

PC Connectivity over Bluetooth in Java™ Applications

Version 1.0; December 8, 2006

Java

NOKIA

Contents

1	Introduction.....	6
2	Use cases.....	7
2.1	Static connections.....	7
2.2	Dynamic connections.....	7
3	COM-based solution.....	9
3.1	Architecture	9
3.1.1	Needed equipment and software.....	10
3.1.2	Pros and cons.....	10
3.2	Determining which Bluetooth stack is in use.....	11
3.3	Installing the javax.com package.....	12
3.4	Configuring the virtual COM port with the Windows XP Bluetooth stack.....	12
3.4.1	Configuring the system to server mode	12
3.4.2	Configuring the system to client mode	14
3.5	Configuring the virtual COM port with the Widcomm Bluetooth stack	19
3.6	Finding the device's services.....	20
3.7	Examples.....	20
3.7.1	PC-to-phone example	20
3.7.2	Phone-to-PC example.....	21
4	JSR-82-based solution.....	23
4.1	Architecture	23
4.1.1	Needed equipment and software.....	23
4.1.2	Pros	24
4.2	Example case: BlueCove	24
4.2.1	Installing BlueCove	24
4.2.2	Examples and issues.....	24
5	Deciding factors	28
5.1	Supported Bluetooth stack	28
5.1.1	Windows XP Service Pack 2 Bluetooth stack.....	28
5.1.2	Widcomm Bluetooth stack.....	29
5.2	Expenses.....	30
5.3	API documentation availability.....	30
5.4	Features needed.....	30
5.5	Utility modules.....	31
6	Interoperability.....	32
6.1	Hardware level.....	32
6.2	Protocol level.....	32

6.3 Bluetooth stack implementation/API.....	34
6.4 Libraries and other software implemented over the Bluetooth stack	34
7 References	35
8 Evaluate this resource	36

Change history

December 8, 2006	v1.0	Initial document release

Copyright © 2006 Nokia Corporation. All rights reserved.

Nokia and Forum Nokia are registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

1 Introduction

This document discusses the process of connecting a mobile phone and a PC using Bluetooth. The reader is assumed to be familiar with Bluetooth terminology and basic operations such as finding and pairing devices.

The document focuses specifically on aspects of programming with the Java™ language; although other programming languages can be used for Bluetooth programming, they are not discussed here. Additional information about Bluetooth programming with Java™ Platform, Micro Edition (Java™ ME) MIDlets is available in the document [MIDP: Bluetooth API Developer's Guide](#) [1].

Two use cases are identified and explained in Chapter 2. Understanding these two cases is important for understanding further discussion about possible solutions. Chapters 3 and 4 discuss two common solutions — COM-based and JSR-82-based solutions — in more detail. These chapters will help the reader understand the pros and cons of both solutions. Chapter 5 lists the most important factors that must be considered when evaluating and choosing between different solutions. Finally, Chapter 6 discusses interoperability issues with Bluetooth.

The code examples that are referenced in the following chapters can be downloaded together with this document.

2 Use cases

With Bluetooth connections there are two basic use cases:

- Static connections
- Dynamic connections

The simpler static connections case refers to data transferring between two connected devices. The more advanced dynamic connections case involves searching for a device(s), selecting services, and then transferring data between devices.

Note : This separation into static and dynamic connections use cases is done only for purposes of better describing use cases in this document.

2.1 Static connections

A common use for Bluetooth communication is when the user wants to synchronize data or send files (or applications) between a personal computer and a mobile phone. Examples include synchronizing a calendar or installing a new application to a mobile phone.

When a user wants to frequently communicate between two known devices and authentication is needed, these devices are usually paired (this only needs to be done once). Pairing must be done via a mobile phone and a computer's user interface and usually involves typing the same passkey to both.

If authorization is required (which is usually the case), a user must accept each incoming Bluetooth connection. However, the connection can be set as trusted so that incoming connections from a certain device are automatically accepted. This arrangement can be further developed so that the devices automatically open a connection when they detect each other. Figure 1 illustrates this case.

Note that authentication and authorization can be set On or Off in the MIDlet of the receiving end (slave) device.

For the purposes of this document, the above-described case is called a static connection.

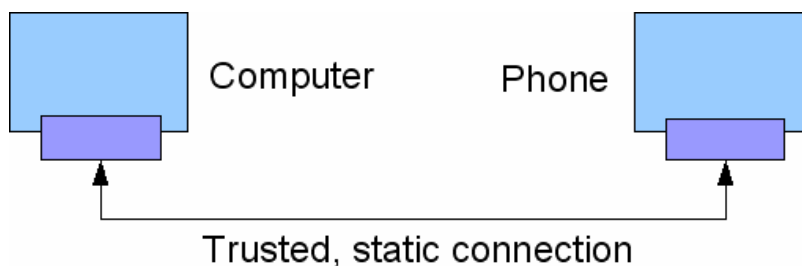


Figure 1 : Static connections

2.2 Dynamic connections

The second case for Bluetooth connections involves finding devices according to the services they publish. It may not be important which device connection is established, but it is important to have the correct service in use.

One interesting example of this kind of connection is a commercial service — for example, printing pictures from the mobile phone — provided by a company. The user walks into a photo kiosk and starts up the photo-printing software in his or her mobile phone. The software searches for a nearby Bluetooth device that publishes custom image, transfer service. Once a device publishing that service

is found, a connection is established and the mobile phone asks the user to select pictures for printing. After the pictures are transferred, the connection is closed (see Figure 2).

In this case, the computer and mobile phone do not know about each other. The first task is to enable the computer and mobile phone to find each other. The second task is to pair the computer and mobile device. After pairing is done, the connection can be opened and data transferred.

Finding other Bluetooth devices is usually initiated by hand using the mobile phone or computer's user interface. The same applies to pairing devices. However, these tasks can be automated, thus reducing the actions needed from the user and greatly simplifying such tasks.

Ultimately, the entire process, from finding devices to synchronizing data, can be automated, and the user is simply informed that data is now synchronized. From the user's point of view this is a significant advantage compared to the simpler, static connection.

For the purposes of this document, the above-described case is called dynamic connection.

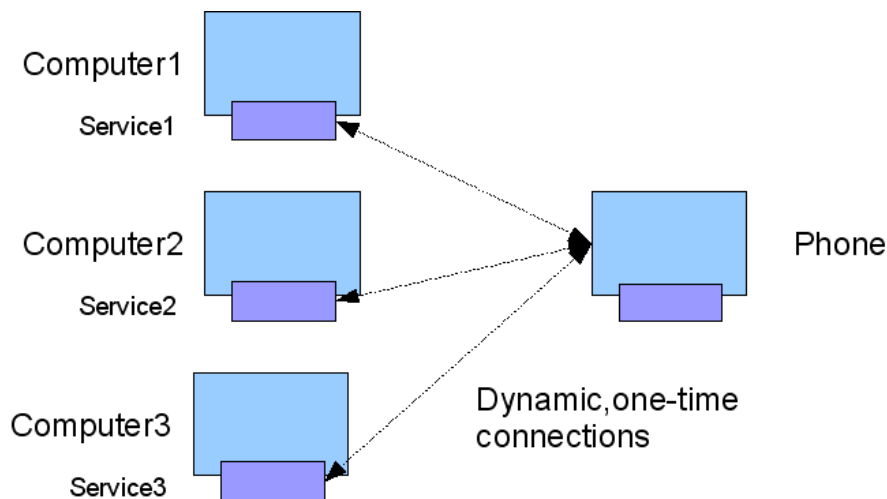


Figure 2 : Dynamic connections

3 COM-based solution

The COM-based solution is based on virtual COM ports created by both communicating devices. From the programmer's point of view, using a virtual COM port is similar to using a normal COM port.

This solution is best for the static connections use case.

3.1 Architecture

From the programmer's point of view, the virtual COM solution's architecture is simple and transparent at the PC end. At the mobile-device end there is not much difference between a COM-based solution and a JSR-82-based solution.

The programmer only needs to be concerned with the existence of a virtual COM port and finding the correct port number. Once a port is available, it's a simple matter of opening the COM port and starting to transfer the data. There is no need to be concerned about differences in implementations.

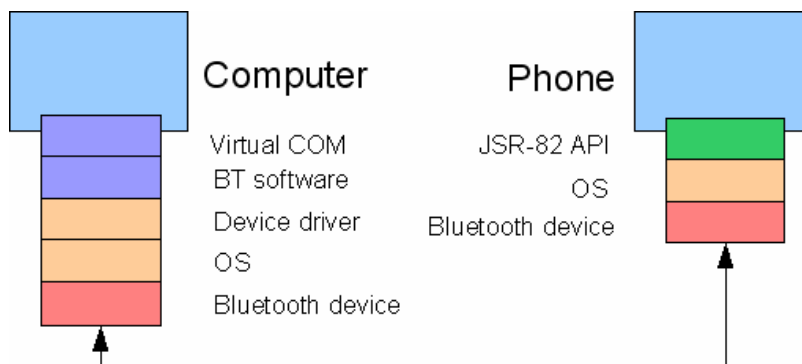


Figure 3 : Virtual COM port architecture

However, there are many components involved in this solution. Although these components are not directly visible, it is good to know them. The overall architecture is shown in Figure 3.

Virtual COM port: The virtual COM port is a software-based COM port. It is created and handled by Bluetooth software and there is no associated hardware present. Normal COM ports are attached to actual hardware, so if a programmer opens the COM1 or COM2 port on a PC, a real COM port is accessed.

Although virtual COM is drawn in Figure 3 as its own component, it is implemented by a Bluetooth software component. Drawing it as its own component makes illustrating the architecture cleaner.

JSR-82 API: The JSR-82 API is implemented over Bluetooth software and/or a device driver on the PC. See Chapter 4, "JSR-82-based solution," for more information about the JSR-82 solution at the device end.

Bluetooth software: Bluetooth software publishes API(s) for accessing Bluetooth devices. Bluetooth software allows configuring of Bluetooth — for example, giving a name to a device. The important part is the user interface for managing devices and connections. It is Bluetooth software's task to offer a user interface for finding other devices and creating pairings.

Device driver: A device driver communicates with the Bluetooth device at the lowest level. A device driver usually implements the Bluetooth protocol stack and other lower-level software components.

Operating system: The operating system runs other software components.

Bluetooth device: A Bluetooth device is physical hardware related to Bluetooth connections. A device is attached to other PC or mobile phone hardware.

3.1.1 Needed equipment and software

Implementing this solution requires a Bluetooth device compatible with a Bluetooth stack and Bluetooth software (for example, the Bluetooth stack of Windows Service Pack 2 or Widcomm Bluetooth stack) and software at the computer end.

A second required software package is the javax.com package, which allows Java™ 2 Platform, Standard Edition (J2SE™) to use COM ports.

3.1.2 Pros and cons

The pros of a COM-based solution include easier implementation and better interoperability between different devices.

Easy implementation: From the programmer's point of view, this solution is easy to implement. The connection is already established and active, and all needed configurations at both ends are already done. All the programmer has to do is open the specified COM port and start reading and writing data. The only tricky part might be finding the number of the COM port at the computer end. This usually requires adding a user interface through which the user can specify the port.

Interoperability: This solution works whenever virtual COM ports are available. Implementation does not need to take into account the differences between implementations or devices.

Safety: Since creating a connection requires user activity, the user knows about active connections and knows exactly what devices are communicating. There is a slight possibility for human error when accepting communication requests from unknown or untrusted devices.

Not a Bluetooth-specific solution: A virtual COM-based solution is not a Bluetooth-specific solution. Indeed, virtual COM ports can be used to communicate over other mediums that can create virtual COM ports, such as a USB cable.

The cons of a COM-based solution are in defining virtual ports and creating a connection. In a computer, the virtual COM port must be defined in Bluetooth software. Configuration methods and details are different for every software implementation. Furthermore, different Bluetooth software implementations may use a different range of COM port numbers. So, there is no universal way to configure virtual COM ports for use or to even know which port numbers are available.

Tricky configuration: As mentioned above, the virtual COM port must be defined using Bluetooth software tools, and each type of software has its own configuration user interface and parameters. One good example is in how each Bluetooth software type publishes virtual COM ports. For example, Widcomm software publishes the virtual COM port once it is defined in the configuration options. But Windows XP Service Pack 2 Bluetooth software requires that there is software that opens the configured virtual COM port before it is published. This may be a surprise for users.

User activity required: This solution requires the user to actively participate in finding devices and pairing them. Manually performing device inquiries and pairings is a tedious and frustrating task. Nobody wants to go through these procedures every time before transferring data between devices. One solution is to define commonly used devices as trusted devices. Then connections can be created automatically whenever devices detect each other.

Point-to-point connection: Virtual COM ports are like normal COM ports. They are just an address for connecting. This solution does not allow developers to create and publish own, application-specific services.

3.2 Determining which Bluetooth stack is in use

To use the Windows XP Bluetooth stack, the stack must be installed and must work with the Bluetooth device to be used. First make sure the Bluetooth device is connected to the computer. Many configuration options and applications are not visible if a device is not connected.

The Windows XP Service Pack 2 Bluetooth stack can be recognized with the following steps:

1. Open the Control Panel from the Start menu.
2. Open Bluetooth Devices.
3. Select the Hardware tab.

The dialog shown in Figure 4 will appear:

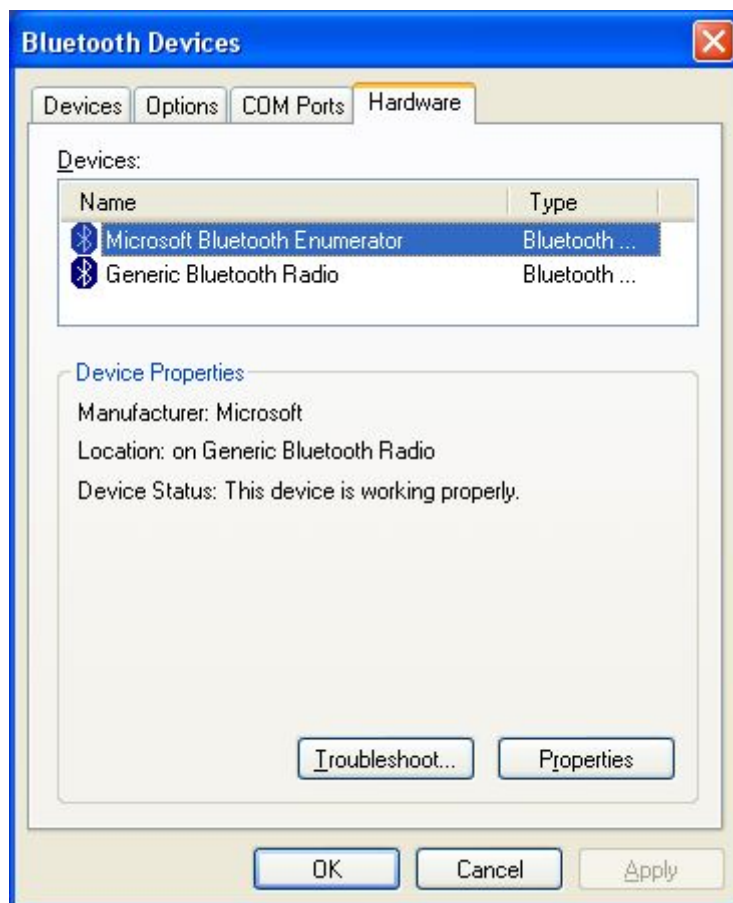


Figure 4 : Identifying Windows XP Bluetooth stack

If both Microsoft Bluetooth Enumerator and the Generic Bluetooth Radio items are visible, and the Device Status line says, "This device is working properly," then the Windows Bluetooth stack is in use.

When the Widcomm Bluetooth stack and software are installed, a new Control Panel item named Bluetooth Configuration is added. There are also many related applications and icons added, such as My Bluetooth Places to the Start menu.

3.3 Installing the javax.com package

To use the virtual COM solution, the javax.com package must be installed first. The javax.com package can be downloaded from <http://java.sun.com/downloads/index.html>. The downloaded package has a file with installation instructions.

3.4 Configuring the virtual COM port with the Windows XP Bluetooth stack

After the javax.com package is installed, the virtual COM port must be configured before it can be used. There are two options for the Windows XP Bluetooth stack. The system may be configured to the server or client mode depending on the application domain. The configuration to server or client mode will be examined in Sections 3.4.1 and 3.4.2

The Bluetooth port in the server mode accepts incoming connections from other devices. The Bluetooth client mode port starts the connection to the server mode Bluetooth port.

3.4.1 Configuring the system to server mode

The configuration of the virtual COM port to server mode can be done as follows:

1. Open Control Panel from the Start menu.
2. Open Bluetooth Devices.
3. Select the COM Ports tab; the dialog shown in Figure 5 appears.

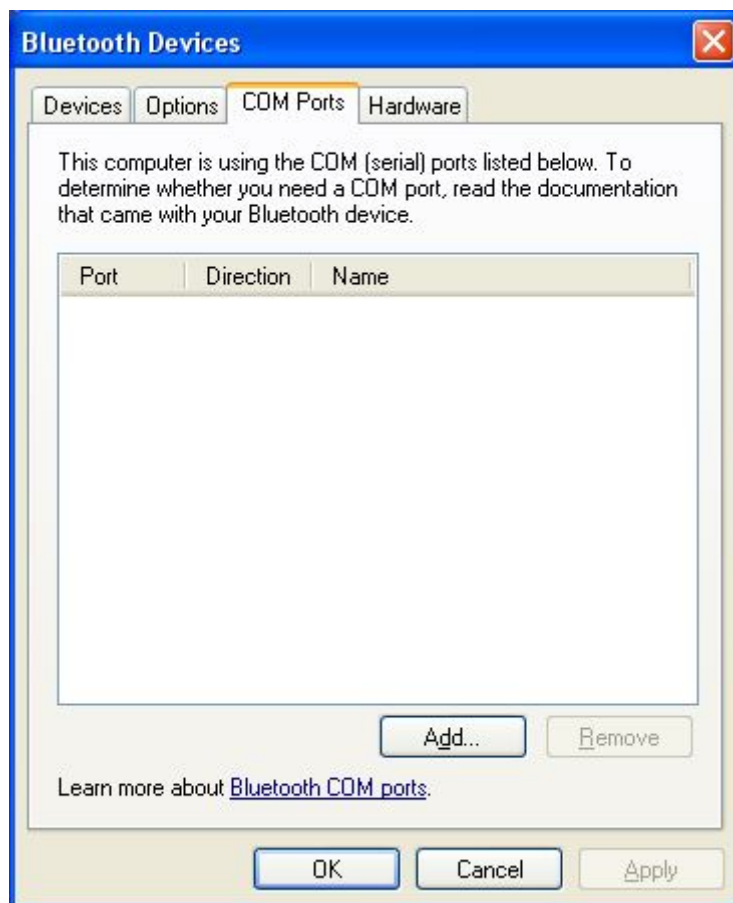


Figure 5 : Windows XP Bluetooth stack virtual COM ports

4. Click the Add button, and the dialog shown in Figure 6 appears.



Figure 6 : Adding virtual COM port to Windows XP Bluetooth stack

This dialog allows the user to select who opens the connection. Usually it is wise to allow the device to open the connection (“Incoming” type selected). For computer-initiated connection, there must be a published service available in the device with which to connect. This is not the case when using virtual COM ports, since new services are not added to the mobile phone.

5. After clicking the OK button, a new port is created and the COM port is assigned to it. The dialog should look like the one shown in Figure 7.

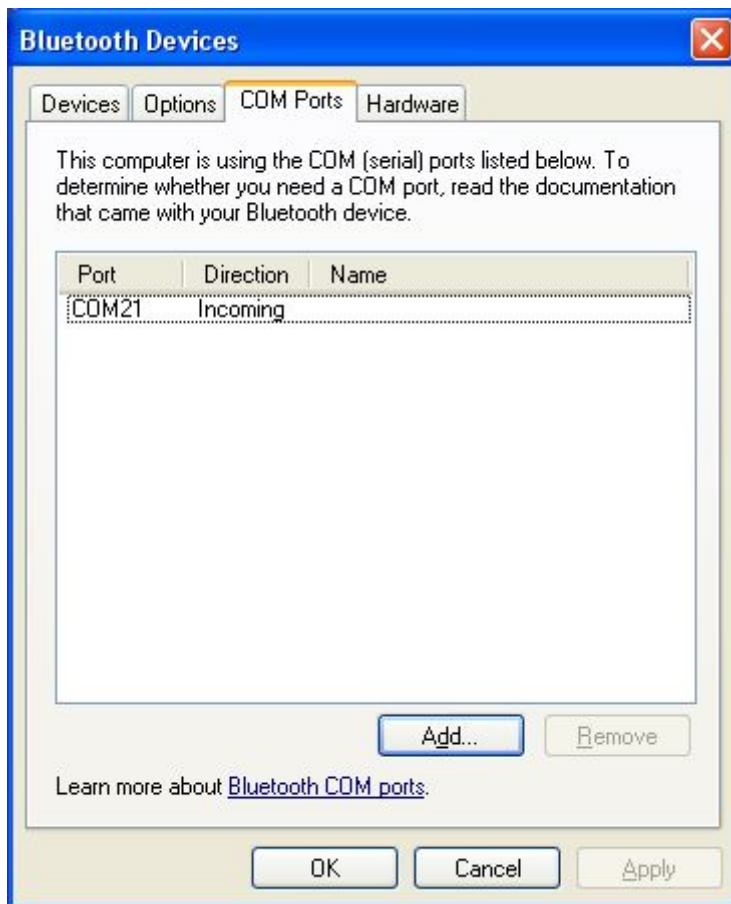


Figure 7 : COM port added to Windows XP Bluetooth stack

Note that there is no name assigned to the COM port connection.

Note : Windows does not publish this COM port until there is some user application that opens it. This means that this port cannot be found from the mobile phone until a user application runs in Windows.

Unlike the Windows XP Bluetooth stack, the Widcomm Bluetooth stack publishes the virtual COM port once it is added, so other devices can find that port immediately.

3.4.2 Configuring the system to client mode

The configuration of the virtual COM port to client mode can be done as follows:

1. With the device: Start the Serial Port Profile application on the device. An example application of a server mode serial port profile MIDlet is given in Section 3.7, "Examples."

Note : Windows is unable to find the service if the service record of the application is not correct.

Note : The Windows XP Bluetooth stack performs the service search once when pairing with the device. The user cannot use the COM Ports tab (as in Figure 5) afterwards if the server application on the device was not running during the pairing phase.

2. Continue with the PC: Open Control Panel from the Start menu.
3. Open Bluetooth Devices.

4. Select the Devices tab and click the Add button to start pairing with the device (Figure 8). However, if the device is already listed, remove it by selecting the device and then clicking the Remove button.

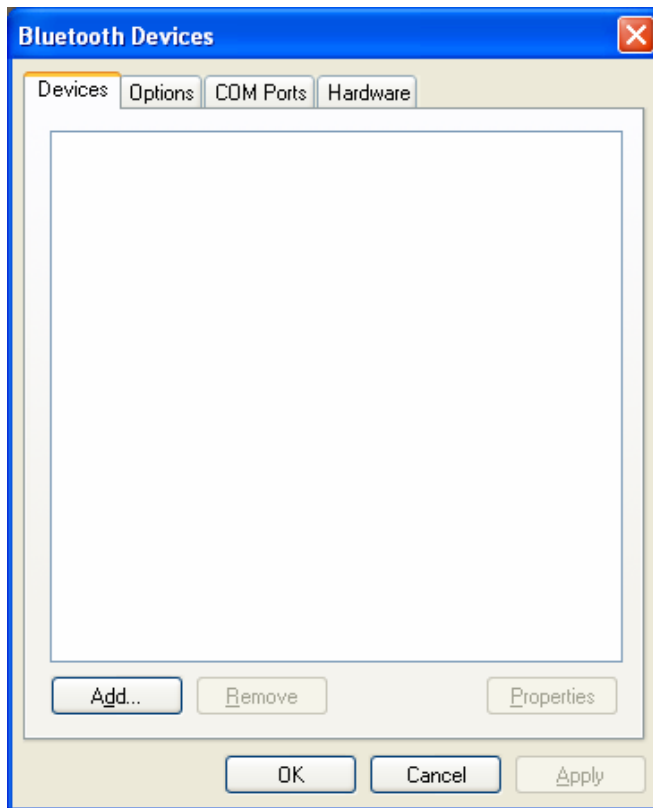


Figure 8 : Pairing the system with a device that has a serial port service running

5. From the opening dialog (Figure 9) select the checkbox of "My device is set up and ready to be found." Then click the Next button.



Figure 9 : Adding a new Bluetooth device with wizard

6. The Windows XP Bluetooth stack starts an inquiry to find new devices. Select the device from the list, as shown in Figure 10, and click the Next button.

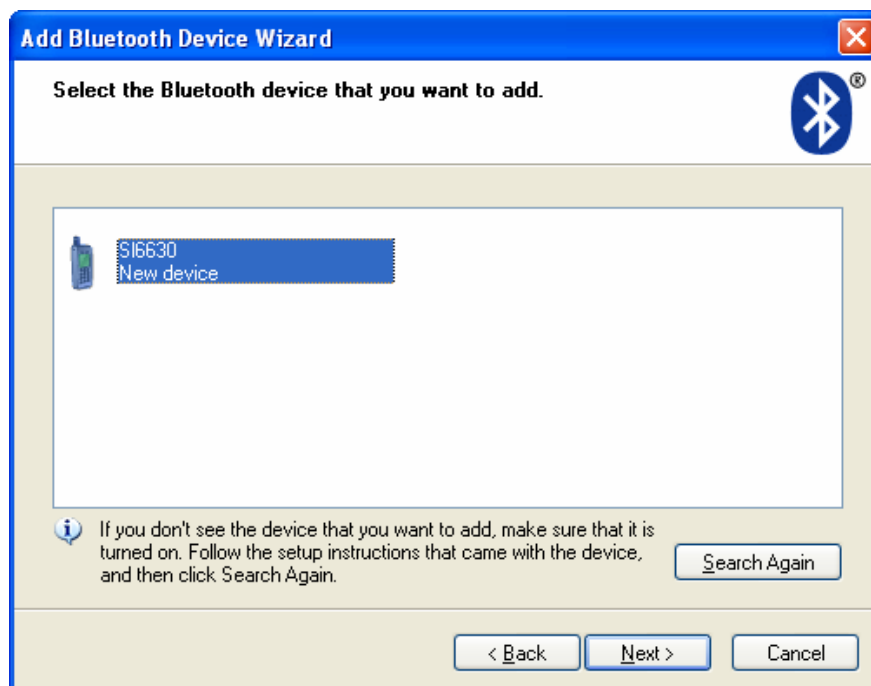


Figure 10 : Searching devices with the wizard

7. Pair the system with the device by clicking the Next button (Figure 11).

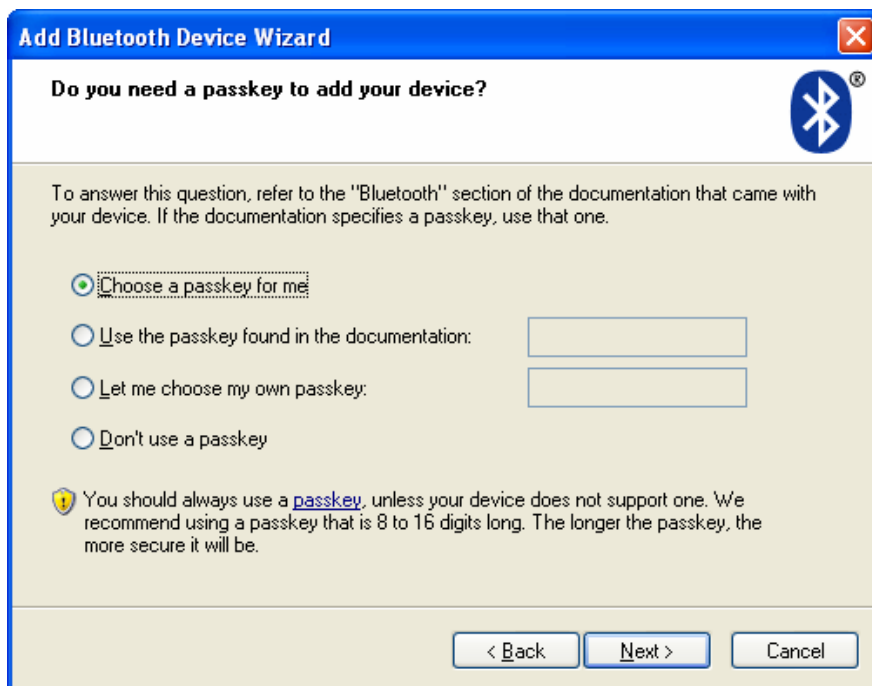


Figure 11 : Pairing the device with the system using a passkey

8. With the device: Accept a new connection by giving the passkey found in the system's dialog (Figure 12).

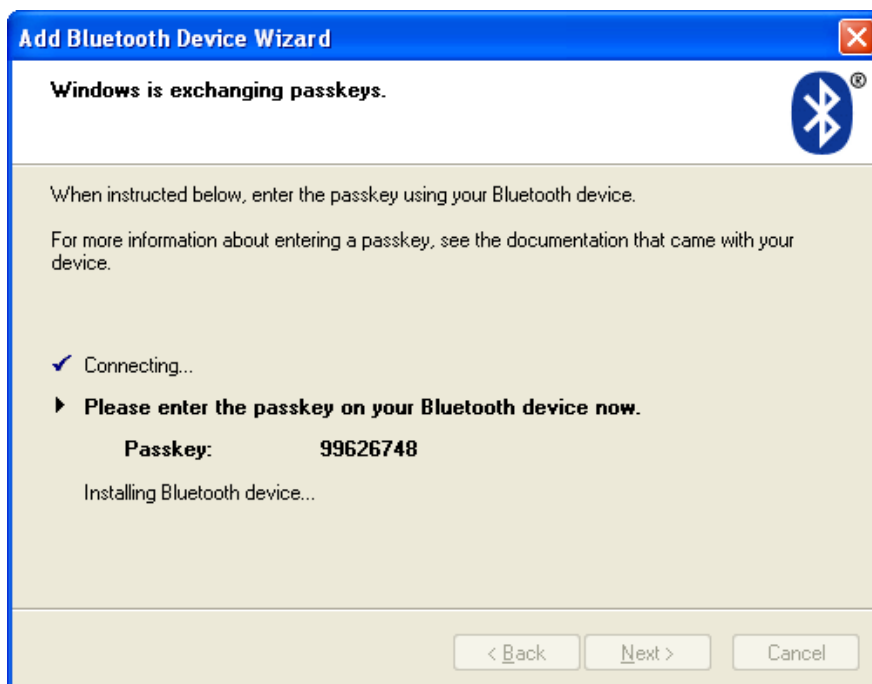


Figure 12 : Dialog showing the passkey to pair a device and the system

9. Continue with the PC: Click the Finish button to complete the wizard (Figure 13).



Figure 13 : Completing the wizard

10. The installed virtual COM ports can be seen in the COM Ports tab (Figure 14).

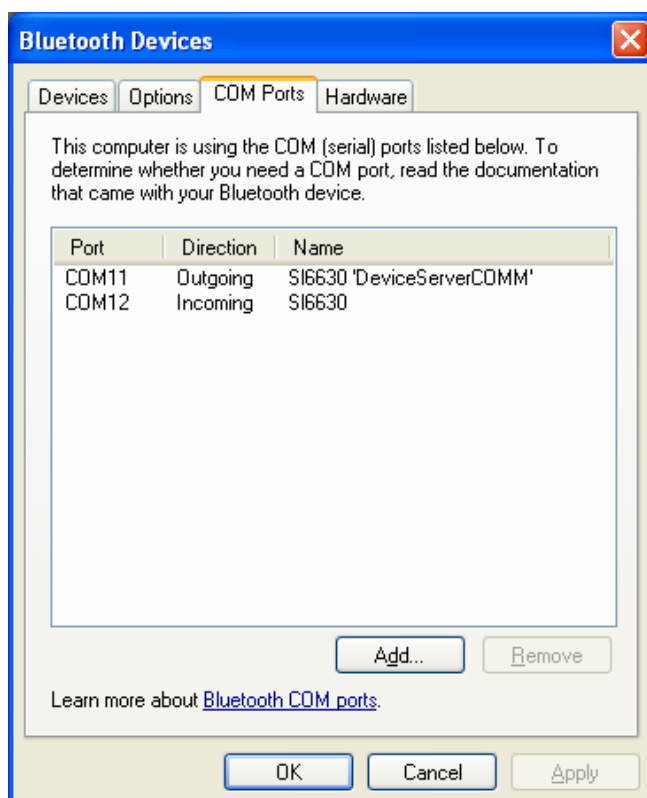


Figure 14 : COM Ports tab

3.5 Configuring the virtual COM port with the Widcomm Bluetooth stack

Widcomm software is easier and more flexible to configure. Follow these steps:

1. Open Bluetooth Configuration from the Control Panel.
2. Select the Local Services tab. The dialog shown in Figure 15 appears.

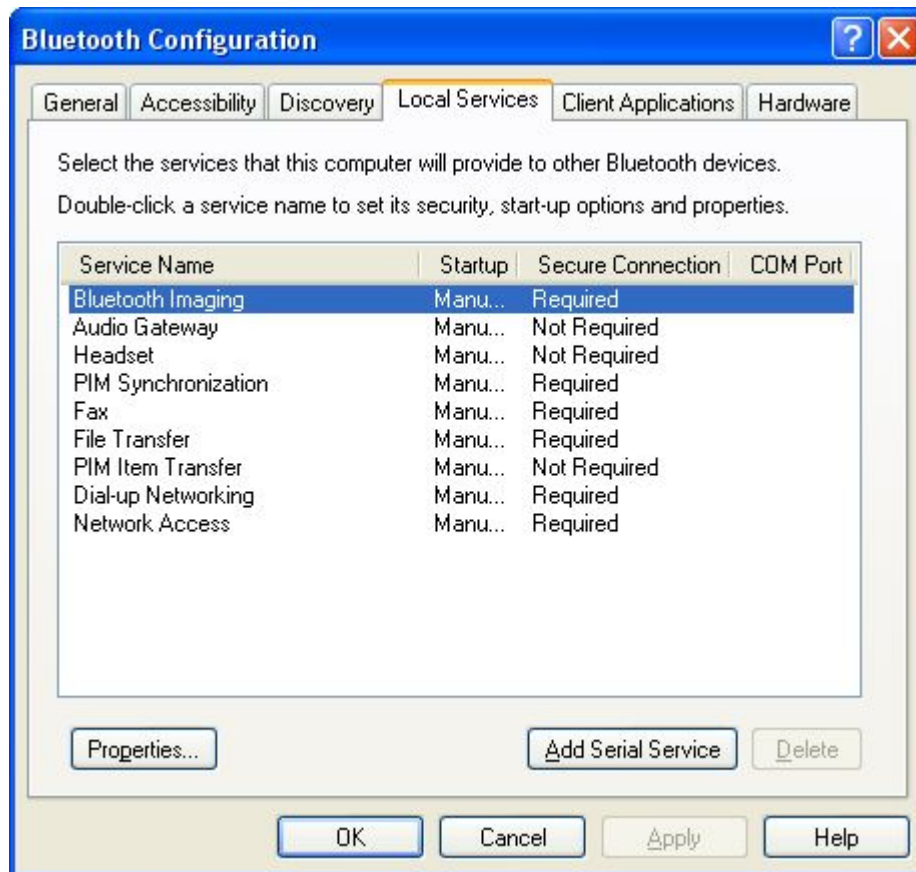


Figure 15 : Widcomm Local Services

3. Click the Add Serial Service button, and the dialog shown in Figure 16 appears.

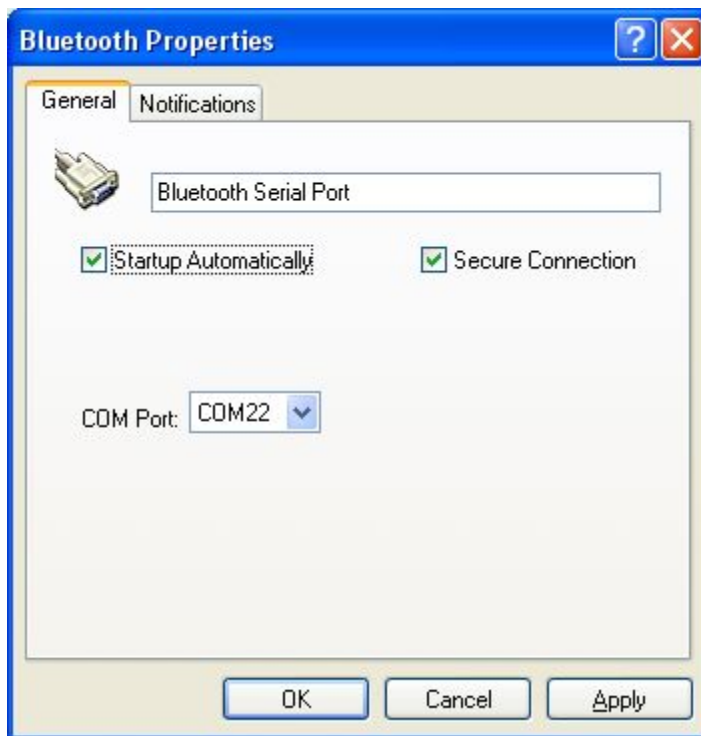


Figure 16 : Adding the virtual COM port to Widcomm software

4. The dialog allows the user to select the name and port number for the virtual COM port. The Startup Automatically checkbox should be enabled unless the user wants to start the virtual COM port manually every time before use.

3.6 Finding the device's services

Before it is possible to connect to a service in the device from the PC, that service must be found from the device. The published service requires that there is an application publishing the service. Usually services are found when the device is found. This is also the easiest way to find services.

Note : When working with the Windows XP Bluetooth stack, the service must be available in the device before Windows finds the device. The Windows Bluetooth stack cannot find the device's services after the device is added to the device list. This means the application providing the service must be running in the device before the device is introduced to the Windows Bluetooth stack.

3.7 Examples

There are two example pairs. The first one contains an example for a mobile phone acting as a server and a PC client connecting to it (see Section 3.7.1, "PC-to-phone example"). The second pair consists of an example for a PC connecting as a server and a mobile phone client connecting to it (see Section 3.7.2, "Phone-to-PC example"). Mobile phone applications are MIDlet applications and PC applications are Java applications.

3.7.1 PC-to-phone example

Before running this example, make sure the mobile phone and PC are not yet connected over Bluetooth. If there is already a connection, remove the existing connection first. The reason for this is that the Windows XP Service Pack 2 Bluetooth stack performs service finding only when pairing a device. So the service must be available in the mobile phone when it is paired.

1. Install the DeviceServerCOMM MIDlet to the mobile phone and run it.

2. Pair the mobile phone with the PC. Now the PC finds the service the MIDlet has created.
3. Start the serial port application in the PC. SerialDemo is provided with the javax.comm package, for instance.

An example code of the device server is as follows:

```
// Set the service record to Serial Port Profile which can be seen with
// Win XP.
private final static String UUID = new UUID("1101", true).toString();
public class DeviceServerCOMM extends MIDlet {
protected void startApp() throws MIDletStateChangeException {
    try {
        LocalDevice device = LocalDevice.getLocalDevice();
        device.setDiscoverable(DiscoveryAgent.GIAC);
        String url = "btspp://localhost:"+UUID+";name=DeviceServerCOMM";
        StreamConnectionNotifier notifier =
            StreamConnectionNotifier) Connector.open(url);
        StreamConnection conn = notifier.acceptAndOpen();
        // Now, we can do anything with streams.
        InputStream in = conn.openInputStream();
        OutputStream out = conn.openOutputStream();
    } catch (Throwable e) {}
}
protected void pauseApp() {}
protected void destroyApp(boolean b) throws MIDletStateChangeException {}
```

3.7.2 Phone-to-PC example

Unlike the PC-to-phone connection example, this example does not require pairing to be done after starting the server component.

1. Start the PCServerCOMM application in the PC.
2. Edit, compile, and install the DeviceClientCOMM MIDlet to the mobile phone and run it (the remote host is hard-coded into the source for simplicity).

With the Windows XP Bluetooth stack it is recommended to use a short UUID, as in the following example code:

```
private void connectToHost(RemoteDevice host) {
    DiscoveryAgent discv =
        LocalDevice.getLocalDevice().getDiscoveryAgent();
    // 0x1101 is base UUID value for serial port profile as specified in
    // the JSR-82 documentation. As a side note, the device client shall
    // use the proper UUID constructor, i.e., new UUID("1101", true)
    // instead of new UUID(0x1101).
    UUID uuids[] = new UUID[] { new UUID("1101", true) };
    discv.searchServices(null, uuids, host, this);
}
```

Note : Use the short UUID constructor `public UUID(java.lang.String uuidValue, boolean shortUUID)` to connect to virtual COM in a Windows XP Bluetooth stack-based system, i.e., `new UUID("1101", true)`.

4 JSR-82-based solution

This solution is based on JSR-82, which is the Java API defined for using Bluetooth in Java™ ME devices. J2SE™ used in personal computers does not support the JSR-82 API by default. There are third-party implementations for the JSR-82 API to be used with J2SE. These third-party JSR-82 implementations are based on existing Bluetooth stacks in computers and usually only implement the API over Bluetooth stacks services.

One example of JSR-82 implementation for J2SE is BlueCove (<http://bluecove.sourceforge.net/>). BlueCove works with the Windows XP Service Pack 2 Bluetooth stack (and all Bluetooth devices it supports).

4.1 Architecture

JSR-82 solution architecture is mostly identical to the virtual COM port solution. The important difference is that the JSR-82 API implementation is its own component, separate from the Bluetooth software or operating system.

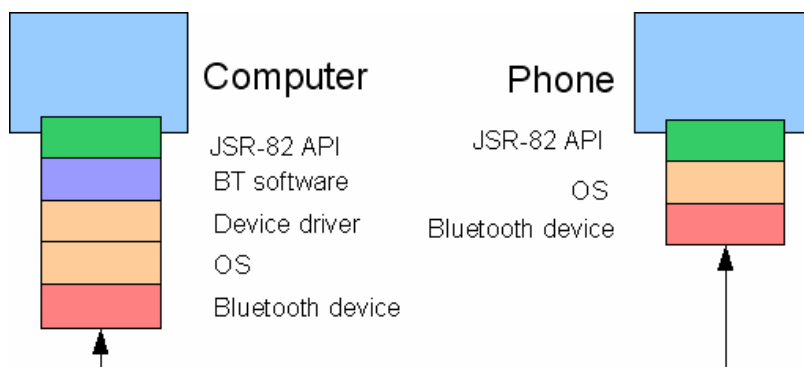


Figure 17 : JSR-82 solution architecture

With the JSR-82-based solution, the programmer must be aware of the possible limitations that every component causes. Usually there are restrictions caused by a device driver or Bluetooth software implementations. Limitations at the lowest level affect all levels above it. So, device driver limitations are visible in Bluetooth software and JSR-82 levels. Figure 17 illustrates the overall architecture.

JSR-82 API: JSR-82 API implemented over Bluetooth software and/or device driver.

Other components: See virtual COM architecture in Section 3.1.

4.1.1 Needed equipment and software

Implementing the JSR-82-based solution requires a Bluetooth device compatible with the Bluetooth stack and software at the computer and the device ends. Windows XP Service Pack 2 Bluetooth stack and Widcomm Bluetooth stack are two examples.

A second needed software component is JSR-82 implementation. Most of the available JSR-82 implementations are specific to a certain Bluetooth stack and software. For example, there are two free JSR-82 API implementations for the Windows XP Service Pack 2 Bluetooth stack:

- BlueCove (<http://bluecove.sourceforge.net/>)
- BlueSock (<https://bluesock.dev.java.net/>)

4.1.2 Pros

The pros for the JSR-82 solution are in the automation of device finding and connecting.

Automation: The JSR-82 API allows automatic device finding and connecting. JSR-82-based software can, for example, run device discovery constantly, automatically create a connection with authorized devices, and do the needed data transfer. At best this does not require any user interaction; it can be an automatic background process.

Rich and capable API: A virtual COM-based solution transfers data between already connected devices. A JSR-82-based solution can handle a list of devices and perform device finding and connection handling — basically, all Bluetooth communication-related tasks.

Portability: By using the JSR-82 API a large amount of code can be shared between J2SE and Java ME implementations.

Differences in back-end support: With J2SE JSR-82 implementations, it is important to remember that all Bluetooth software does not support all the features that the JSR-82 API itself supports. BlueCove that uses Windows XP Service Pack 2 Bluetooth software supports only RFCOMM communications. There is no L2CAP support, which may be a restriction for some applications. Similarly there might be other restrictions with other JSR-82 API implementations.

4.2 Example case: BlueCove

The example described in this document uses BlueCove JSR-82 implementation. A description of the installation phase is followed by examples and some special issues regarding BlueCove implementation.

4.2.1 Installing BlueCove

Installing BlueCove is a straightforward task:

1. Download BlueCove binary distribution from <http://bluecove.sourceforge.net/>.
2. Unpack the downloaded Zip file to any temporary folder.
3. Copy the intelbth.dll file to a location where your software can find it. Possible locations are, for example, the Windows/System32 folder, or any folder that is in the %PATH% environment variable.
4. Copy the bluecove.jar file to a folder. As with other JAR files you may need, add this JAR file to your application's classpath.

4.2.2 Examples and issues

Two examples will be presented: a simple echo application using the same serial port profile as in the virtual COM example (Section 3.7); and a larger chat application with a user interface. Both simple and larger applications can be found in the source package as PC applications and MIDlets for mobile devices. Since PC applications use the JSR-82 API with J2SE, they require BlueCove (or a similar JSR-82 API implementation) to be installed to the PC first.

To run the simple echo application:

- Running PC-to-phone example:
 1. Install the DeviceServerCOMM MIDlet to the mobile phone and run it.
 2. Start the PCClientCOMM application in the PC.

- Running phone-to-PC example:

1. Start the PCServerCOMM application in the PC.
2. Install the DeviceClientCOMM MIDlet to the mobile phone and run it.

The larger application is a noncentralized chat application that operates over point-to-point Bluetooth connections. The chat application is inspired by Internet Relay Chat (IRC) applications. Starting the chat application is simple — the chat starts once the application is started. The user interface has an inbound area for all read messages from chatters and an outbound text field to send message to other chatters.

The PC version and device version have minor differences. In the PC version, bootstrapping is done with the Java `main()` method whereas the device version extends the MIDlet class (see `example.chat.ChatApplication` for details). The main differences are in the user interface. The PC version uses Swing UI (see `example.chat.SwingUI` for details) whereas the device version uses LCDUI (see `example.chat.LcdUI` for details). A custom item was developed to implement a good inbound area for messages on LCDUI. The documentation of the communication protocol can be found in the source code.

The larger chat application is run as follows. Install ChatMIDlet to the mobile phone and run it. With the PC, start the application with Ant build script or import the project into your development environment and run it.

There are some special issues regarding implementation; these are covered in the immediate following subsections. All issues listed here are bound to BlueCove versions 1.1.1 and 1.1.2. In addition to the issues listed here, the UUID feature may be problematic, as described in Section 3.7.2.

4.2.2.1 BlueCove properties

BlueCove does not implement any properties specified in the JSR-82 specification. The developer needs to use some arbitrary values instead, as in the following code:

```
// BlueCove does not implement properties, the method returns null.
//String property = LocalDevice.getProperty("bluetooth.sd.trans.max");
sdTransMax = 1; //Integer.parseInt(property);
```

Note : BlueCove does not implement JSR-82 properties.

4.2.2.2 RemoteDevice class's method getFriendlyName issue

BlueCove may crash if Windows has not yet resolved the friendly name of the RemoteDevice. The solution is to not use the method at all on BlueCove applications.

Note : BlueCove may crash if the getFriendlyName method is invoked from the RemoteDevice class.

4.2.2.3 RemoteDevice constructor bug in BlueCove version 1.1.1

There is a bug in the RemoteDevice implementation of BlueCove. The following code clip expresses the problem in detail. Additionally, the UUID feature is described. The feature has been fixed in version 1.1.2.

```
private static final String HOST_ADDRESS = "0013FDC157C8"; // N6630
class InternalRemoteDevice extends RemoteDevice {
    InternalRemoteDevice() {
```

```

        // BlueCove has a bug in parsing hex address, i.e.
        // it parses decimal address instead.
        //
        // This bug will not be visible if you use a normal inquiry method
        // to resolve the remote host.
        super("" + Long.parseLong(HOST_ADDRESS, 16));
    }
}

```

Note : BlueCove has a bug in the RemoteDevice protected constructor in version 1.1.1.
--

4.2.2.4 Low resources in service search

Some devices have features in the JSR-82 implementation — for example, all resources are not deallocated properly. The resource deallocation feature can be seen in the DiscoveryListener's `serviceSearchCompleted` method, i.e., `public void serviceSearchCompleted(int transID, int responseCode)`. If the `responseCode` is `SERVICE_SEARCH_ERROR` it is a sign of the resource deallocation feature. The solution is to try again later. The issue is shown only if there are a number of Bluetooth devices around. Some example code follows (this is a clip from a larger application). For example, the Nokia 6630 mobile device has an approximately two-second time-out when resources are deallocated. After that the service search is successfully accomplished.

```

private final static int SLEEP_TIME_BETWEEN_SERVICE_SEARCH_TRYES = 250;
private final static int MAX_TRY_COUNT_OF_SERVICE_SEARCH = 25;
public void serviceSearchCompleted(int transId, int responseCode) {
    /*
     * If the search response code is SERVICE_SEARCH_ERROR it usually
     * indicates that we have run out of device resources. The
     * implementation may, however, have a time-out when resources are
     * de-allocated. For example, with Nokia 6630 the time-out is around
     * 2 seconds. After there are free resources available the service
     * search is successfully completed.
     */
    if (responseCode == DiscoveryListener.SERVICE_SEARCH_ERROR
        && currentCountOfServiceSearchTries <
        MAX_TRY_COUNT_OF_SERVICE_SEARCH) {
        currentCountOfServiceSearchTries++;
        try {
            searchThread.wait(SLEEP_TIME_BETWEEN_SERVICE_SEARCH_TRYES);
        } catch (InterruptedException e) {
            LogScreen.log(e.toString());
        }
    }
    else {
        unsearchedRemoteDevices.removeElementAt(0);
    }
}

```

Note : There seems to be a resources-related problem with mobile phones regarding device and service searching. If there are a number of Bluetooth devices nearby, then the mobile phone's available connection resources are spent in searching for a device and there are no resources left for service searching. This problem is solvable by redoing a) the service search after waiting for a little while or b) the device search, when devices are found in another order, which affects the service search order.

5 Deciding factors

Choosing between the solutions presented in this document is not easy, and indeed there are many aspects that have not yet been discussed. This chapter discusses several different approaches and points of view for deciding between various solutions and products.

5.1 Supported Bluetooth stack

With Windows-based implementations this is probably the most important decision. It is very unfortunate, but the importance of Bluetooth stack selection cannot be underestimated.

Selecting Bluetooth stack implementation sets limits for the APIs and features that can be used. Bluetooth stack selection sets limits on the hardware. Only hardware supported by the Bluetooth stack can be used.

As already discussed in previous chapters there are free solutions that come with operating systems; there are also commercial software products that must be purchased and licensed separately.

5.1.1 Windows XP Service Pack 2 Bluetooth stack

Microsoft added Bluetooth support for Windows in Windows XP Service Pack 2. Earlier versions of Windows lack support for Bluetooth. There is a patch available for using Bluetooth with Windows XP Service Pack 1 but it is very limited, and one can assume it is not preinstalled in the computer.

Windows XP Service Pack 2 comes with Microsoft's Bluetooth stack implementation. The Windows Bluetooth stack works with a large set of hardware. While a limited set of hardware is recognized specifically, numerous hardware devices work as general Bluetooth devices. Working as a general Bluetooth device means a device may not work optimally, but it will work.

The Windows XP Bluetooth stack supports only RFCOMM connections. There is no support for L2CAP.

Another interesting point in programming with the Windows Bluetooth stack is that there are basically two APIs for Bluetooth:

- `Windows socket-based API`
- `Bluetooth API`

Bluetooth can be used almost like any other socket connection with a socket-based API. When the connection is opened, the Bluetooth protocol is specified with a couple of parameters. This is an easy way to transfer data. The downside is almost the same as with virtual COM: there is no device finding or management support. However, a socket-based API supports defining a connection string so one can select the device and service with which to connect.

The Bluetooth API, much like the JSR-82 API, allows lower-level access to Bluetooth connections. There is support for device and service finding, creating services, etc. However, the Bluetooth API is a fairly low-level API, so it is tricky and difficult to use.

Fortunately, JSR-82 implementations that work over those APIs hide most of the details, but it is still important to recognize the limitations they set.

The pros for a Windows XP Bluetooth stack are mostly related to the fact that it is distributed and installed with the operating system.

Installed with operating system: Because the Windows XP Service Pack Bluetooth stack is installed when Windows is installed, there is no need to install additional software. It is safe to assume that most target Windows XP computers nowadays have Service Pack 2 (and the Bluetooth stack) installed.

Supports lots of hardware: As mentioned earlier, the Windows Bluetooth stack does not support a wide set of hardware natively, but it supports lots of hardware through its general Bluetooth device driver.

Public API: The Windows Bluetooth stack's API is publicly documented on the Microsoft Developer Network (MSDN) at <http://msdn.microsoft.com>. Compiling Bluetooth-enabled applications requires that the Platform SDK for Windows XP or later is installed. That SDK is freely available from Microsoft's Web site.

The cons for the Windows XP Bluetooth stack are related to the limited set of features it provides.

Supports only RFCOMM connections: While RFCOMM connections are sufficient for many solutions, missing L2CAP support may still be a burden.

Bluetooth software is limited: Bluetooth software that comes with the Windows XP Bluetooth stack is limited in functionality and configuration. For example, configuring virtual COM ports is not straightforward.

Supports only Windows XP Service Pack 2: The Windows Bluetooth stack comes only with Windows XP Service Pack 2. In order to support earlier Windows versions one must use another Bluetooth stack.

5.1.2 Widcomm Bluetooth stack

Widcomm Bluetooth stack is commercial third-party software from Broadcom. The Widcomm Bluetooth stack is practically a de facto Bluetooth stack that comes with numerous Bluetooth devices. It is also the only choice when using Windows versions other than Windows XP Service Pack 2.

At the time of creating of the examples, Broadcom did not publish its Bluetooth API unless you purchased their Bluetooth Development Kit. Currently the development kit is available with a royalty-free license.

The pros for Widcomm are that it supports several Windows versions and offers good, feature-rich Bluetooth software.

Supports other versions than Windows XP Service Pack 2: Widcomm supports Windows 98 Second Edition and all later Windows versions. This might be important for some applications.

Supports L2CAP: Unlike Windows' own Bluetooth stack, the Widcomm Bluetooth stack supports L2CAP.

Good Bluetooth software: Bluetooth software that comes with the Widcomm Bluetooth stack is versatile. The software allows easy configuration for connections (including virtual COM ports) and other features. For example, it is possible to allow only certain devices to connect.

The cons are about separate installation and possibly price.

Separate installation: Widcomm Bluetooth stack and software must be separately installed to the computer. Running installer software might not sound like a big task, but it is an additional one compared to the Windows XP Bluetooth stack.

Price: Widcomm development kit is available with a royalty-free licence for developers.

No general Bluetooth device support: Unlike the Windows XP Bluetooth stack there is no general Bluetooth device, so the Widcomm Bluetooth stack supports only a limited set of Bluetooth devices.

5.2 Expenses

The free Bluetooth stack comes with Windows XP Service Pack 2, while most other Bluetooth stack implementations are commercial software — and as mentioned earlier, there is a significant difference in price. The price of third-party Bluetooth stacks is usually very well justified by the presence of more versatile Bluetooth software and support.

If a solution needs to be freely available (or even Open Source licensed) then third-party Bluetooth stacks are in most cases practically unusable. On the other hand, there are limitations in using free and Open Source licensed Bluetooth solutions. Study licensing terms for free and Open Source software carefully before using them.

There are already lots of software and tools freely available for the Windows XP Bluetooth stack. Using other Bluetooth stacks might entail buying expensive tools from the manufacturer or other software developer. In addition, those tools usually support the one Bluetooth stack for which they were developed.

5.3 API documentation availability

For developers, good documentation is extremely important, and there are great differences in documentation quality and how it is published.

The Windows Bluetooth API is documented at MSDN, <http://msdn.microsoft.com>. The API documentation is complete, but it is not very easy to navigate or understand. As with other MSDN documentation there is a lot of information, but the challenge is to find the needed information.

However, easy availability of documentation and tools (Platform SDK) allows others to create more tools using the Windows Bluetooth stack. Many Open Source Software products are also available using the Windows XP Bluetooth stack. Two examples are JSR-82 implementations:

- BlueCove, <http://bluecove.sourceforge.net/>
- BlueSock, <https://bluesock.dev.java.net/>

Widcomm Bluetooth API documentation is currently available for developers as part of the Development Kit under a royalty-free licence.

5.4 Features needed

It is very important to know which kind of feature set a certain Bluetooth implementation supports or does not support. There is a potentially big problem here — the implementation might support all the features currently needed, but may not allow you to expand the application later on. This scenario can limit further improvements to a product.

Section 5.1 lists pros and cons for different Bluetooth stacks and software.

The Windows XP Service Pack 2 Bluetooth stack is the “basic” model:

- Supports only RFCOMM connections
- Provides limited configuration options

The Widcomm Bluetooth stack is much more feature rich:

- Supports L2CAP and RFCOMM

- Easy and flexible to configure
- Lots of features, making Bluetooth easier to use in Bluetooth software (for example, My Bluetooth Places shortcut added to Windows Desktop).

Interoperability is an important thing to consider, and is discussed in more detail in Chapter 6.

5.5 Utility modules

Usually some utilities are needed when working with Bluetooth solutions. There is significant difference between Bluetooth stacks with regard to what kind of tools they offer.

Available utilities can be divided into two categories:

- Utilities that come with Bluetooth stack/software. Usually implemented by Bluetooth stack/software implementer.
- Third-party utilities.

Utilities that come with Bluetooth stack/software are usually designed and implemented for use with that specific software/implementation. As such, they are efficient and can use all features supported.

Third-party utilities are based on APIs that Bluetooth stack/software vendors publish or license for others. These utilities can support several Bluetooth stacks/types of software. However they do not have access to internal knowledge, so they might not be so efficient or support all features.

Being public, the Windows XP Bluetooth API and SDK allow third parties to freely publish their tools with source code. Currently most free tools and articles about Bluetooth programming on the Internet are written for the Windows XP Bluetooth stack.

If an API is not published and/or it must be licensed, it is not possible to publish free tools or articles about it. Publishing an article about how to use a closed API would mean publishing the API as well.

6 Interoperability

Ideally every Bluetooth-capable device would be able to connect to every other Bluetooth-capable device. In practice there are several layers where there are incompatibilities. This chapter lists the most important levels in Bluetooth implementation and how interoperability works on each.

6.1 Hardware level

There are a several chipset manufacturers for Bluetooth devices, and unfortunately there are some incompatibilities between them. Thus, it is possible that Bluetooth devices from two different manufacturers will not work properly together.

These incompatibilities are best detected by simply testing the different devices.

6.2 Protocol level

There are several revisions of the Bluetooth specification and protocols. Windows' Bluetooth software usually shows which versions have attached device supports. To find out supported versions with the Windows XP Bluetooth stack:

1. Open the Control Panel from the Start menu.
2. Open Bluetooth Devices.
3. Select the Hardware tab; the flowing dialog shown in Figure 18 appears.

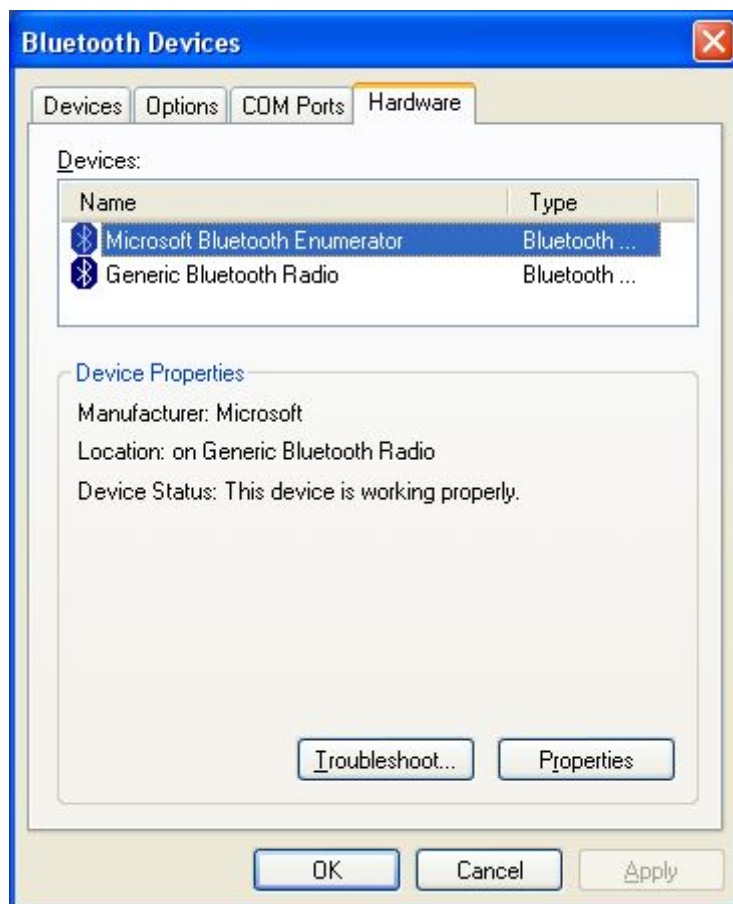


Figure 18 : Bluetooth hardware dialog

4. Select Generic Bluetooth Radio (lower line) and then select Properties.
5. From the dialog that opens, select the Advanced tab. The dialog shown in Figure 19 appears.

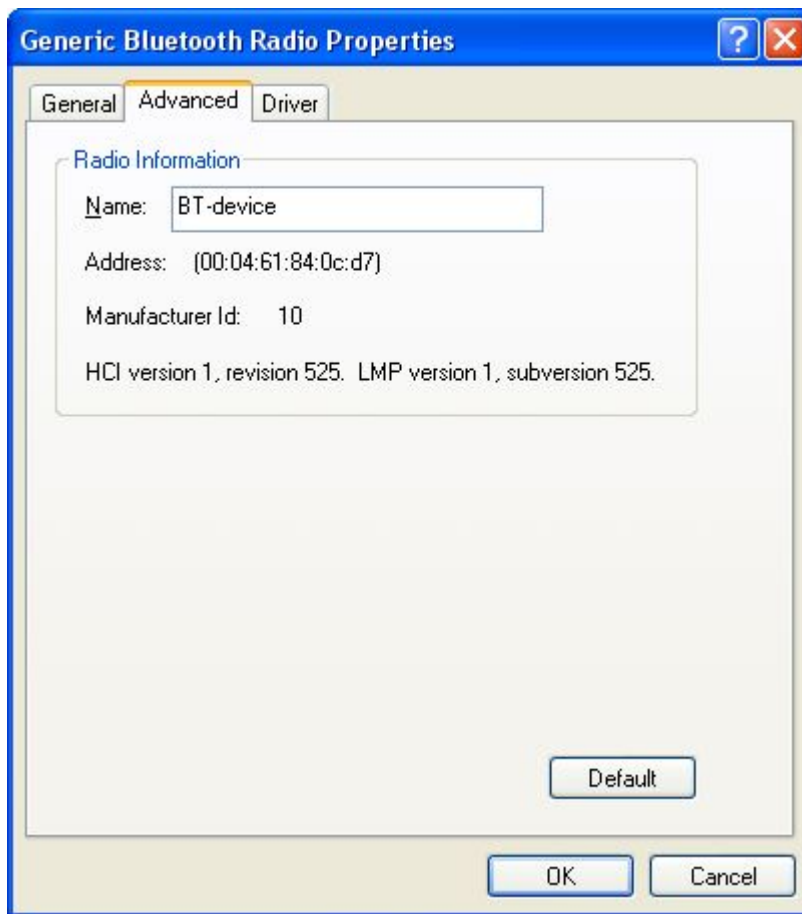


Figure 19 : Bluetooth Version 1.0 device

The nickname of the Bluetooth device can also be changed here. The nickname is usually shown when devices are listed in the user interface.

For compatibility, the HCI version shown is the important item. The screenshot shown in Figure 19 is from a device that supports version 1.0. Compare the differences with the screenshot shown in Figure 20.

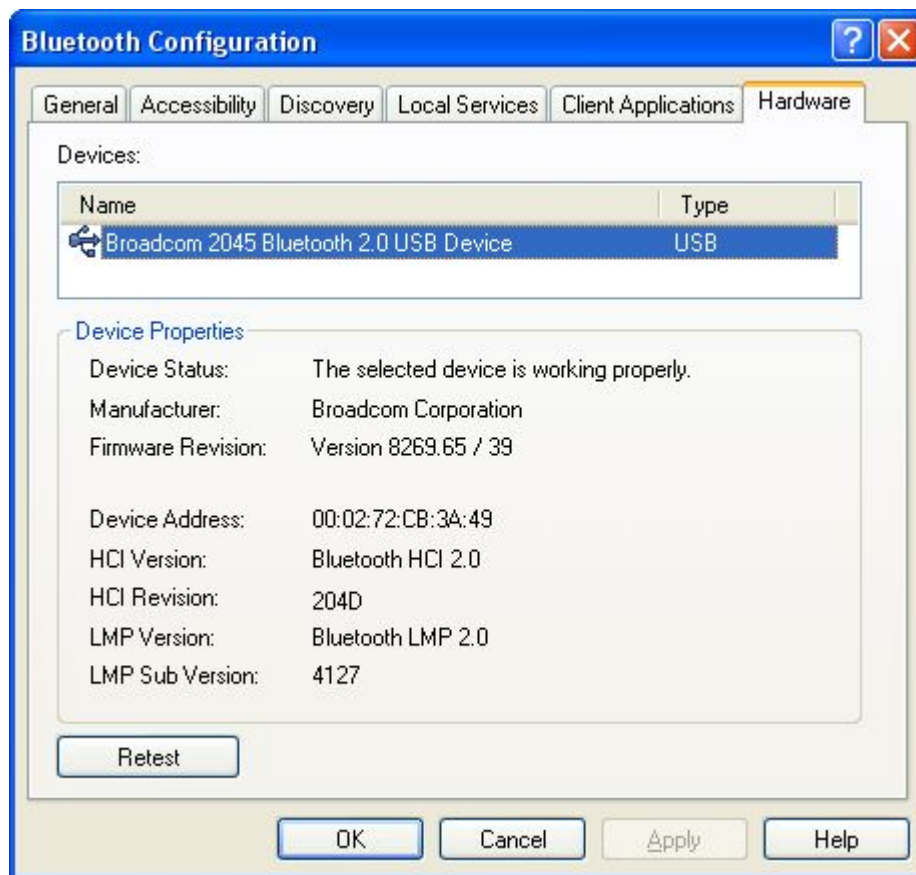


Figure 20 : Bluetooth Version 2.0 device

Bluetooth protocols are downwards compatible. This means the Bluetooth version with a higher number is compatible with the version with a lower number. For example, version 2.0 is compatible with version 1.0. However, version 1.0 may have incompatibilities with version 2.0. So, the device from Figure 20 can communicate with the device from Figure 19, but attempting communication in the other direction may cause problems.

6.3 Bluetooth stack implementation/API

Bluetooth stacks implement the protocol, so they should be able to handle and hide the version differences described in Section 6.2. However there are differences in features that different Bluetooth stacks support. If a certain Bluetooth stack does not support L2CAP connections, it naturally cannot create L2CAP connections with other Bluetooth stacks.

6.4 Libraries and other software implemented over the Bluetooth stack

Unfortunately there are big differences between the APIs and features of different types of Bluetooth software and libraries. There is no common Bluetooth API in the PC/J2SE world, so every manufacturer/implementer creates its own API. This makes it very difficult for upper-level software to communicate with several types of Bluetooth software and libraries. For example, APIs for Windows XP Bluetooth software and Widcomm software are completely different.

In the Java ME world there is the JSR-82 API (see Chapter 4, "JSR-82-based solution") defined for Bluetooth. Currently that API seems to be the only sensible way to implement Bluetooth applications on the PC in a compatible way.

If there is no need for enhanced functionality, another way is using virtual COM ports, as described in Chapter 3, "COM-based solution."

7 References

- [1] [MIDP: Bluetooth API Developer's Guide](http://www.forum.nokia.com) available at <http://www.forum.nokia.com>
- [2] Bluetooth API, JSR 82: *Java™ APIs for Bluetooth*, Java Community Process, 2002
<http://www.jcp.org/en/jsr/detail?id=82>

8 Evaluate this resource

Please spare a moment to help us improve documentation quality and recognize the resources you find most valuable, by [rating this resource](#).