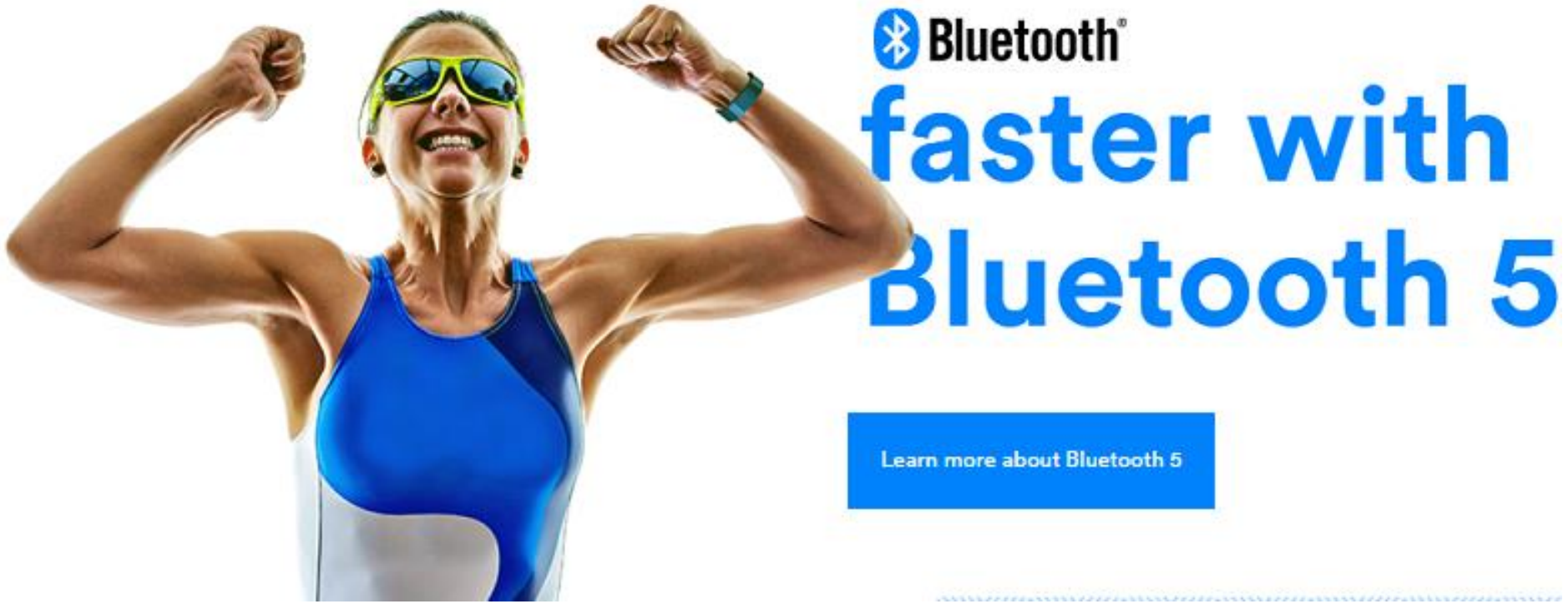
Bluetooth
&
Wifi

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

Bluetooth



Courtesy: http://www.bluetooth.com

Bluetooth - Introduction

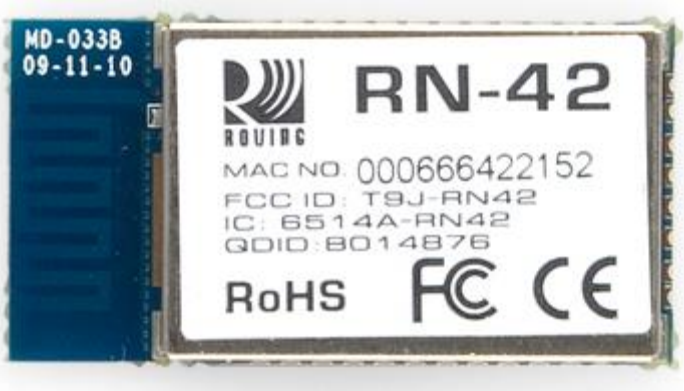
- **Bluetooth** is a wireless technology standard for exchanging data over short distances
- Invented by telecom vendor Ericsson in 1994
- Bluetooth operates at frequencies between 2402 and 2480 MHz, or 2400 and 2483.5 MHz including guard bands 2 MHz wide at the bottom end and 3.5 MHz wide at the top
- Communication between Bluetooth devices happens over short-range, ad hoc networks known as piconets. A piconet is a network of devices connected using Bluetooth technology.
- Bluetooth is a packet-based protocol with a master-slave structure
- One master may communicate with up to seven slaves in a piconet

Bluetooth - Introduction

- The Bluetooth Core Specification mandates a range of not less than 10 metres (33 ft), but there is no upper limit on actual range
- Bluetooth flavors
  - Bluetooth Basic Rate/Enhanced Data Rate(BR/EDR)
  - Bluetooth Low Energy(LE)
- Bluetooth profiles:Wireless interface specification for Bluetooth-based communication between devices eg
  - Handsfree headset
  - Wireless speakers
  - Set top box
  - Tele health

Bluetooth Addresses and Names

- Every single Bluetooth device has a unique 48-bit address, commonly abbreviated BD_ADDR. This will usually be presented in the form of a 12-digit hexadecimal value. The most-significant half (24 bits) of the address is an organization unique identifier (OUI), which identifies the manufacturer. The lower 24-bits are the more unique part of the address.



- This address should be visible on most Bluetooth devices. For example, on this RN-42 Bluetooth Module, the address printed next to “MAC NO.” is 000666422152:

Bluetooth Addresses and Names

- The “000666” portion of that address is the OUI of Roving Networks, the manufacturer of the module. Every RN module will share those upper 24-bits. The “422152” portion of the module is the more unique ID of the device.
- Bluetooth devices can also have user-friendly names given to them. These are usually presented to the user, in place of the address, to help identify which device it is.

Bluetooth - Classes

In Android, Bluetooth devices and connections are handled by the following classes:

- **BluetoothAdapter** The Bluetooth Adapter represents the local Bluetooth device—that is, the Android device on which your application is running.
- **BluetoothDevice** Each remote device with which you wish to communicate is represented as a BluetoothDevice.
- **BluetoothSocket** Call createRfcommSocketToServiceRecord on a remote Bluetooth Device object to create a Bluetooth Socket that will let you make a connection request to the remote device, and then initiate communications.
- **BluetoothServerSocket** By creating a Bluetooth Server Socket (using the listenUsingRfcommWithServiceRecord method) on your local Bluetooth Adapter, you can listen for incoming connection requests from Bluetooth Sockets on remote devices.
- **BluetoothClass:** Describes the general characteristics and capabilities of a Bluetooth device
- **BluetoothProfile**:An interface that represents a Bluetooth profile

Permission

- android.permission.BLUETOOTH
- android.permission.BLUETOOTH_ADMIN

- Both are normal permissions

Accessing the Local Bluetooth Device Adapter

- BluetoothAdapter bluetooth = BluetoothAdapter.getDefaultAdapter();

Local Bluetooth Device – Check Status

- isEnabled()
- String toastText;
- if (bluetooth.isEnabled()) {

```
String address = bluetooth.getAddress();
String name = bluetooth.getName();
  toastText = name + " : " + address;
}
else
 toastText = "Bluetooth is not enabled";
Toast.makeText(this, toastText,
  Toast.LENGTH_LONG).show();
```

Naming Bluetooth Device

- If you also have the BLUETOOTH_ADMIN permission you can change the friendly name of the Bluetooth Adapter using the setName method..

```
bluetooth.setName(“sisoftBlue");
```

Local Bluetooth Device - State

- To find a more detailed description of the current Bluetooth Adapter state, use the **getState** method, which will return one of the following BluetoothAdapter constant.
  - **STATE_TURNING_ON**
  - **STATE_ON**
  - **STATE_TURNING_OFF**
  - **STATE_OFF**
- By default the Bluetooth adapter will be turned off. In order to conserve battery life and optimize security, most users will keep Bluetooth disabled unless it’s in use.

Local Bluetooth Device – Enable

- To enable the Bluetooth Adapter you can start a system sub-Activity using the ACTION_REQUEST_ENABLE Bluetooth Adapter static constant as a startActivityForResult action string:

```
String enableBT =
    BluetoothAdapter.ACTION_REQUEST_ENABLE;
startActivityForResult(new Intent(enableBT), 0);
```

Local Bluetooth Device – Enable

- It is also possible to turn the Bluetooth Adapter on and off directly, using the enable and disable methods, if you include the BLUETOOTH_ADMIN permission in your manifest.
- Note that this should be done only when absolutely necessary and that the user should always be notified if you are manually changing the Bluetooth Adapter status on the user's behalf.

Local Bluetooth Device-Discoverable

- The Bluetooth Adapter's discoverability is indicated by its scan mode.
- To get the scan mode by calling **getScanMode** on the BluetoothAdapter object.
- It will return one of the following BluetoothAdapter constants: .
  - SCAN_MODE_CONNECTABLE_DISCOVERABLE
  - SCAN_MODE_CONNECTABLE
  - SCAN_MODE_NONE

Local Bluetooth Device-Discoverable

- String aDiscoverable = **BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE;**
- startActivityForResult(new Intent(**aDiscoverable),**DISCOVERY_REQUEST);

Local Bluetooth Device-Discoverable

- By default discoverability will be enabled for two minutes. You can modify this setting by adding an EXTRA_DISCOVERABLE_DURATION extra to the launch Intent, specifying the number of seconds you want discoverability to last.

Bluetooth Device-Discovery

- The process of two devices finding each other in order to connect is called *discovery.*
- *Before you can* establish a Bluetooth Socket for communications, the local Bluetooth Adapter must bond with the remote device.
- Before two devices can bond and connect, they first need to discover each other.

Discovering Remote Bluetooth Device

- You can check to see if the local Bluetooth Adapter is already performing a discovery scan using the isDiscovering method.
- To initiate the discovery process call startDiscovery on the Bluetooth Adapter. To cancel a discovery in progress call cancelDiscovery.
- bluetooth.startDiscovery();
- bluetooth.cancelDiscovery();

Discovering Remote Bluetooth Device

- The discovery process is asynchronous
- In order to receive information about each device discovered, your application must register a BroadcastReceiver for the ACTION_FOUND intent The system broadcasts this intent for each device.
- The intent contains the extra fields EXTRA_DEVICE and EXTRA_CLASS, which in turn contain a BluetoothDevice and a BluetoothClass, respectively

Connected and Paired Devices

- To be *paired* means that two devices are aware of each other's existence, have a shared link-key that can be used for authentication, and are capable of establishing an encrypted connection with each other.
- To be *connected* means that the devices currently share an RFCOMM channel and are able to transmit data with each other. The current Android Bluetooth API's require devices to be paired before an RFCOMM connection can be established. Pairing is automatically performed when you initiate an encrypted connection with the Bluetooth APIs.

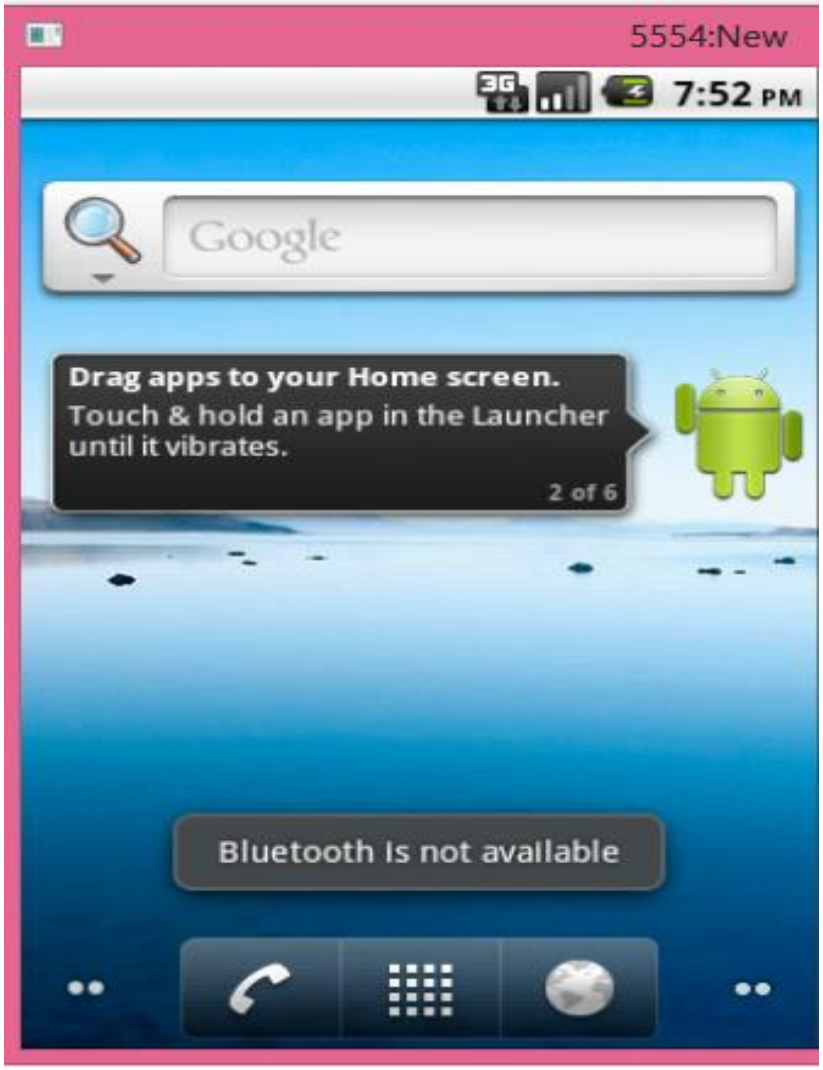
Querying Paired devices

It's worth querying the set of paired devices to see if the desired device is already known. To do so, call getBondedDevices(). This will return a set of BluetoothDevice objects representing paired devices.

```
Set<BluetoothDevice> pairedDevices;
ArrayList list = new ArrayList();
pairedDevices = myBluetooth.getBondedDevices();
for(BluetoothDevice bt : pairedDevices)
{
      list.add(bt.getName() + "\n" + bt.getAddress());
 //Get the device's name and the address
         }
```

Connecting devices

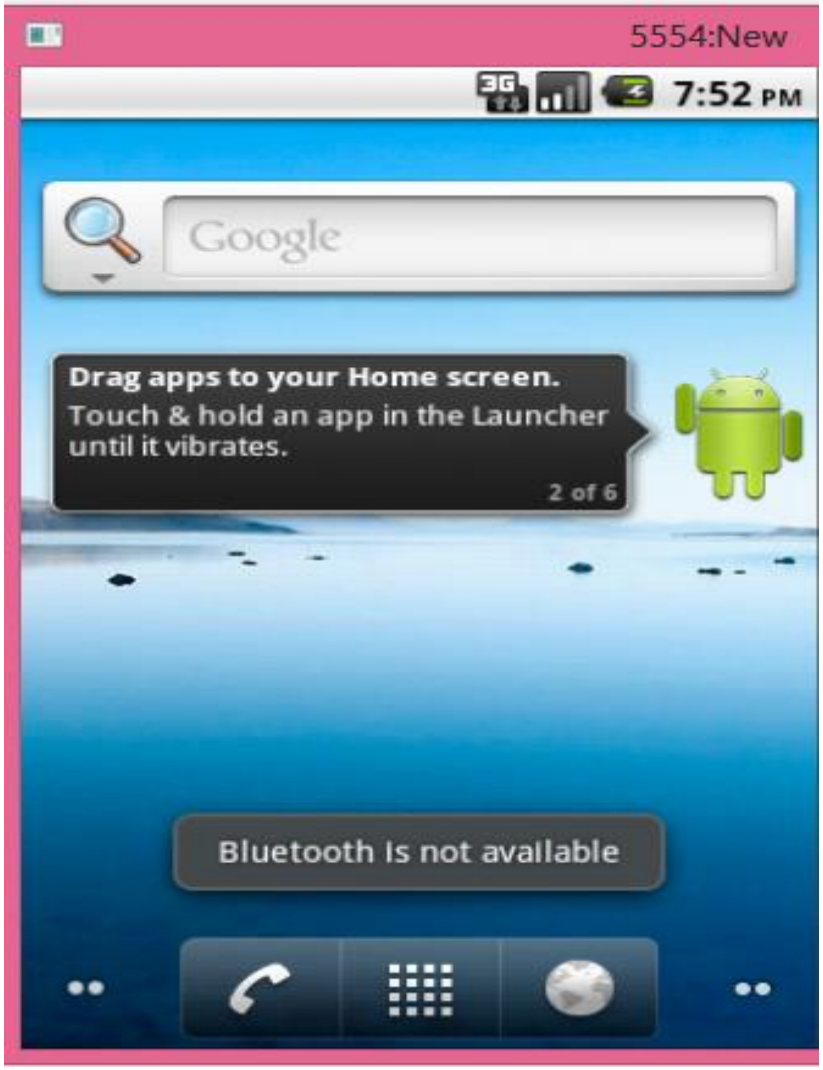
- To get a BluetoothDevice, use BluetoothAdapter.getRemoteDevice(String) to create one representing a device of a known MAC address (which you can get through device discovery with BluetoothAdapter) or get one from the set of bonded devices returned by BluetoothAdapter.getBondedDevices().
- You can then open a BluetoothSocket for communication with the remote device, using createRfcommSocketToServiceRecord(UUID).


5554:New
7:52 PM
Google
Drag apps to your Home screen.
Touch & hold an app in the Launcher until it vibrates.
2 of 6
Bluetooth is not available

Wi-fi

- Wireless local area networking with devices based on the IEEE 802.11 standards

Courtesy: http://www.wi-fi.org


5554:New
7:52 PM
Google
Drag apps to your Home screen.
Touch & hold an app in the Launcher until it vibrates.
2 of 6
Bluetooth is not available

WifiManager

- This class provides the primary API for managing all aspects of Wi-Fi connectivity. Get an instance of this class by calling Context.getSystemService(Context.WIFI_SERVICE). It deals with several categories of items:
- The list of configured networks. The list can be viewed and updated, and attributes of individual entries can be modified.
- The currently active Wi-Fi network, if any. Connectivity can be established or torn down, and dynamic information about the state of the network can be queried.
- Results of access point scans, containing enough information to make decisions about what access point to connect to.
- It defines the names of various Intent actions that are broadcast upon any sort of change in Wi-Fi state.

Wi-Fi Peer-to-Peer

- Wi-Fi peer-to-peer (P2P) allows Android 4.0 (API level 14) or later devices with the appropriate hardware to connect directly to each other via Wi-Fi without an intermediate access point
- Android's Wi-Fi P2P framework complies with the Wi-Fi Alliance's Wi-Fi Direct™ certification program
- The Wi-Fi P2P APIs consist of the following main parts:
  - Methods that allow you to discover, request, and connect to peers are defined in the WifiP2pManager class.
  - Listeners that allow you to be notified of the success or failure of WifiP2pManager method calls. When calling WifiP2pManager methods, each method can receive a specific listener passed in as a parameter.
  - Intents that notify you of specific events detected by the Wi-Fi P2P framework, such as a dropped connection or a newly discovered peer.