own a Bluetooth-enabled PDA and a Bluetooth-enabled wireless phone. Both devices have Bluetooth stacks. How can you  
tell if those two devices will allow you to synchronize the phone lists between  
each other? How will you know if you can send a phone number from the PDA to  
the phone? And most importantly, how can you determine if these devices will  
allow you to browse the Internet from the PDA, using the phone as a wireless modem?

A Bluetooth profile is a designed set of functionality for Bluetooth devices.  
For instance, using the examples listed above, the phone and the PDA must both  
support the Synchronization Profile in order to synchronize data between them.  
In order to send object data such as a .vcf file from the PDA to the phone, then both  
devices need to have the Object Push Profile implemented. Finally, the PDA  
and the wireless phone must both support the Dialup Networking Profile in order  
for the PDA to wirelessly browse the Internet via the phone. If you want your  
Bluetooth-enabled devices to interact, having a Bluetooth stack is not good  
enough -- they also need to conform to a particular profile.

A word of caution here: do not get Bluetooth profiles confused with J2ME profiles. J2ME profiles are a set of Java classes that extend the functionality of a J2ME  
Configuration. For instance, the MID Profile is a set of Java classes that extend  
the functionality of the Connected Limited Device Configuration. On the other  
hand, a Bluetooth profile can be implemented in any language and on any platform,  
because it refers to a defined set of functionality for a Bluetooth-enabled  
device. So the Object Push Profile can be implemented on a Palm OS PDA in C++,  
and can be implemented on a Bluetooth-enabled printer in assembly language.  
For those of you who are familiar with RUP methodology, Bluetooth Profiles are  
also called Bluetooth Use Cases.

**Java Bluetooth Application Concepts**

The basic concepts of any Bluetooth application (Java or otherwise) consist  
of the following components:

* Stack Initialization
* Device Discovery
* Device Management
* Service Discovery
* Communication

The Java Bluetooth Specification adds a special component to the mix called  
the Bluetooth Control Center (BCC), which is outside of the scope of this article.

**Stack Initialization**

Before you can do anything, you need to initialize your stack. Remember,  
the stack is the piece of software (or firmware) that controls your Bluetooth  
device. Stack initialization can consist of a number of things, but its main  
purpose is to get the Bluetooth device ready to start wireless communication.  
Every vendor handles stack initialization differently, so we'll cover how to  
initialize the stack using the Atinav Java Bluetooth SDK.

import javax.bluetooth.\*;  
import javax.microedition.io.\*;  
import com.atinav.BCC;  
  
public class WirelessDevice implements DiscoveryListener {  
    LocalDevice localDevice = null;   
      
    public WirelessDevice (){   
        //setting the port number using Atinav's BCC  
        BCC.setPortName("COM1");   
          
        //setting the baud rate using Atinav's BCC  
        BCC.setBaudRate(57600);  
          
        //connectable mode using Atinav's BCC  
        BCC.setConnectable(true);  
          
        //Set discoverable mode using Atinav's BCC   
        BCC.setDiscoverable(DiscoveryAgent.GIAC);   
          
        try{  
            localDevice = LocalDevice.getLoaclDevice();   
        }  
        catch (BluetoothStateException exp) {  
        }  
          
        // implementation of methods in DiscoveryListener class  
        // of javax.bluetooth goes here  
          
        // now do some work  
    }  
}

**Device Management**

LocalDevice and RemoteDevice are the two main classes  
in the Java Bluetooth Specification that allow you to perform Device Management.  
These classes give you the ability to query statistical information about  
your own Bluetooth device (LocalDevice) and information  
on the devices in the area (RemoteDevice). The static method LocalDevice.getLocalDevice()  
returns an instantiated LocalDevice object for you to use. In order to get the  
unique address of your Bluetooth radio, just call getBluetoothAddress() on your  
local device object. The Bluetooth address serves the same purpose of the MAC  
address on the network card of your computer; every Bluetooth device has a unique  
address. If you want other Bluetooth devices in the area to find you, then call  
the setDiscoverable() method in LocalDevice object.

In a nutshell, that's about all it takes to perform Device Management with the  
Java Bluetooth Specification APIs. Now, let's take a look at the concept in Bluetooth  
that allows you to discover other Bluetooth devices: device discovery.

**Device Discovery**

Your Bluetooth device has no idea of what other Bluetooth devices are in  
the area. Perhaps there are laptops, desktops, printers, mobile phones, or PDAs  
in the area. Who knows? The possibilities are endless. In order to find out,  
your Bluetooth device will use the Device Discovery classes that are provided  
into the Java Bluetooth API in order to see what's out there.

Let's take a look at the two classes needed in order for your Bluetooth  
device to discover remote Bluetooth devices in the area: DiscoveryAgent and  
DiscoveryListener.

After getting a LocalDevice object, just instantiate a DiscoveryAgent  
by calling LocalDevice.getDiscoveryAgent().

LocalDevice localdevice = LocalDevice.getLocalDevice();   
DiscoveryAgent discoveryagent = localdevice.getDiscoveryAgent();

The are multiple ways to discover remote Bluetooth devices, but to be brief,  
I'll just show you one particular way. First, your object must implement the  
DiscoveryListener interface. This interface works like any listener, so it'll  
notify you when an event happens. In this case, you'll be notified when Bluetooth  
devices are in the area. In order to start the discovery process, just call  
the startInquiry() method on your DiscoveryAgent. This method is non-blocking,  
so you are free to do other things while you wait for other Bluetooth devices  
to be found.

When a Bluetooth device is found, the JVM will call the deviceDiscovered()  
method of the class that implemented the DiscoveryListener interface.  
This method will pass you a RemoteDevice object that represents the device discovered by  
the inquiry.

**Service Discovery**

Now that you know how to find other Bluetooth devices, it would be really nice  
to see what services that those devices offer. Of course, if the RemoteDevice  
is a printer, then you know that it can offer a printing service. But what if  
the RemoteDevice is a computer? Would it readily come to mind that you can also print to a  
printer server?

That's where Service Discovery comes in. You can never be sure what services  
a RemoteDevice may offer; Service Discovery allows you to find out what they are.

Service Discovery is just like Device Discovery in the sense that you use the  
DiscoveryAgent to do the "discovering." The searchServices()  
method of the DiscoveryAgent class allows you to search for services  
on a RemoteDevice. When services are found, the servicesDiscovered()  
will be called by the JVM if your object implemented the DiscoveryListener  
interface. This callback method also passes in a ServiceRecordobject  
that pertains to the service for which you searched. With a ServiceRecord  
in hand, you can do plenty of things, but you would most likely would want to  
connect to the RemoteDevice where this ServiceRecord originated:

String connectionURL = servRecord[i].getConnectionURL(0, false);

**Service Registration**

Before a Bluetooth client device can use the Service Discovery on a Bluetooth  
server device, the Bluetooth server needs to register its services internally  
in the Service Discovery database (SDDB). That process is called Service Registration.  
This section will discuss what's involved for Service Registration for a Bluetooth  
device, and I'll also give you a rundown of the classes needed to accomplish this.

Note: In a peer-to-peer application, such as a file transfer or chat application,  
be sure to remember that any device can act as the client or the server, so  
you'll need to incorporate that functionality (both client and server) into  
your code in order to handle both scenarios of Service Discovery (i.e., the client)  
and Service Registration (i.e., the server). Here's a scenario of what's involved  
to get your service registered and stored in the SDDB.

1. Call Connector.open() and cast the resulting Connection to a StreamConnectionNotifier.
   1. Connector.open() creates a new ServiceRecord and sets some attributes.
2. Use the LocalDevice object and the StreamConnectionNotifier to obtain the ServiceRecord that was created by the system.
3. Add or modify the attributes in the ServiceRecord (optional).
4. Use the StreamConnectionNotifier and call acceptAndOpen() and wait for Bluetooth clients to discover this service and connect.
   1. The system creates a service record in the SDDB.
5. Wait until a client connects.
6. When the server is ready to exit, call close() on the StreamConnectionNotifier.
   1. The system removes the service record from the SDDB.

StreamConnectionNotifier and Connector both come from the javax.microedition.io  
package of the J2ME platform. The code that accomplishes the above task is shown  
below in the following snippet:

// lets name our variables  
StreamConnectionNotifier notifier = null;  
StreamConnection sconn = null;  
LocalDevice localdevice = null;  
ServiceRecord servicerecord = null;  
  
// step #1  
// the String url will already be defined with the   
// correct url parameters  
notifier = (StreamConnectionNotifier)Connector.open(url);  
  
// step #2  
// we will get the LocalDevice if not already done so  
localdevice = LocalDevice.getLocalDevice();  
servicerecord = localdevice.getRecord(notifier);  
  
// step #3 is optional  
  
// step #4  
// this step will block the current thread until  
// a client responds this step will also cause the  
// service record to be stored in the SDDB  
notifier.acceptAndOpen();  
  
// step #5  
// just wait...  
// assume the client has connected and you are ready to exit  
  
// step #6  
// this causes the service record to be removed   
// from the SDDB  
notifier.close();

And that's all that you need to do Service Registration in Bluetooth. The next  
step is Communication.

**Communication**

Bluetooth is a communication protocol, so how do you communicate with it?  
Well, the Java Bluetooth API gives you three ways to send and receive data,  
but for right now, we'll cover only one of them: RFCOMM.

Note: RFCOMM is the protocol layer that the Serial Port Profile uses in order  
to communicate, but these two items are almost always used synonymously.

**Server Connections with the Serial Port Profile**

The code listing below demonstrates what is needed to open a connection on a Bluetooth  
device that will act as a server.

// let's name our variables  
  
StreamConnectionNotifier notifier = null;  
StreamConnection con = null;  
LocalDevice localdevice = null;  
ServiceRecord servicerecord = null;  
InputStream input;  
OutputStream output;  
  
// let's create a URL that contains a UUID that   
// has a very low chance of conflicting with anything  
String url =   
  &quot;btspp://localhost:00112233445566778899AABBCCDDEEFF;name=serialconn&quot;;  
// let's open the connection with the url and  
// cast it into a StreamConnectionNotifier  
notifier = (StreamConnectionNotifier)Connector.open(url);  
  
// block the current thread until a client responds  
con = notifier.acceptAndOpen();  
  
// the client has responded, so open some streams  
input = con.openInputStream();  
output = con.openOutputStream();  
  
// now that the streams are open, send and  
// receive some data

For the most part, this looks like just about the same code used in Service Registration,  
and in fact, it is! Service Registration and Server Communication are both accomplished  
using the same lines of code. Here's a few items that I want to point out. The  
String url begins with btspp://localhost:, which is  
required if you're going to use the Bluetooth Serial Port Profile. Next comes  
the UUID part of the URL, which is 00112233445566778899AABBCCDDEEFF.  
This is simply a custom UUID that I made up for this service; I could have chosen  
any string that was either 32 bits or 128 bits long. Finally, we have ;name=serialconn  
in the url String. I could have left off this part, but I want my custom service  
to have a name, so the actual service record in the SDDB has the following entry:

ServiceName = serialconn

The implementation has also assigned a channel identifier to this service.  
The client must provide the channel number along with other parameters in order  
to connect to a server.

**Client Connections with the Serial Port Profile**

Establishing a connection with the Serial Port Profile for a J2ME client is  
simple because the paradigm hasn't changed for J2ME I/O. You simply call Connector.open().

StreamConnection con =(StreamConnection)Connector.open(url);

You obtain the url String that is needed to connect to the device from the  
ServiceRecord object that you get from Service Discovery. Here's a more  
complete listing of code that will show you how a Serial Port Profile client  
makes a connection to a Serial Port Profile server.

String connectionURL = serviceRecord.getConnectionURL(0, false);  
StreamConnection con =(StreamConnection)Connector.open(connectionURL);

What does a SPP client connection URL look like? If the address of the server  
is 0001234567AB, the String that the SPP client would look something like this:

btspp://0001234567AB:3

The 3 at the end of the url String is the channel number that the server assigned  
to this service when this service was added to the SDDB.

**Java Bluetooth Development Kits**

The most widely available development kit for Java Bluetooth applications is  
the [J2ME  
Wireless Toolkit 2.2](http://www.java.net/external?url=http://java.sun.com/products/j2mewtoolkit/download-2_2.html) from Sun. It incorporates a Bluetooth network simulator,  
and has support for OBEX. And best of all, it's free! The current version of  
the J2ME Wireless Toolkit is available on Windows platforms.

If you're targeting JSR-82-enabled Nokia phones, such as the 6600, then you  
may also want to try out the [Nokia  
Developer's Suite 2.1](http://www.java.net/external?url=http://www.forum.nokia.com/main/0,,034-2,00.html). Much like Sun's Wireless Toolkit, the Nokia Developer's  
Suite is free and it also includes a Bluetooth network simulator. The Nokia  
Developer's Suite supports Windows and Linux platforms.

SonyEricsson also makes a free development kit for its P900 Java Bluetooth-enabled phone, which can be found at their [developer site](http://www.java.net/external?url=http://developer.sonyericsson.com).

Atinav makes one of the most comprehensive [JSR-82  
implementations and developer kits](http://www.java.net/external?url=http://www.avelink.com/Bluetooth/Products/JSR-82/index.htm) with support for J2ME CLDC, J2ME CDC,  
and J2SE devices. They support numerous RS-232, UART, USB, CF, and PCMCIA Bluetooth  
devices. Their solution is based on an all-Java stack, and their SDK includes  
the following profiles: GAP, SDAP, SPP, OBEX, FTP, Sync, OPP, Fax, and Printing  
-- whew! They make the only JSR-82 implementation for the PocketPC platform,  
and also support Windows and Linux.

Possio makes a [JSR-82  
development kit](http://www.java.net/external?url=http://www.possio.com/scripts/split.asp?cat=3&prod=wireless&dynfile=wirelessgateways&id=english&dh=3) that complements their Java Bluetooth-enabled access point,  
the PX30. The PX30 is a Linux-based access point, and is powered by an Intel XScale  
processor. It includes Wi-Fi, Bluetooth, and the CDC Foundation Profile.

[Rococo](http://www.java.net/external?url=http://www.rococosoft.com) is famous for making the first  
Java Bluetooth Simulator, although they also make a Java Bluetooth developer  
kit for the Palm OS 4 platform. The simulator is currently priced at $1000, and supports the following profiles:  
GAP, SDAP, SPP, and GOEP.

[Avetana](http://www.java.net/external?url=http://www.avetana-gmbh.de/avetana-gmbh/jsr82.xml)  
is a German company that makes the only JSR-82 implementation for the Mac OS  
X platform. They also provide JSR-82 implementations for Windows and Linux.

**Summary**

What have we learned here? Hopefully, you should have a good understanding  
of what Bluetooth is and how to use it. Before you start communicating to other  
Bluetooth devices, you need to discover the devices in your vicinity, and search  
for their services. After all of the preliminaries are out of the way, you can  
stream data back and forth to any Bluetooth-enabled device in your area, whether  
it's running Java or not.

With over one million Bluetooth-enabled devices shipping per week (that's  
right, one million devices per week), there's a lot of PDAs, cell phones, laptops,  
desktops, access points, cameras, keyboards, mice, printers, audio players,  
and vehicles out there for your mobile Java apps to play with!

|  |
| --- |
|  |

[Bruce Hopkins](http://today.java.net/pub/au/220) author of [Bluetooth for Java](http://www.java.net/external?url=http://www.javabluetooth.com/), is an enthusiast for mobile, embedded, and wireless application development. He's currently working for the startup [BlogRadio](http://www.java.net/external?url=http://www.podblogr.com/). Bruce is also a [Java Champion](https://java-champions.dev.java.net/).

**Related Topics >>**[**Mobility**](http://today.java.net/articles/topic/19)**|**

**Article Links >>**

* [Login](http://today.java.net/user/login?destination=comment%2Freply%2F219503%23comment-form) or [register](http://today.java.net/user/register?destination=comment%2Freply%2F219503%23comment-form) to post comments

* [Printer-friendly version](http://today.java.net/print/219503)

* [ShareThis](javascript:void(0))

* 64333 reads

**Comments**

[**Hey, so how would a stack initialization for the bluecove ...**](http://today.java.net/pub/a/today/2004/07/27/bluetooth.html#comment-818950)

by elvelon - 2011-10-06 03:02

Hey,

so how would a stack initialization for the bluecove work? I unfortunately couldn't figure this out by myself...

* [Login](http://today.java.net/user/login?destination=comment%2Freply%2F219503%23comment-form) or [register](http://today.java.net/user/register?destination=comment%2Freply%2F219503%23comment-form) to post comments

**Get Involved**

* [**About Java.net**](http://today.java.net/about-javanet)
* [**Adopt a JSR**](http://java.net/projects/jugs/pages/AdoptAJSR)
* [**Create a Project**](http://today.java.net/create-project)
* [**Java.net Enhancements**](http://today.java.net/enhancements-list)

**Get Informed**

* [**Articles**](http://today.java.net/articles)
* [**Blogs**](http://today.java.net/blogfront)
* [**Events**](http://today.java.net/events)
* [**Java Magazine**](http://www.oracle.com/technetwork/java/javamagazine/index.html?origref=http://java.net/projects/java-magazine)
* [**Oracle University**](http://today.java.net/java-training-and-certification-oracle-university)
* [**Feedback**](http://today.java.net/contact)

* [**FAQ**](http://today.java.net/javanet-faq)

* [**Terms of Use**](http://today.java.net/javanet-web-site-terms-use)

* [**Privacy**](http://www.java.net/external?url=http://www.oracle.com/us/legal/privacy/index.html)

* [**Trademarks**](http://www.java.net/external?url=http://www.oracle.com/us/legal/third-party-trademarks/index.html)

Your use of this web site or any of its content or software indicates your agreement to be bound by these [Terms of Participation](http://today.java.net/javanet-web-site-terms-use).

Copyright © 2012, Oracle and/or its affiliates. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

[http://today.java.net/sites/all/themes/java_adaptive/images/oracle.png](http://www.java.net/external?url=http://www.oracle.com)[http://today.java.net/sites/all/themes/java_adaptive/images/kenai.png](http://www.java.net/external?url=http://kenai.com)[http://today.java.net/sites/all/themes/java_adaptive/images/cognisync.png](http://www.java.net/external?url=http://www.cognisync.com)

Powered by Oracle, Project Kenai and Cognisync