

---

---

# Bluetooth

## *Part 8: The JSR-82 API for Device Discovery*

**Kjell Jørgen Hole**

UiB



Last updated 29.03.08

Mail: [Kjell.Hole@ii.uib.no](mailto:Kjell.Hole@ii.uib.no)

URL: [www.kjhole.com](http://www.kjhole.com)

- Device discovery overview
- Class of device records
- Accessing the local device
- Simple device discovery
- Device discovery via inquiry

- [Device discovery](#) and [service discovery](#) are two separate processes. We only consider device discovery for now
- Device discovery is
  - a process that finds the other devices in the area
  - also known as an *inquiry*
- A device responds to an inquiry with its Bluetooth address and [class of device record](#) that describes the type of Bluetooth device, as well as the types of available services

- The class of device record
  - specifies the type of device responding and the services available on this device
  - is made up of the
    1. *major service class*
    2. *major device class*
    3. *minor device class*

# Major Service Classes

---

KJhole.com

- The major service classes define the services available on a device:

Positioning

Information

Networking

Limited discoverable

Capturing

Rendering

Object transfer

Telephony

Audio

- A device can have multiple major *service* classes, i.e., a device can run multiple services at the same time

# Major and Minor Device Classes

---

- A device can only have *one* major device class

Computer

Phone

LAN/network access point

Audio/video

Peripheral

Imaging

Miscellaneous

Wearable

Toy

Medical

- The minor device classes are defined on the basis of the major device classes
  - a minor device class is a more specific description of the device

## Example: Minor Device Class

---

KJhole.com

- Assume that a device is classified as an imaging device in the major device class
- The *minor* device class then specifies whether the device is a camera, scanner, or printer
- The minor device class makes it possible to perform service searches on only the devices that fit a particular need

- The `DeviceClass` provides three methods to access the class of device record:

`int getServiceClasses()`—Retrieves the major service classes. A device may have multiple major service classes. When this occurs, the major service classes are bitwise OR'ed together

`int getMajorDeviceClass()`—Retrieves the major device class. A device may have only a single major device class

`int getMinorDeviceClass()`—Retrieves the minor device class. Must be interpreted on the basis of the major device class

# Check for Major Service Class

---

KJhole.com

- A major service class is represented by setting a bit in the class of device record. As an example, the rendering service class is represented by bit 18

```
boolean checkForRenderingService(DeviceClass d) {  
    if (d.getServiceClasses() & 0x40000 != 0)  
        return true;  
    else  
        return false;  
}
```

## Some Major Service Classes Defined by SIG

<b>Service Class</b>	<b>Type of Service</b>	<b>Bit Number</b>	<b>Hex Value</b>
Limited Discov. Mode	Device in this mode	13	0x2000
Positioning	Location identification	16	0x10000
Networking	LAN, ad hoc	17	0x20000
Rendering	Printing, speaker	18	0x40000
Capturing	Scanner	19	0x80000
Audio	headset service, etc.	21	0x200000
Telephony	Cordless phone, modem, headset	22	0x400000

# Check for Device Class

---

KJhole.com

- The major device class “Imaging” is represented by the hex value 0x600

```
boolean checkForImaging(DeviceClass d) {  
    if (d.getMajorDeviceClass() == 0x600)  
        return true;  
    else  
        return false;  
}
```

## Some Major Device Classes Defined by SIG

<b>Major Device Class</b>	<b>Example</b>	<b>Hex Value</b>
Computer	Desktop, laptop, PDA	0x100
Phone	Cellular, cordless, modem	0x200
LAN/AP		0x300
Audio/video	Headset, speaker, video display, VCR	0x400
Peripheral	Mouse, joystick, keyboard	0x500
Imaging	Printer, scanner, camera, display	0x600

# Types of Inquiries

---

KJhole.com

- There are two types of inquiries:

**General** A general discoverable device responds only to general inquiries

**Limited** A limited discoverable device responds to both general and limited inquiries

- A device that is not discoverable will not answer any inquiry

- JSR-82 allows an application to retrieve a list of devices that would likely be in the area without carrying out an inquiry
  - saves time and power
- There are two types of these *predefined devices*:

**Pre-known** Device which the local device frequently interacts.  
The device is set in the BCC (Bluetooth Control Center)

**Cached** Device found with a previous inquiry performed by some application

## Predefined Devices ...

---

KJhole.com

- There is no guarantee that a pre-known device will be available
- How many devices are cached and for how long is implementation dependent

- LocalDevice class provides access to and control of the local Bluetooth device
- Private constructor ensures that there is only a single LocalDevice object for each device
- An application calls the LocalDevice.getLocalDevice() method to retrieve the LocalDevice object (see example in previous lecture)
  - may throw a BluetoothStateException if the Bluetooth stack or radio is not working

# LocalDevice Methods

---

KJhole.com

<b>Method</b>	<b>Purpose</b>
<code>String getAddress()</code>	Retrieves the Bluetooth address of the local device
<code>DeviceClass getDeviceClass()</code>	Retrieves the DeviceClass object that represents the service classes, major device class, and minor device class
<code>int getDiscoverable()</code>	Retrieves the local device's discoverable mode (GIAC, LIAC, NOT_DISCOVERABLE)
<code>DiscoveryAgent getDiscoveryAgent()</code>	Returns the discovery agent for this device
<code>String getFriendlyName()</code>	Retrieves the name of the local device
<code>LocalDevice getLocalDevice()</code>	Retrieves the LocalDevice object for the local Bluetooth device
<code>String getProperty()</code>	Retrieves Bluetooth system properties
<code>ServiceRecord getRecord()</code>	Gets the service record corresponding to a btspn notifier

---

---

Method	Purpose
<code>boolean setDiscoverable()</code>	Sets the discoverable mode of the device
<code>void updateRecord()</code>	Updates the service record in the local SDDB that corresponds to the ServiceRecord parameter

---

- The `DiscoveryAgent` class contains the constants:
  - `DiscoveryAgent.GIAC` (*General Inquiry Access Code*)
  - `DiscoveryAgent.LIAC` (*Limited Inquiry Access Code*)
  - `DiscoveryAgent.NOT_DISCOVERABLE`
- The following program demonstrates the use of these constants, as well as the `LocalDevice.getProperty()` method

```

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.bluetooth.*;

public class BluetoothInfoMIDlet extends BluetoothMIDlet {

    public void startApp() throws MIDletStateChangeException {
        Display currentDisplay = Display.getDisplay(this); //Display manager
        Form infoForm = new Form("Device Info"); // Screen
        currentDisplay.setCurrent(infoForm); // Set current screen
        getBluetoothInfo(infoForm); // To be introduced
        infoForm.addCommand(new Command("Exit",Command.EXIT,1));
        infoForm.setCommandListener(this); // Can receive high-level
    } // events from the implementation
}

```

```
private void getBluetoothInfo(Form f) {
    LocalDevice local = null;
    try {
        local = LocalDevice.getLocalDevice();
    } catch (BluetoothStateException e) {
        f.append("Failed to retrieve the local device (" + e.getMessage() + ")");
        return;
    }
    f.append("Address: " + local.getBluetoothAddress() + "\n");
    String name = local.getFriendlyName();
    if (name == null)
        f.append("Name: Failed to Retrieve");
        else f.append("Name: " + name);
    f.append("\n");
}
```

```
int mode = local.getDiscoverable();
StringBuffer text = new StringBuffer("Discoverable Mode: ");
switch (mode) {
    case DiscoveryAgent.NOT_DISCOVERABLE:
        text.append("Not Discoverable"); break;
    case DiscoveryAgent.GIAC:
        text.append("General"); break;
    case DiscoveryAgent.LIAC:
        text.append("Limited"); break;
    default:
        text.append("0x");
        text.append(Integer.toString(mode,16)); break;
}
f.append(text.toString() + "\n");
```

```
f.append("API Version:" +
    LocalDevice.getProperty("bluetooth.api.version")+"\n");
f.append("Master Switch Supported:" +
    LocalDevice.getProperty("bluetooth.master.switch")+"\n");
f.append("Max Attributes:" + LocalDevice.getProperty(
    "bluetooth.sd.attr.retrievable.max")+"\n");
f.append("Max Connected Devices:" +
    LocalDevice.getProperty("bluetooth.connected.devices.max")+"\n");
f.append("Max Receive MTU:" +
    LocalDevice.getProperty("bluetooth.l2cap.receiveMTU.max")+"\n");
    :
}
}
```

- We first show how to retrieve *pre-known* and *cached* devices
- Each device has a single `javax.bluetooth.DiscoveryAgent` object that provides methods to initiate device (and service) searches
  - the method `LocalDevice.getDiscoveryAgent()` is used to retrieve the object
  - multiple calls to this method return the same object

# Retrieve DiscoveryAgent Object

---

KJhole.com

```
public static DiscoveryAgent getLocalDiscoveryAgent() {  
    try {  
        // getLocalDevice may throw an exception  
        LocalDevice local = LocalDevice.getLocalDevice();  
        DiscoveryAgent agent = local.getDiscoveryAgent();  
        return agent;  
    } catch (BluetoothStateException e) { return null; }  
}
```

- The `RemoteDevice` class represents a remote Bluetooth device
- It provides basic information about a remote device including the device's Bluetooth address and its friendly name
- We will use the method `DiscoveryAgent.retrieveDevices()` to obtain lists of pre-known and cached `RemoteDevices` (see following example)

```
import java.util.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.bluetooth.*;

public class DiscoveryMIDlet extends BluetoothMIDlet {
    private List deviceList;    // List of remote devices on screen
    private DiscoveryAgent agent; // DiscoveryAgent for the local device

    public void startApp() throws MIDletStateChangeException {
        deviceList = new List("List of Devices",List.IMPLICIT);
        deviceList.addCommand(new Command("Exit",Command.EXIT,1));
        deviceList.setCommandListener(this);
        Display.getDisplay(this).setCurrent(deviceList);
    }
}
```

```
try {
    LocalDevice local = LocalDevice.getLocalDevice();
    agent = local.getDiscoveryAgent();
} catch (BluetoothStateException e) {
    // Prevent app from starting if the Bluetooth
    // device could not be retrieved
    throw new MIDletStateChangeException(
        "Unable to retrieve local Bluetooth device.");
}
addDevices(); // to be introduced
}
```

```

private void addDevices() {
    RemoteDevice[] list = agent.retrieveDevices(DiscoveryAgent.PREKNOWN)
    if (list != null) {
        for (int i=0; i<list.length; i++) {
            String address = list[i].getBluetoothAddress();
            deviceList.insert(0,address+"-P",null); } // at start of list
        }
    list = agent.retrieveDevices(DiscoveryAgent.CACHED);
    if (list != null) {
        for (int i=0; i<list.length; i++) {
            String address = list[i].getBluetoothAddress();
            deviceList.insert(0,address+"-C",null); }
        }
    }
}

```

- An application must implement the `DiscoveryListener` interface methods `deviceDiscovered()` and `inquiryCompleted()` to request an inquiry

```
public void deviceDiscovered(RemoteDevice dev, DeviceClass cod)
```

```
public void inquiryCompleted(int discType)
```

- The `DiscoveryAgent.startInquiry()` method is called to initiate a general or a limited inquiry. The `DiscoveryAgent.cancelInquiry()` method cancels an inquiry

## deviceDiscovered()

---

KJhole.com

- `deviceDiscovered(RemoteDevice dev, DeviceClass cod)` is called when a device is found during an inquiry. An inquiry searches for devices that are discoverable. The same device may be returned multiple times

`dev`—the device that was found during the inquiry

`cod`—the major service classes, major device class, and minor device class of the remote device

- The method `inquiryCompleted(int discType)` is called when an inquiry is completed

`discType`—the type of request that was completed; either

\* `DiscoveryListener.INQUIRY_COMPLETED`

\* `DiscoveryListener.INQUIRY_TERMINATED`

\* `DiscoveryListener.INQUIRY_ERROR`

- The following program starts an inquiry and displays the Bluetooth addresses of all responding devices

```
public class DiscoveryMIDlet extends BluetoothMIDlet
    implements DiscoveryListener {

    private List deviceList;
    private DiscoveryAgent agent;

    public void startApp() throws MIDletStateChangeException {
        : // Code from previous example finds pre-known and cached devices
        addDevices();
        try {
            agent.startInquiry(DiscoveryAgent.GIAC, this);
        } catch (BluetoothStateException e) {
            throw new MIDletStateChangeException("Unable to start the inquiry")
        }
    }
}
```

```
private void addDevices() {
    :    // Code from previous example goes here
}

public void deviceDiscovered(RemoteDevice device, DeviceClass cod) {
    String address = device.getBluetoothAddress();
    deviceList.insert(0, address + "-I", null);
}

public void servicesDiscovered(int transID, ServiceRecord[] record) {
}

public void serviceSearchCompleted(int transID, int type) {
}
```

```
public void inquiryCompleted(int type) {
    Alert dialog = null;
    // Determine if an error occurred.  If one did occur display
    // an Alert before allowing the application to exit
    if (type != DiscoveryListener.INQUIRY_COMPLETED){
        dialog = new Alert("Bluetooth Error",
            "The inquiry failed to complete normally",
            null, AlertType.ERROR);
    } else {
        dialog = new Alert("Inquiry Completed",
            "The inquiry completed normally", null,AlertType.INFO);
    }
    dialog.setTimeout(Alert.FOREVER);
    Display.getDisplay(this).setCurrent(dialog);
}
```

```
public void commandAction(Command c, Displayable d) {  
    if (c.getCommandType() == Command.EXIT) {  
        // Try to cancel the inquiry  
        agent.cancelInquiry(this);  
        notifyDestroyed();  
    }  
}  
}
```

# Information from Remote Device

---

- A remote device is represented by a `javax.bluetooth.RemoteDevice` object
- An application cannot directly instantiate a new object since there is no public constructor
- There are three ways of obtaining a `RemoteDevice` object. First, an object can be obtained via the device discovery process, as we have seen

## Second Method for RemoteDevice Initiation

- Method that extends RemoteDevice can be instantiated:

```
public class MyRemoteDevice extends RemoteDevice {  
  
    // Creates new RemoteDevice object based upon provided address  
    public MyRemoteDevice(String address) {  
        super(address);  
    }  
}
```

- The Bluetooth device address must be 12 hex characters with no preceding "0x"

## Third Method for RemoteDevice Initiation

- Use the static method

```
public static RemoteDevice getRemoteDevice  
(javax.microedition.io.Connection conn) throws java.io.IOException  
    conn—connection to remote Bluetooth device
```

- After the RemoteDevice object is retrieved, the following methods may be called:

```
getBluetoothAddress()
```

```
isTrustedDevice()
```

```
getFriendlyName()
```

- The RemoteDevice class provides the methods:

`boolean authenticate()` throws `java.io.IOException`—Authenticates the remote device. Can cause pairing to occur

`boolean encrypt(javax.microedition.io.Connection conn, boolean on)` throws `java.io.IOException`—Turns encryption on and off. Will first authenticate this RemoteDevice if it has not already been authenticated

`public boolean authorize(javax.microedition.io.Connection conn)` throws `java.io.IOException`—Requests the BCC to authorize the connection

- The `RemoteDevice` class also have three methods to determine the security level on a connection:

`boolean isAuthenticated()` — Determines if this `RemoteDevice` has been authenticated

`boolean isEncrypted()` — Determines if data exchanges with this `RemoteDevice` are currently being encrypted

`boolean isAuthorized(javax.microedition.io.Connection conn)`  
throws `java.io.IOException` — Determines if this `RemoteDevice` has been authorized previously by the BCC of the local device

\* connection based, multiple connections can exist over single link

- Device discovery, or *inquiry*, is a key part of the JSR-82 API
- There are two types of inquiry: general and limited
- There are three types of discoverable mode: not discoverable, limited discoverable, and general discoverable mode
- Device discovery is started using one of the methods

```
DiscoveryAgent.retrieveDevices()
```

```
DiscoveryAgent.startInquiry()
```

## Summary (2)

---

- An application calling `startInquiry()` must implement the `DiscoveryListener` interface methods

`deviceDiscovered()`

`inquiryCompleted()`

- `LocalDevice` class provides access to the local Bluetooth device
- A remote device is represented by a `RemoteDevice` object