

Reviewer Approved



Reviewer Approved



To use the Bluetooth protocol on Java ME application your mobile device must implement the JSR 82 Bluetooth API. One of the main challenges on Bluetooth applications is to find out the connection address, for devices with an specific services available on them.

In the following sample we are going to see how to implement this code using an class called **BtManager**.

Let's start creating the class and some variables to store our results.

```
import java.util.Vector;
// bluetooth classes
import javax.bluetooth.DeviceClass;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.UUID;

public class BTManager implements DiscoveryListener {
    // used to store devices we found
    public Vector btDevicesFound;
    // used to store service information for each device
    public Vector btServicesFound;

    public BTManager() {
        btDevicesFound = new Vector();
        btServicesFound = new Vector();
    }

    public int find(UUID[] aServices){
        // search for devices
        findDevices();
        // search for services in the devices found
        findServices(aServices);
        return btDevicesFound.size();
    }
}
```

As you can see we are going to implement the DiscoveryListener interface. This interface provides all the callbacks for our bluetooth calls. Let's start for searching for devices:

```
public int findDevices() {
    try {
        // cleans previous elements
        btDevicesFound.removeAllElements();
        // resets status variable
        isBTSearchComplete = false;
        LocalDevice local = LocalDevice.getLocalDevice();
        DiscoveryAgent discoveryAgent = local.getDiscoveryAgent();
        // start discovery of new devices
        discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
        while ((!isBTSearchComplete)) {
            //waits for a fixed time, to avoid long search
            synchronized (this) {
                this.wait(BTManager.BLUETOOTH_TIMEOUT);
            }
            // check if search is completed
        }
    }
}
```

How_to_search_for_Bluetooth_devices_and_services

```
        if (!isBTSearchComplete) {
            // search no yet completed so let's cancel it
            discoveryAgent.cancelInquiry(this);
        }
    }
} catch (Exception e) {
    e.printStackTrace();
}
// returns the number of devices found
return btDevicesFound.size();
}

public void deviceDiscovered(RemoteDevice remoteDevice,
    DeviceClass deviceClass) {
    btDevicesFound.addElement(remoteDevice);
}

public void inquiryCompleted(int param) {
    isBTSearchComplete = true;
    // notifies and wake main thread that device search is completed
    synchronized (this) {
        this.notify();
    }
}
```

Now that we have a list of devices, let's search for the services we want:

```
public void findServices(UUID[] aServices) {
    // cleans previous elements
    btServicesFound.removeAllElements();
    try {
        LocalDevice local = LocalDevice.getLocalDevice();
        DiscoveryAgent discoveryAgent = local.getDiscoveryAgent();
        // discover services
        if (btDevicesFound.size() > 0) {
            for (int i = 0; i < btDevicesFound.size(); i++) {
                isBTSearchComplete = false;
                // adds a null element in case we don't found service
                btServicesFound.addElement(null);
                int transID = discoveryAgent.searchServices(null, aServices,
                    (RemoteDevice) (btDevicesFound.elementAt(i)), this);
                // wait for service discovery ends
                synchronized (this) {
                    this.wait(BTManager.BLUETOOTH_TIMEOUT);
                }
                if (!isBTSearchComplete) {
                    discoveryAgent.cancelServiceSearch(transID);
                }
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void servicesDiscovered(int param, ServiceRecord[] serviceRecord) {
    int index = btServicesFound.size() - 1;
    for (int i = 0; i < serviceRecord.length; i++) {
        btServicesFound.setElementAt(serviceRecord[i], index);
    }
}
```

How_to_search_for_Bluetooth_devices_and_services

```
public void serviceSearchCompleted(int transID, int respCode) {
    isBTSearchComplete = true;
    // notifies and wake mains thread that service search is completed
    synchronized (this) {
        this.notify();
    }
}
```

After we got all the data stored in our Vectors, we just need to add two methods to retrieve information, one to retrieve the [Bluetooth](#) Url and another [Bluetooth](#) user friendly name, after all [Bluetooth](#) urls, don't have a lot of meaning to the end users of our application.

```
public String getDeviceName(int deviceID) {
    try {
        return ((RemoteDevice) btDevicesFound.elementAt(deviceID))
            .getFriendlyName(false);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return "Error";
}

public String getServiceURL(int deviceID) {
    try {
        return ((ServiceRecord) btServicesFound.elementAt(deviceID))
            .getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return "Error";
}
```

And with those last methods we completed our [Bluetooth](#) finder class. You can check the full source code where I provide a example of how to use it to search for [Bluetooth GPS](#).

Downloads

- [Full Source Code](#)
- [Jad File](#)
- [Jar File](#)